UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Felipe Alencar Lopes

Modelos em Tempo de Execução para Redes Definidas por Software Auto-Adaptáveis

Recife
2020

Felipe Alencar Lopes

Modelos em Tempo de Execução para Redes Definidas por Software Auto-Adaptáveis

Tese apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Doutor em Ciência da Computação.

Área de Concentração: Redes de Computadores

Orientador: Stênio Flávio de Lacerda Fernandes

Recife
2020

Catalogação na fonte
Bibliotecário Cristiano Cosme S. dos Anjos, CRB4-2290

L864m Lopes, Felipe Alencar
Modelos em tempo de execução para redes definidas por software auto-adaptáveis/ Felipe Alencar Lopes. – 2020.
150 f.: il., fig., tab.

Orientador: Stênio Flávio de Lacerda Fernandes.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndices.

1. Redes de Computadores e Sistemas Distribuídos. 2. Modelos em tempo de execução. 3. Redes definidas por software. 4. Gerenciamento autonômico. I. Fernandes, Stênio Flávio de Lacerda (orientador). II. Título

004.6 CDD (23. ed.) UFPE - CCEN 2021 – 142

Felipe Alencar Lopes

“Modelos em Tempo de Execução para Redes Definidas por Software Auto-Adaptáveis”

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação.

Aprovado em: 05/03/2020.

_____

Orientador: Prof. Dr. Stênio Flávio de Lacerda Fernandes

BANCA EXAMINADORA

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática

_____

Prof. Dr. Robson do Nascimento Fidalgo
Centro de Informática / UFPE

_____

Prof. Dr. Glauco Estácio Gonçalves
Departamento de Estatística e Informática / UFRPE

_____

Prof. Dr. Denio Mariz Timoteo de Sousa
Instituto Federal da Paraíba / Campus João Pessoa

_____

Prof. Dr. Rafael Thyago Antonello
Instituto Federal de Alagoas / Campus Palmeira dos Indios

À todos aqueles que acreditaram em mim. E aos que não acreditaram também.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer a Deus por este momento e por todas as vezes que Ele me mostrou um caminho a seguir, apesar de qualquer adversidade. Sou todo grato também à minha família pelo apoio incondicional, em especial: à minha mãe, a dona Rê, que nunca mediu esforços para cuidar da nossa família, abrindo mão e passando por cima de tanta coisa pra que a gente pudesse ter uma condição de vida um pouco melhor; à minha irmã, por ser sempre a minha referência de dedicação e integridade, por ser meu porto seguro em tantos momentos e por, no meio dessa trajetória, ter me dado o melhor presente que já ganhei: o meu sobrinho Miguel; e ao meu pai que, mesmo doente, me ensinou a nunca desistir da vida e a buscar ser alguém melhor – que ao ler esse agradecimento ele saiba que as noites mal dormidas rodando em um táxi não foram em vão.

Destaco também o quanto sou grato ao meu orientador, Stenio, não só pela oportunidade de ter sido seu aluno e orientando, mas também pelos ensinamentos muito além dos acadêmicos, pela paciência, pelos incentivos e conselhos fundamentais em momentos críticos dessa jornada. Eu realmente não sei onde estaria se não fosse a provocação feita há 7 anos atrás para que eu prosseguisse minha vida acadêmica.

Aproveito para mostrar a minha gratidão aos amigos que tenho desde os tempos de aluno no CEFET (Nicolas, Deivid, Paulinho) e aos que apareceram no meio do caminho (Cibele, Karla, Francisco, Jullyana, Marianna, Flávia, Caio, Karyne, Carlos, Laís). Vocês não têm ideia da importância que tiveram para que os dias mais difíceis fossem mais fáceis. Grato ainda aos amigos do IFAL (Felipe Piu, Ray, Marcela, Uziel, Handrik, Geraldo) e do CIn (Jean, Weslley, Michel, Rafael, Leônidas, Marcelo) pelos conselhos, conversas e que muitas vezes me motivaram a seguir em frente.

Agradeço mesmo àquelas pessoas que não se fazem mais presentes na minha vida, mas que também foram muito importantes na realização desta etapa.

Toda gratidão aos professores do Centro de Informática da UFPE, que foram fundamentais, aos colegas do IFAL que me deram todo suporte para conclusão deste trabalho, e aos colegas do Instituto de Tecnologia de Karlsruhe (KIT), na Alemanha, que me acolheram e contribuíram bastante para evolução da minha pesquisa, principalmente a professora Martina Zitterbart por ter me aceitado para o estágio de pesquisa no Instituto de Telemática, ter me coorientado e dado todo o apoio durante minha estadia.

Por fim, agradeço à FACEPE (processo IBPG-1200-1.03/14) por financiar este trabalho e ao projeto ERASMUS Be Mundus (projeto BM15DM0988) que financiou minha estadia de pesquisa fora do país.

RESUMO

O surgimento do paradigma de Redes Definidas por Software (SDN) permitiu diversas possibilidades de inovação no desenvolvimento e gerenciamento de redes. Entretanto, estas possibilidades não vieram, necessariamente, acompanhadas de facilidades para desenvolvedores e operadores de rede. Atividades como a criação de aplicações e o próprio gerenciamento destas redes muitas vezes envolve um baixo nível de abstração e um alto grau de complexidade e incerteza para os envolvidos. Linguagens de programação, algoritmos de roteamento e posicionamento de controlador são exemplos de categorias de propostas que visam contornar os problemas atuais das redes SDN. Entretanto, a literatura ainda não conta com uma solução que possa integrar os diversos avanços e o alto potencial de abstração e aplicação de inteligência artificial na área de SDN. Neste cenário, este trabalho (1) evolui uma técnica anterior para modelagem de aplicações de rede e (2) propõe um framework chamado MOSES que se baseia no conceito de Modelos em Tempo de Execução para prover gerenciamento autonômico em SDN. Seguindo conceitos da Engenharia de Software Baseada em Modelos, e arquiteturas de referência como MAPE-K e GANA, o MOSES é baseado em três pilares: i) a camada modelo de rede que permite a modelagem da rede desejável; ii) a camada de adaptabilidade, que realiza – através de algoritmos de Aprendizagem por Reforço Profunda – as mudanças necessárias na rede para atingir a lógica e comportamento definidos pelo modelo alvo; e iii) camada de rede composta pelas sub-camadas da arquitetura SDN. Além destas camadas, MOSES prevê a integração de blocos funcionais intermediários para a realização de tarefas como a geração de código, a reflexão do estado da rede na camada modelo de rede, e a análise das ações de rede a serem tomadas visando alcançar os objetivos modelados. Tal integração visa resolver os seguintes problemas: i) baixo nível de abstração no gerenciamento e desenvolvimento de SDN; e ii) complexidade ao lidar com infraestruturas heterogêneas. Para avaliar o MOSES, os blocos da arquitetura proposta foram implementados e verificados de acordo com a viabilidade em cenários de uso, visando identificar seus benefícios, utilizando a ferramenta Mininet para realizar simulações. Nos experimentos realizados, foram utilizadas topologias reais e simuladas interagindo com modelos de aplicações de balanceamento de carga, monitoramento e QoS. Os resultados mostram que os blocos funcionais do MOSES permitem: i) a validação semântica de aplicações SDN, ii) o aumento do nível de abstração e compatibilidade (este último a um custo de 15% de sobrecarga); e iii) a garantia dos níveis de QoS fazendo uso unicamente de algoritmos de aprendizagem de máquina e modelos ao invés de técnicas tradicionais de enfileiramento, os quais melhoraram taxas de atraso em até 22% quando comparados com implementações padrões.

Palavras-chave: modelos em tempo de execução; redes definidas por software; gerenciamento autonômico.

ABSTRACT

The emergence of the Software-Defined Networks (SDN) paradigm enabled several possibilities for innovation in the development and management of networks. However, these possibilities were not necessarily accompanied by facilities for developers and network operators. Activities such as the creation of applications and the actual management of these networks often involve a low level of abstraction, and a high degree of complexity and uncertainty for those involved. Programming languages, routing algorithms, controller placement techniques, and mechanisms to ensure Quality of Service (QoS) are examples of proposals categories that aim to address the current problems of SDN networks. However, although there are several approaches to overcome these problems, the literature does not yet have a solution that can integrate the several advances and the high potential of abstraction and computational intelligence involved in the SDN area. In this scenario, to soften the problems involved mainly in the development and management activities of SDN, this work improves a previous approach on modeling *Software-Defined Networking* (SDN) applications and proposes a framework called MOSES that is based on autonomic runtime models to provide autonomic management in SDN. Guided by concepts of Model-Driven Engineering, and following reference architectures such as MAPE-K and GANA, MOSES is based on three pillars: i) a network model layer that allows a target network model; ii) an adaptability layer, which makes necessary changes in the network to obtain logic and behavior defined at the model; and iii) a network layer reflecting the SDN architecture. In addition to this layering, MOSES provides the integration of intermediary functional blocks to perform tasks, such as code generation, reflection of the network state on the network model layer, and a formal analysis of modeled logic. To validate this architecture, we started building a prototype of the MOSES framework and applied it to modeling SDN applications and scenarios to identify its benefits. We used the Mininet tool to perform simulations. In the experiments carried out so far, we modeled real topologies interacting with application models of load balancing, monitoring and QoS. The preliminary results show that the functional blocks of MOSES enable: i) a semantic validation of SDN applications, ii) the increase of abstraction level and compatibility (the latter at the cost of 15% of overhead); and iii) a guarantee of QoS levels using only algorithms rather than traditional queuing techniques, which improved delay rates by up to 22% when compared to standard implementations.

Keywords: model-driven engineering; autonomic networking management; software-defined networking.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AN	*Autonomic Networking*
ANet	*Active Networking*
ANM	*Autonomic Network Management*
API	*Application Programming Interface*
CASE	*Computer-Aided Software Engineering*
CIM	Common Information Model
DF	Delegação de Fluxo
DMTF	*Distributed Management Task Force*
DRL	*Deep Reinforcement Learning*
DSL	*Domain-Specific Language*
DSML	*Doman-Specific Modeling Language*
E2E	*End-to-End*
EGL	*Eclipse Generation Language*
ETSI	*European Telecommunications Standard Institute*
EVL	*Eclipse Validation Language*
FRP	*Functional Reactive Programming*
GANA	*Generic Autonomic Network Architecture*
GMF	*Graphical Modeling Framework*
GPL	*General-Purpose Language*
IETF	*Internet Engineering Task Force*
IRTF	*Internet Research Task Force*
MAPE-K	*Monitor, Analyze, Plan, Execute, and Knowledge*
MART	*Models at Run-Time*
MDC	*Multiple Description Coding*
MDD	*Model-Driven Development*
MDE	*Model-Driven Engineering*
MDN	*Model-Driven Networking*
MOSES	*Models at Runtime for Self-Adaptive SDN*
NBI	*Northbound Interface*
NF	*Network Function*

NFV	*Network Functions Virtualization*
NML	*Network Modeling Language*
ONF	*Open Networking Foundation*
Opensig	*Open Signaling*
OVS	Open vSwitch
POF	*Protocol Oblivious Forwarding*
QoE	Qualidade de Experiência
QoS	Qualidade de Serviço
QPN	*Queueing Petri Nets*
RdP	Redes de Petri
RPC	*Remote Procedure Call*
RSVP	*Resource Reservation Protocol*
SBI	*Southbound Interface*
SDN	*Software-Defined Networking*
SPN	*Stochastic Petri Nets*
SRA	*Source Routing Algorithm*
UMF	*Unified Management Framework*
UML	*Unified Modeling Language*
VPN	*Virtual Private Network*
WAN	*Wide Area Networks*
XML	*eXtensible Markup Language*

LISTA DE SÍMBOLOS

γ	Letra grega Gama
$\in$	Pertence
δ	Delta
θ	Teta
σ	Sigma
μ	Mi

SUMÁRIO

1 INTRODUÇÃO

Desenvolver aplicações que interajam com a rede e, possivelmente, modificam políticas de rede, sempre foi um objetivo de pesquisadores e projetistas. Essa possibilidade é vista como uma das principais peças que faltavam para permitir inovações na rede e alcançar outros benefícios, como a redução de custos operacionais e financeiros. Com o surgimento do paradigma de Redes Definidas por Software, ou SDN (MCKEOWN et al., 2008), alcançar essa interação entre rede e aplicações tornou-se possível graças à programabilidade da rede promovida pela arquitetura SDN. Entretanto, dificuldades na criação de aplicações e no gerenciamento de SDN prejudicam a obtenção dos benefícios (e.g., inovação, redução de custos) possibilitados por esse paradigma de redes.

Historicamente, o ato de desenvolver software já é considerado complexo, e torna-se ainda mais difícil quando inserido no contexto de SDN. Lidar com diferentes protocolos, eventos e, ainda assim, garantir a consistência do funcionamento da rede ilustram essa dificuldade. Com a crescente criação de aplicações SDN, diversos autores (REICH et al., 2013a; LOPES et al., 2016) perceberam que, para desenvolver estas aplicações, a utilização de linguagens de programação de propósito geral, i.e., *General-Purpose Language* (GPL), não é recomendada dado o alto grau de complexidade e baixo nível de abstração envolvidos nestas linguagens em um cenário de SDN. Isso resultou em diversas propostas de linguagens de programação de domínio específico, i.e., *Domain-Specific Language* (DSL)s, para facilitar o desenvolvimento de aplicações e definição de políticas em SDN (LOPES et al., 2016). Por exemplo, a linguagem Pyretic (REICH et al., 2013a) permite um alto nível de abstração na implementação de políticas de rede, sem a necessidade de lidar com mecanismos de baixo nível de protocolos SDN (e.g., OpenFlow).

Vale ressaltar que, ao discutir o desenvolvimento de aplicações SDN, este trabalho refere-se à implementação de fato destas aplicações. Balanceamento de carga, monitoramento, roteamento e diversas outras ações são exemplos possíveis de aplicações que podem ser implementadas numa rede SDN, a partir de algoritmos executados no plano de controle. Em comparação, o *European Telecommunications Standard Institute* (ETSI), dentro do conceito de *Network Functions Virtualization* (NFV) – uma arquitetura que propõe a virtualização da rede e seus serviços (MATIAS et al., 2015) – define estas aplicações como funções de rede, ou *Network Function* (NF). No geral, são aplicações que interagem com a rede (e.g., especificação de rotas, alteração de cabeçalhos) definindo o seu comportamento.

A mudança na forma de interação com a rede, através de aplicações e políticas programáveis, e as contínuas inovações promovidas em SDN têm também um impacto direto no seu gerenciamento (YEGANEH; TOOTOONCHIAN; GANJALI, 2013a). O mesmo fator que permite a programabilidade da rede em SDN (i.e., separação entre plano de controle e planos de dados) torna o seu gerenciamento um desafio, já que o controle centralizado

de SDN deve lidar com diversas aplicações, tipos de dispositivos e protocolos, além de eventos inesperados como falhas ou sobrecargas. Essa heterogeneidade contribui para que o gerenciamento destas redes possa ser tão complexo quanto o seu desenvolvimento.

Neste cenário, requisitos do gerenciamento como desempenho, flexibilidade e visibilidade ainda são considerados desafios em aberto pela literatura (WICKBOLDT et al., 2015), ainda que necessários para alcançar o pleno gerenciamento de redes SDN. Diante disso, alguns autores defendem que as características da arquitetura SDN favorecem que o gerenciamento de rede seja feito de forma dinâmica e adaptativa (TUNCER et al., 2015; POULIOS et al., 2014; KIM; FEAMSTER, 2013). No paradigma NFV, por exemplo, juntamente com o conceito de arquitetura de rede virtualizada (e.g., virtualização do plano de controle de SDN), o próprio ETSI prevê em sua especificação a autonomia no gerenciamento e orquestração da rede virtual.

Felizmente, apesar de SDN ser um paradigma relativamente novo, muitas classes de problemas relacionados ao seu desenvolvimento e gerenciamento já foram solucionadas em outras áreas ou possuem arquiteturas de referência fornecem uma base para compor soluções robustas. Um exemplo disso é a área da Engenharia de Software Baseada em Modelos, do inglês *Model-Driven Engineering* (MDE) (SCHMIDT, 2006b), a qual fornece um alto nível de abstração no desenvolvimento de aplicações a partir de modelos. A área de MDE, e os modelos resultantes de soluções baseadas nela, podem ser aplicados tanto ao desenvolvimento quanto ao gerenciamento de SDN (LOPES et al., 2015).

Da perspectiva específica do gerenciamento da rede, a tendência em realizar tal atividade de forma autonômica fez com que arquiteturas e modelos de referência como a GANA (CHAPARADZA et al., 2013), descrita pelo ETSI, e o modelo de AN (BEHRINGER et al., 2017), estabelecido pela *Internet Research Task Force* (IRTF), surgissem para guiar a implementação de soluções para realizar este tipo de gerenciamento, ainda que questões sobre essa implementação em SDN não tenham sido respondidas.

Neste cenário, este trabalho apresenta o framework MOSES para resolver classes de problemas envolvidos nas áreas de desenvolvimento e gerenciamento de aplicações de redes SDN autonômicas. Para realizar realizar este framework, uma abordagem evolutiva baseada em modelos em tempo de execução foi utilizada, visando alcançar a implementação de redes SDN auto-adaptáveis. Os resultados alcançados neste trabalho mostram que o MOSES e seus componentes podem trazer benefícios significantes para o cenário de SDN, tais como:

i) aumento no nível de abstração para gerenciar a rede: a adição de uma camada de modelagem na arquitetura SDN permite que detalhes de baixo nível sejam abstraídos;

ii) validação semântica de aplicações: ao permitir a modelagem de aplicações, a semântica dos modelos possibilita que projetistas de rede validem a lógica e a compo-

sição as aplicações modeladas;

iii) aumento na compatibilidade entre aplicações e infraestrutura: uma técnica de delegação de fluxo, associada ao desenvolvimento baseado em modelos e a visão global da rede por controladores SDN, permite que os requisitos das aplicações sejam atendidos, ainda que parcialmente, por elementos de rede;

iv) predições de desempenho: os modelos criados fornecem a semântica, os dados e as relações que compõem a rede, podendo ser utilizados para realizar predições de desempenho; e

v) auto-adaptação para atingir objetivos definidos em modelos de alto nível: a partir dos modelos e das predições realizadas, é possível realizar ações auto-adaptativas para alcançar objetivos modelados em alto nível.

Nos experimentos realizados, os ganhos de desempenho de rede ao utilizar, por exemplo, o componente de delegação de fluxo (cf. Capítulo 4) do MOSES resultou numa melhoria de até 22% no atraso para o cenário avaliado.

1.1 MOTIVAÇÃO

O nível de complexidade a ser superado nas atividades de desenvolvimento e gerenciamento em SDN tem motivado a realização de diversas pesquisas para encontrar soluções que permitam criar e gerenciar SDNs em alto nível de abstração, correção e consistência. Por exemplo, considerando requisitos fundamentais como disponibilidade e desempenho, é extremamente complexo e oneroso para um operador continuamente verificar e modificar a rede para reparar possíveis erros ou melhorar métricas. Isso fica ainda mais evidente quando esta rede possui um número considerável de nós a serem gerenciados.

Além disso, desenvolver aplicações de software complexas não é uma tarefa fácil, ainda mais quando o alvo destas aplicações é um ambiente SDN. De acordo com Avaya (2015), 89% de 1.421 profissionais em 15 países dizem que a programação em SDN precisa ser simples para motivar a sua implantação. Além disso, 25% deles afirmam que a adoção de SDN fica estagnada devido a sua complexidade inerente. Por outro lado, na pesquisa realizada por Kim et al. (2015a), os autores apresentaram resultados de uma análise onde 89% dos participantes não tinha certeza sobre a existência de bugs na configuração ou mal comportamento de algum elemento de rede.

O mesmo caso aplica-se à realização de mudanças na rede de acordo com seu estado, e.g., definição de novas rotas para balanceamento de carga, QoS, segurança. Neste cenário, Barron et al. (2016), Wickboldt et al. (2015) e Tuncer et al. (2015) são exemplos de autores que apontam para a realização de um gerenciamento autonômico para superar estes desafios, diminuindo custos e a margem de erro humano. Ressaltando que desenvolver

NFs que possibilitem o gerenciamento autonômico também não é uma tarefa trivial. Este cenário apresenta indícios da necessidade de propostas que auxiliem operadores de rede em tarefas de implementação. É neste contexto que este trabalho se insere, ao oferecer uma solução para prover o desenvolvimento e o gerenciamento autonômico de redes SDN.

Apesar de muitas abordagens terem sido propostas na literatura, ainda não há um consenso sobre como desenvolver NFs que possam compor a lógica e o gerenciamento autonômico da rede, de forma que características de flexibilidade e desempenho sejam alcançadas. Além disso, mesmo com o surgimento de uma abundância de controladores SDN como NOX (GUDE et al., 2008a), Floodlight (HARKAL; DESHMUKH, 2016), OpenDaylight (MEDVED et al., 2014) e Ryu, um padrão de desenvolvimento e gerenciamento para estas redes ainda continua em aberto. Isso dificulta prevenir que aplicações e ações de gerenciamento ineficientes, ou mesmo incorretas, apareçam. Ressaltando que este cenário se traduz na introdução de comportamentos inesperados e indesejados na rede.

Geralmente, as tarefas de **verificar**, **corrigir** e **garantir o correto funcionamento da rede** têm um custo alto (e.g., tempo, recursos financeiros). Atualmente, a verificação manual de aplicações implementadas com GPL ou mesmo com algumas DSLs (LOPES et al., 2016) é ineficiente e propensa a erros humanos. Alguns trabalhos realizam estas tarefas a partir do seu plano de dados (KAZEMIAN et al., 2013; KHURSHID et al., 2012), mas iniciativas que considerem o plano de controle na verificação e realizem isso de forma autonômica ainda são escassas (KIM et al., 2015b; WICKBOLDT et al., 2015; TUNCER et al., 2015). Este cenário também é motivo para buscar meios de garantir que o controle da rede está sendo feito corretamente. Ou seja, é preciso prover mecanismos para permitir que a rede possa corrigir seu comportamento de forma autonômica e de acordo com requisitos e objetivos do operador de rede.

Além disso, embora existam iniciativas para permitir a realização de um gerenciamento autonômico da rede (LI et al., 2013; AHMAD et al., 2015; YAHIA et al., 2017) e outras que permitem descrever aplicações SDN que reagem a determinados eventos na rede, continuam em aberto soluções que realizem a integração entre estas ações, realizando um melhor **aproveitamento dos recursos**. Ou seja, se faz necessária uma solução que promova flexibilidade, compatibilidade e adaptabilidade, onde suas **aplicações não resultem em um mau comportamento da rede** e que **ações de gerenciamento não prejudiquem a lógica esperada destas aplicações**.

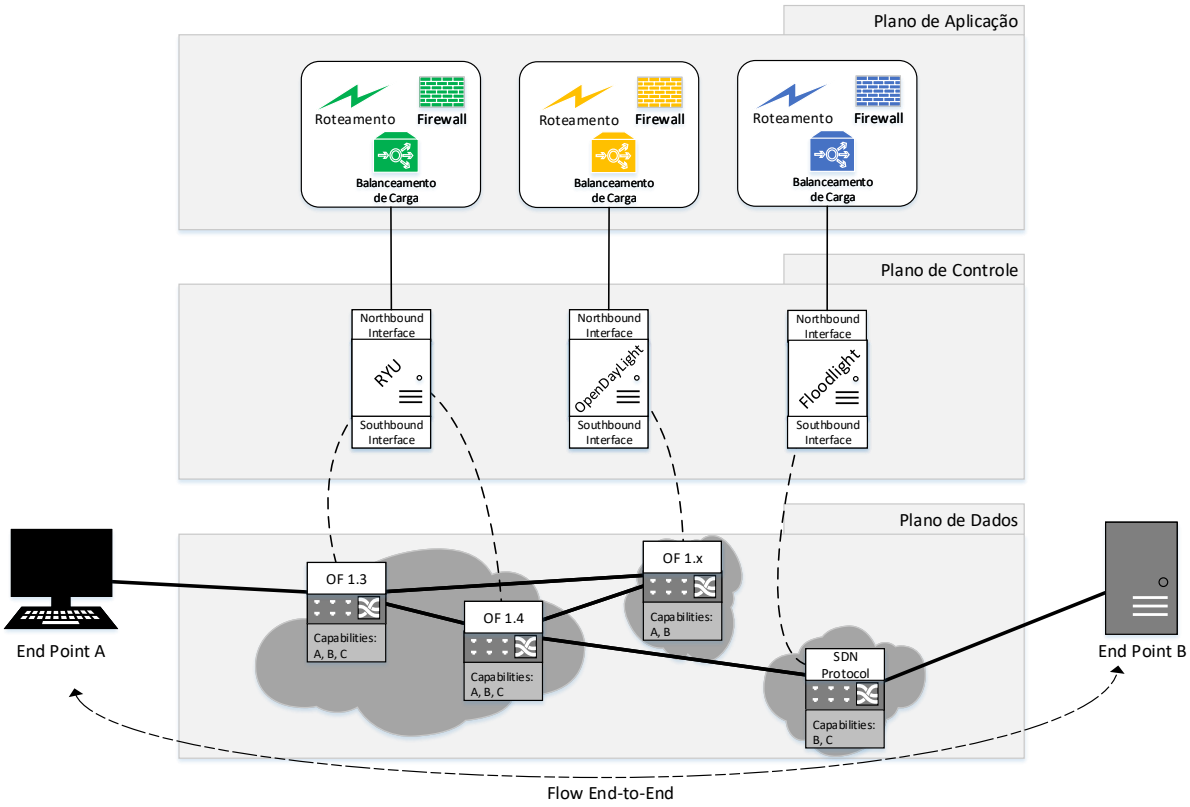
Por fim, vale ressaltar também a crescente necessidade em utilizar SDN para atender a demanda por flexibilidade de novas tecnologias de rede (e.g., 5G) (SUN et al., 2015) e para remover a ossificação dos *backbones* na Internet (TURNER; TAYLOR, 2005). Aplicar autonomicidade nessa utilização também demonstra um potencial interessante ao permitir a redução de custos e a alocação flexível de recursos (DROSTE et al., 2016). É neste cenário que a adequação de **modelos em tempo de execução** é defendida nesta tese como um fator chave na implementação e gerenciamento de redes SDN autonômicas, devido ao alto

grau de abstração, consistência e funcionalidades *self-** permitidas em instâncias destes modelos.

1.2 DESCRIÇÃO DO PROBLEMA

Conforme já discutido, é fato que o sucesso de SDN deve-se à sua arquitetura, ilustrada na Figura 1, que permite definir todo o comportamento da rede através de software (i.e., aplicações e seu plano de controle). Nesta arquitetura, sabe-se também que as saídas de software são regras de fluxo ou requisições que realizam interações com o plano de dados da rede, definindo destino de fluxos (e.g., porta de saída nos switches) ou obtendo informações (e.g., topologia, estatísticas).

Figura 1 – Camadas da arquitetura SDN e cenário de heterogeneidade.



Fonte: O autor (2020).

Um dos problemas na interação entre as camadas da arquitetura SDN é **garantir que aplicações**, ao serem executadas pelo plano de controle, **estejam consistentes** entre si e com o plano de dados. Por exemplo, é preciso evitar que uma ou mais aplicações insiram regras de fluxo contraditórias no plano de dados (cf. Tabela 1, primeira e segunda linhas com ações opostas).

Ainda com relação à implementação e execução das aplicações, com o aumento da heterogeneidade de soluções (e.g., diferentes tipos e versões de protocolos, dispositivos e

Tabela 1 – Exemplo de tabela OpenFlow resumida com duas regras inconsistentes.

Prioridade	Porta de Entrada	MAC Endereço Origem	MAC Endereço Destino	Protocolo	IP Origem	IP Destino	Porta Destino	Ações
100	*	*	*	*	10.0.0.1	*	*	Encaminhar para Porta 1
100	*	*	*	*	10.0.0.1	*	*	*Drop*
30000	4	*	*	*	10.0.0.5	*	*	Enviar para Controlador

Fonte: O autor (2020).

controladores), cresce também a necessidade por implementar aplicações que **melhorem a utilização de recursos e sejam compatíveis com diferentes infraestruturas** subjacentes (SUN et al., 2015; SONG, 2013a). Por exemplo, uma aplicação de monitoramento que utiliza o protocolo IPFIX em sua implementação pode tornar-se inutilizável, caso o monitoramento seja realizado em dispositivos que não suportem esse protocolo, ainda que, por outro lado, elementos vizinhos na mesma rede sejam compatíveis.

Por fim, Tuncer et al. (2015), Sun et al. (2015) e Kukliński et al. (2014) avaliam que gerenciar a lógica e o comportamento da rede manualmente é uma tarefa lenta, extremamente difícil e propensa a erros. Este nível de dificuldade aumenta consideravelmente em um plano de controle composto por múltiplos controladores (cf. Figura 1). Já que, apesar dos benefícios de um plano de controle (lógico) centralizado, é improvável que apenas um controlador (físico) consiga atender a demanda de redes grandes e complexas. Assim, diversas questões sobre a melhor forma de gerenciar múltiplos controladores, juntamente com a gerência de aplicações e da infraestrutura subjacente, estão em aberto. Uma destas questões refere-se, por exemplo, à quantidade mínima de controladores que devem ser instanciados numa rede, bem como onde posicioná-los e como manter a consistência entre as suas aplicações.

Neste cenário, considerando o problema abordado por esta tese, sobre o alto nível de complexidade no gerenciamento e baixo nível de abstração para desenvolver soluções SDN, evidencia-se que uma abordagem autonômica pode ter um impacto significativo no melhoramento da gerência de redes SDN. Porém, a realização de tal abordagem precisa superar desafios que a tornem possível e benéfica de fato. Para alcançar essa realização, seguindo observações que foram feitas ao analisar a literatura sobre gerenciamento autonômico em SDN, implementação de NFs e possibilidades de utilização de modelos em tempo de execução, este trabalho baseia-se nas seguintes premissas:

1. A criação de NFs deve seguir um processo de desenvolvimento que permita a definição de requisitos, a implementação e a validação destas NFs.

2. Cenários virtualizados ou mudanças no ambiente de rede não devem resultar em comportamentos de rede falhos.

3. O gerenciamento autonômico de redes SDN é possível e pode ser realizado.

Desta forma, com base nas premissas anteriores, as seguintes hipóteses foram investigadas:

Hipótese 1 *A utilização de uma abordagem baseada em modelos permite o desenvolvimento e o gerenciamento de redes SDN, obtendo ganhos funcionais, independentemente do fabricante dos elementos que compõem a rede e tipo de instanciação (i.e., física ou virtual).*

Hipótese 2 *Modelos em tempo de execução favorecem a realização do gerenciamento autonômico de SDN, permitindo o alcance dos requisitos que definem este tipo de gerenciamento*;

Hipótese 3 *Blocos funcionais adaptáveis, construídos a partir de modelos, podem abrigar algoritmos e técnicas heurísticas que potencializem o aproveitamento de recursos de rede.*

Após o estabelecimento destas hipóteses, quatro principais questões de pesquisa (QP) surgem para serem respondidas no decorrer do trabalho:

Questão de Pesquisa 1 *Quais estratégias são mais indicadas para desenvolver aplicações SDN, de forma que requisitos como um alto nível de abstração, a validação e a independência de controladores sejam atendidos?*

Questão de Pesquisa 2 *Quais as formas de realizar o gerenciamento autonômico de redes SDN e quais os pontos positivos e negativos em utilizar esse tipo de gerenciamento?*

Questão de Pesquisa 3 *Qual o impacto de utilizar uma abordagem baseada em modelos no desenvolvimento e gerenciamento de SDN?*

Questão de Pesquisa 4 *Quais características devem estar presentes em uma arquitetura para permitir a implementação de aplicações e o gerenciamento autonômico de redes SDN?*

1.3 OBJETIVOS E METAS

Na direção de comprovar ou refutar as hipóteses e responder as questões de pesquisa definidas na subseção anterior, o objetivo geral desta tese é: *oferecer uma estrutura baseada em modelos em tempo de execução que permita o desenvolvimento e gerenciamento de SDNs auto-adaptáveis*. Porém, para alcançar tal composição, é preciso atingir os seguintes objetivos específicos (OE):

- OE1: Realizar uma revisão bibliográfica sobre ferramentas, linguagens e estruturas focadas no desenvolvimento e gerenciamento de SDN, considerando abordagens que visem auto-adaptabilidade; (QP1, QP2 e QP4)
- OE2: Realizar a implementação e análise de algoritmos e metamodelos que favoreçam o aumento da compatibilidade na rede e melhorem métricas de utilização de recursos e desempenho de forma autonômica; (QP2, QP3)
- OE3: Implementar uma ferramenta baseada nos metamodelos do objetivo OE2 para permitir a elaboração de modelos de alto nível que reflitam e modifiquem o comportamento da rede de forma autonômica, avaliando os pontos fortes e fracos desta estratégia; (QP3)
- OE4: Elaborar e executar experimentos que avaliem a solução proposta, obtendo métricas do seu desempenho e caracterizando de que forma essa proposta atende aos requisitos de gerenciamento autonômico; (QP2, QP3)
- OE5: Comparar quanti-qualitativamente a solução proposta e seus componentes com outras abordagens da literatura. (QP4)

1.4 CONTRIBUIÇÕES ALCANÇADAS

Uma das primeiras contribuições deste trabalho foi demonstrar as lacunas existentes no desenvolvimento de aplicações SDN (LOPES et al., 2016). Através de uma análise do estado da arte, diversas linguagens de programação, paradigmas e desafios foram identificados, norteando parte do desenvolvimento dessa tese.

Em seguida, ao estender uma abordagem anterior, que ficou conhecida como *Model-Driven Networking* (MDN) (LOPES et al., 2015), foi possível observar o benefício da utilização de uma abordagem baseada em modelos para realizar a validação de aplicações SDN, permitindo a verificação de inconsistências ou erros antes da implantação destas aplicações na rede. Ressaltando que esta verificação pôde ser realizada com tempo de execução satisfatório para topologias distintas (LOPES et al., 2016).

Outro passo dado foi em direção à compatibilidade de aplicações e ao aproveitamento de recursos da rede (LOPES et al., 2018). Foi utilizado o potencial de uma abordagem baseada em modelos, através da geração de código das aplicações, para aumentar o nível de compatibilidade entre estas aplicações e a infraestrutura subjacente. Isso permitiu ainda que a mesma geração de código pudesse realizar o favorecimento ou balanceamento de recursos visando atingir requisitos de QoS. Além disso, demonstrou-se como a transformação de modelos pode contribuir para a avaliação de desempenho de cenários SDN (LOPES; SOUZA; FERNANDES, 2018).

Por fim, a arquitetura que baseia o framework MOSES proposto contribuiu para uma expansão do conceito de MDN, demonstrando que a utilização de modelos em tempo de execução favorecem não só a implementação de aplicações SDN através de modelagem, mas também o gerenciamento autonômico em SDN.

1.5 ESTRUTURA DA TESE

Considerando esta linha de desenvolvimento evolutiva desta tese, o presente trabalho está organizado da seguinte forma:

- **Capítulo 2** – Apresenta e define o referencial teórico que baseia os conceitos utilizados na tese, ressaltando as áreas de Redes Definidas por Software e de Engenharia de Software Baseada em Modelos.
- **Capítulo 3** – Apresenta a avaliação de modelos de alto nível para a criação e validação de aplicações SDN.
- **Capítulo 4** – Apresenta a utilização de modelos para melhorar infraestruturas SDN.
- **Capítulo 5** – Apresenta a predição de métricas de desempenho em redes SDN por meio de transformações de modelos.
- **Capítulo 6** – Apresenta modelos em tempo de execução integrados com algoritmos de predição para SDN auto-adaptáveis.
- **Capítulo 7** – Apresenta as considerações finais da tese.

2 REFERENCIAL TEÓRICO

Neste capítulo, os diversos conceitos envolvidos na realização da tese serão definidos e apresentados. Começaremos abordando a definição de SDN que norteia o desenvolvimento deste trabalho, passando pela revisão sobre a área de Engenharia de Software Baseada em Modelos, incluindo uma visão geral sobre o conceito de Modelos em Tempo de Execução, ou *Models at Runtime* (MART), e a sua integração com SDN. Além disso, ao tratar do Gerenciamento Autonômico de Redes, apresentamos os modelos de referência existentes.

2.1 REDES DEFINIDAS POR SOFTWARE

A separação do plano de controle do plano de dados é um dos pilares do paradigma SDN. É esta arquitetura desacoplada que permite a programabilidade da rede. Historicamente, a comunidade de pesquisa realiza diversas tentativas para prover programabilidade na rede. Por exemplo, *Active Networking* (ANet) (TENNENHOUSE et al., 1997) e *Open Signaling* (Opensig) (CAMPBELL et al., 1999) são consideradas abordagens anteriores sobre programabilidade da rede (FEAMSTER; REXFORD; ZEGURA, 2013). Neste contexto, duas questões surgem: i) por que as abordagens anteriores não obtiveram sucesso? e ii) quais são as principais diferenças entre SDN e tais abordagens?

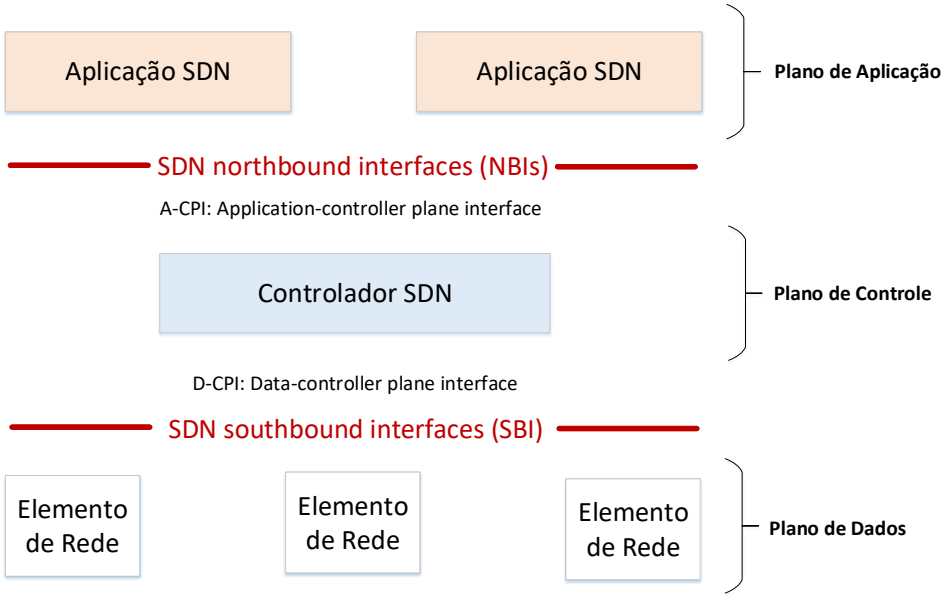
A partir dessas questões, observa-se que o surgimento de SDN não foi sem motivo. ANet focou em possibilitar que usuários configurassem roteadores na rede, assim, a programabilidade foi desenhada para ocorrer diretamente no plano de dados. Este princípio resultou em preocupações para operadores de rede (e.g., aplicações inconsistentes, segurança). De maneira similar à SDN, Opensig também focou em fornecer programabilidade da rede separando o controle da rede da informação de transporte, mas baseada em alta dependência em interfaces de programação estáticas.

SDN avança no estado da arte em programabilidade da rede e foca em superar a dependência de interfaces específicas de fornecedores. Sua arquitetura permite que redes evoluam rapidamente, reduzindo a complexidade para implantar novos protocolos, incluindo os considerados experimentais. Além da separação entre planos de dados e controle, a operação de redes SDN difere da operação de redes tradicionais. Em redes tradicionais, cada switch possui seu próprio software local para realizar a lógica de encaminhamento de pacotes. Em SDN, esta lógica é centralizada e externamente definida no plano de controle através de um controlador em forma de software (podendo ser executado em hardware básico), o qual possui uma visão global da rede e pode ser programado para atender aos requisitos estabelecidos por desenvolvedores, operadores, ou provedores de rede.

2.1.1 Arquitetura

Quando o controle lógico é separado dos dispositivos de encaminhamento, toda a inteligência da rede (e.g., decisões sobre roteamento, permissões) é movida para o controlador. Assim, o controlador SDN se torna o componente de rede responsável por gerenciar a rede. O gerenciamento então ocorre através de uma tabela de fluxos presente nos switches da rede, os quais recebem e registram regras de rede definidas pelo controlador. Em outras palavras, o controlador SDN adiciona entradas nas tabelas de fluxo dos switches para lidar apropriadamente com pacotes ou fluxos. Neste cenário, o controlador possui toda informação de rede necessária (e.g., onde pontos da rede estão conectados, topologia) que pode ser usada para lidar com possíveis conflitos envolvendo políticas ou evitar um mal comportamento de elementos de rede.

Figura 2 – Arquitetura SDN promovida pela ONF.



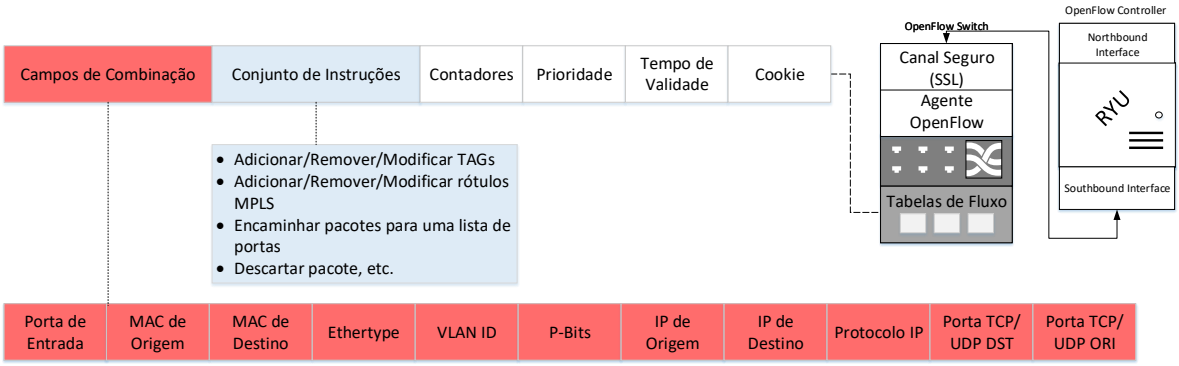
Fonte: Adaptado de ONF (2012).

A Figura 2 faz remete à uma das principais de arquiteturas de referência em SDN, que foi definida pela *Open Networking Foundation* (ONF)[1]. Esta arquitetura mostra uma visão de SDN composta pelos planos de Aplicação, Controle e Dados. O Plano de Dados compreende elementos de rede (e.g., switches) que expõem suas capacidades para o Plano de Controle através da *Southbound Interface* (SBI) do controlador. As aplicações existem no Plano de Aplicação, e comunicam seus requisitos de rede ao Plano de Controle através de *Northbound Interface* (NBI). No meio desta arquitetura, o controlador SDN traduz os requisitos de aplicação e exerce um controle de baixo nível dentre os elementos de rede, ao mesmo tempo em que fornece informações relevantes para as aplicações SDN.

[1] Open Networking Foudation - <http://www.opennetworking.org>

Há um consenso de que a SBI tem no protocolo OpenFlow (MCKEOWN et al., 2008) a sua principal instância (FEAMSTER; REXFORD; ZEGURA, 2013; KREUTZ et al., 2015), apesar de existir espaço para discussão e melhoramentos (DORIA et al., 2010; SMITH et al., 2014), como a eficiência do plano de dados e sua integração com o plano de controle. Resumidamente, o protocolo OpenFlow define a utilização de tabelas de fluxo nos switches e a forma com que regras de fluxo são instaladas nestas tabelas. A Figura 3 ilustra a composição destas tabelas.

Figura 3 – Exemplo de composição das tabelas de fluxos do protocolo OpenFlow.



Fonte: O autor (2020).

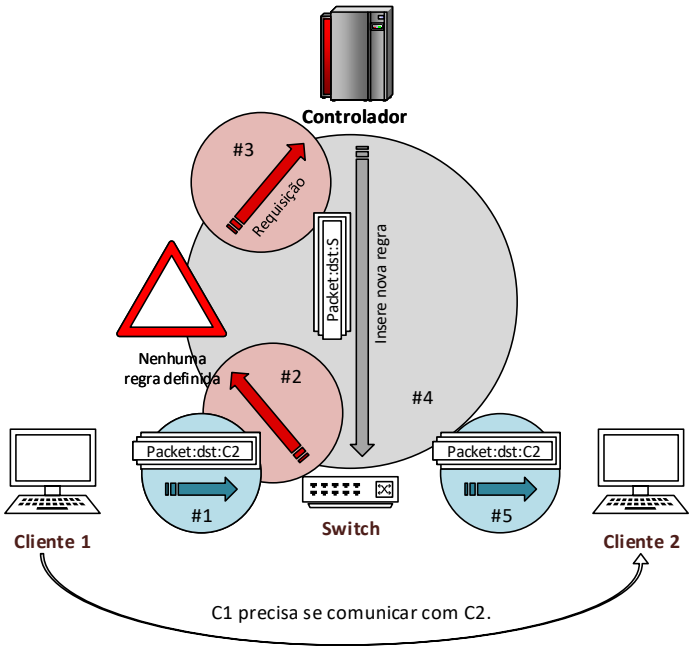
Na NBI, por outro lado, ainda não há um padrão amplamente aceito para especificar de que forma a comunicação entre o plano de aplicação e o plano de controle deve ocorrer. Felizmente, algumas iniciativas já surgiram visando desenhar protótipos, codificar padrões e produzir artefatos que talvez possam validar a criação de um padrão para esta interface (ZHOU et al., 2014; FOSTER et al., 2011a).

2.1.1.1 Tipos de Implantação

Os controladores SDN são componentes estratégicos na implantação de SDN. São eles que se comunicam com a infraestrutura subjacente (via SBI) e com as aplicações SDN (via NBI). Como já discutido, para realizar esta interação, um controlador SDN envia mensagens para os switches disseminando regras específicas ou gerais para manipular pacotes e fluxos, as quais são definidas por desenvolvedores ou administradores de rede através da NBI.

O cenário hipotético ilustrado na Figura 4 nos ajuda a entender a responsabilidade do controlador e como ele pode realizar a inserção de novas regras em um switch. Nesta figura, o Cliente 1 (C1) deseja estabelecer uma conexão com o Cliente 2 (C2). Assim, C1 envia pacote com destino à C2 (passo #1). O elemento Switch (S) no caminho entre C1 e C2 recebe os pacotes. Neste momento, S não possui regras de encaminhamento para os pacotes vindos de C1 com destino à C2 (passo #2). Após essa chegada e a verificação

Figura 4 – Sequência ilustrando a instalação de regras de fluxo passo-a-passo.



Fonte: O autor (2020).

de que não há regras definidas, S gera uma requisição para o Controlador (C) (passo #3) para que ele decida o que fazer com os pacotes. Enquanto S espera a decisão de C, os pacotes são bufferizados e recebem uma identificação. Quando a resposta de C chega, através de um novo pacote que instala regras em S (passo #4), tal switch S se torna capaz de encaminhar todos os pacotes de C1 para C2 (passo #5) sem nenhuma necessidade adicional de comunicação com o controlador. Vale destacar que, atualmente, a maioria das redes SDN utiliza o protocolo OpenFlow (MCKEOWN et al., 2008) para realizar toda a comunicação switch-controlador-switch.

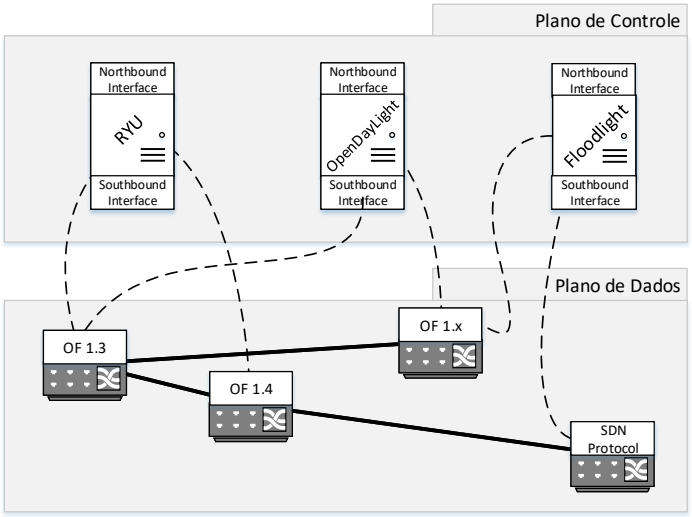
Desde o surgimento do paradigma SDN, diversos controladores têm sido propostos: NOX (GUDE et al., 2008b), Beacon (ERICKSON, 2013a), OpenDaylight[2] e Ryu[3]. Paralelamente, diversas formas de implantá-los na rede têm sido propostas visando alcançar alguns objetivos (e.g., disponibilidade, desempenho). Por exemplo, podemos perceber que uma mudança possível em relação ao comportamento descrito na Figura 4 (i.e., **reativo**) é instalar regras de fluxo sem a necessidade de o controlador receber requisições dos switches (i.e., **pró-ativo**). Geralmente, a implantação dos controladores é feita de forma que eles se comportem de maneira **híbrida**, ou seja, combinando a instalação de regras de forma proativa e reativa.

Outro aspecto sobre a implantação de controladores SDN refere-se à quantidade e posição destes controladores. É possível considerar que uma rede SDN seja composta por

[2] OpenDaylight - <https://www.opendaylight.org>
[3] Ryu - <https://osrg.github.io/ryu/>

múltiplos controladores, principalmente nos casos em que deseja-se obter características de dependabilidade na rede (YEGANEH; TOOTOONCHIAN; GANJALI, 2013a). Neste caso, além de um controlador poder gerenciar múltiplos switches, um switch pode ser gerenciado por múltiplos controladores (cf. Figura 5). Por outro lado, o posicionamento do controlador na rede pode favorecer a obtenção de outros ganhos referentes a desempenho. Tal posicionamento deve variar conforme a topologia (HELLER; SHERWOOD; MCKEOWN, 2012).

Figura 5 – Composição do Plano de controle por múltiplos controladores.



Fonte: O autor (2020).

2.1.2 Casos de Uso

Os benefícios de SDN não seriam claros caso este paradigma não pudesse ser aplicado em casos de uso reais. Baseando-se em Lara, Kolasani e Ramamurthy (2014) e Jarschel et al. (2014), é possível categorizar os casos de uso de SDN nas seguintes classes: roteamento, orquestração de nuvens, balanceamento de carga, monitoramento de rede e medição, gerenciamento de rede e segurança. A seguir, descreveremos como SDN se encaixa em cada uma destas classes.

2.1.2.1 Roteamento

Os esquemas de roteamento descentralizado atual, implantados em dispositivos de rede de fornecedores específicos, tornam difícil a adaptação de rotas a diferentes cenários (e.g., máquinas virtuais em ambientes de nuvem). A interface SDN entre o plano de controle e o plano de dados oferece diversas possibilidades na adaptação de protocolos de roteamento.

Ou seja, é desta forma que SDN permite a implementação de serviços de roteamento para diferentes contextos (e.g., seleção de rotas, otimização de tráfego, roteamento seguro) (AGARWAL; KODIALAM; LAKSHMAN, 2013). Esta capacidade pode ainda ser estendida para fornecer garantias de QoS na rede (BARI et al., 2013).

2.1.2.2 Orquestração de Nuvens Computacionais

Uma orquestração unificada da TI e recursos de rede (e.g., dispositivos de encaminhamento) é considerada uma das mais importantes realizações de uma rede SDN (AUTENRIETH et al., 2013). A orquestração de nuvens envolve o gerenciamento de interconexões e interações entre datacenters e redes de transporte. Com SDN, este caso de uso pode resultar em uma oferta de ambiente virtualizado, por exemplo, larguras de banda customizadas para nós da rede e políticas específicas para tranferir grandes quantidades de dados. Neste cenário, o controlador SDN pode se comunicar com orquestradores de nuvem, e.g., Open-Nebula (MORENO-VOZMEDIANO; MONTERO; LLORENTE, 2012), e, por exemplo, quando uma nova máquina virtual iniciar, o orquestrador de nuvem pode requisitar ao controlador da rede configurações para esta máquina. Além disso, o controlador pode notificar o orquestrador da nuvem sobre uma rota sobrecarregada.

Neste cenário, o controlador pode definir uma rota alternativa, se disponível, ou o orquestrador da nuvem pode mover recursos da nuvem para outras localizações baseado no estado da rede (MORENO-VOZMEDIANO; MONTERO; LLORENTE, 2012; KREUTZ et al., 2015).

2.1.2.3 Balanceamento de Carga

Balanceadores de carga distribuem o volume de trabalho que chega entre um conjunto de múltiplos servidores replicados, definindo novas rotas de tráfego para estes servidores com o objetivo de descongestionar o uso de recursos computacionais. Atualmente, balanceadores de carga são, em sua maioria, implantados como componentes específicos de hardware. Neste cenário, SDN permite que balanceadores de carga sejam parte do controle lógico da rede. Um exemplo desta possibilidade é o seguinte: se um nó da rede estiver sobrecarregado, o balanceador de carga pode notificar ao controlador, o qual encaminha a carga de trabalho para um nó diferente e replicado. Outro caminho é a verificação ativa de enlaces congestionados por parte do controlador, oferecendo rotas alternativas. De acordo com Jarschel et al. (2014), dispositivos que servem apenas como balanceadores de carga podem ser substituídos por controladores SDN e balanceadores de carga em forma de software.

2.1.2.4 Monitoramento da Rede e Medição

SDN oferece para operadores de rede as capacidades de monitorar e medir fluxos de tráfego, sem a necessidade de um dispositivo de rede adicional (JARSCHEL et al., 2014; YU et al., 2013). A visão global da rede pelo controlador SDN favorece a implementação de mecanismos de monitoramento.

2.1.2.5 Gerenciamento de Rede

Atualmente, o gerenciamento de rede se torna cada vez mais complexo conforme o número de dispositivos que precisam ser configurados e monitorados aumenta. O controle lógico centralizado de SDN permique que operadores definam políticas a partir de um único ponto lógico na rede, sendo que estas políticas serão disseminadas dentre os elementos de encaminhamento por parte do controlador. Essa característica pode reduzir drasticamente a complexidade na adaptação da rede à novos requisitos de gerenciamento (ISOLANI et al., 2015).

2.1.2.6 QoS e QoE

Requisitos de QoS e QoE são considerados conceitos chave no sucesso da futura geração de redes, e também podem ser atendidos por meio da utilização de SDN. A realização deste casos de uso se dá através da troca de informações entre aplicações que requisitam qualidade e o controlador da rede. O controlador pode gerenciar o tráfego dos fluxos de forma a atender determinado requisito de qualidade definidos por aplicações (AKELLA; XIONG, 2014; SEDDIKI et al., 2014).

2.1.2.7 Segurança e Dependabilidade

Por fim, destacamos a aplicabilidade de SDN para prover segurança e dependabilidade. A autorização para acessar dados ou recursos em uma rede é geralmente gerenciada pelo operador de rede. Autenticar dispositivos de usuários ou aplicações para utilizar recursos da rede é uma funcionalidade inerente de SDN. Por exemplo, Kim e Feamster (2013) substituíram um portal web, domínios VLAN e um servidor de gerenciamento VLAN pela utilização apenas de uma máquina de estado e algoritmos para gerenciar a autenticação de usuários a partir de um controlador SDN.

Além do aspecto da segurança, para garantir que os recursos de rede irão funcionar em diversos cenários ou caso ocorram falhas na rede, SDN pode ser implantada como uma forma de prover ambientes dependáveis (LI et al., 2014; KREUTZ; RAMOS; VERISSIMO,

2013). Esta implantação pode ser realizada, principalmente, através da inserção de múltiplos controladores na rede ou, ainda, do posicionamento adequado destes controladores (HELLER; SHERWOOD; MCKEOWN, 2012).

2.1.3 Paradigmas de Programação em SDN

Visando introduzir novas funcionalidades da rede, comunidades de pesquisa e da indústria tem utilizado métodos de Engenharia de Software para padronizar linguagens de programação para SDN alcançar seu potencial completo. Esta seção apresenta as contribuições obtidas por diferentes paradigmas de programação e como eles favorecem a programabilidade da rede e a realização dos casos de uso discutidos anteriormente.

Um dos principais paradigmas utilizados na especificação de diversas linguagens de programação para o desenvolvimento de aplicações SDN é o *declarativo* (FOSTER et al., 2011a; REITBLATT et al., 2013). Linguagens de programação declarativas têm sido caracterizadas pela sua forma natural, frequentemente baseada em lógica, sem foco na utilização de operações aritméticas (BORDINI et al., 2006). Este paradigma permite definição de qual ação precisa ser feita na rede, mas não como esta ação irá fazer isso. Note que esta definição se aplica a todas as linguagens de programação declarativas. Para tornar isso possível, um interpretador é utilizado para traduzir "*oque´´* para "*como*ẗt. Um exemplo envolvendo esta abordagem em um cenário SDN é exibido na Figura 6 que utiliza a notação da linguagem Frenetic (FOSTER et al., 2011a) para filtrar pacotes em um fluxo.

Figura 6 – Declaração na linguagem Frenetic (FOSTER et al., 2011a) para filtrar pacotes.

```
Select(packets) *
GroupBy([srcmac]) *
SplitWhen([inport]) *
Limit(1)
```

Fonte: Fonte: O autor (2020).

O exemplo apresenta um alto nível de abstração numa declaração para filtrar pacotes em um determinado fluxo, a qual não requer conhecimentos do programador sobre como implementar a cláusula `Select(packets)` que irá receber e direcionar os pacotes para alguma aplicação ou serviços que esteja a invocando.

Um outro paradigma amplamente utilizado nas linguagens de programação SDN é conhecido como Programação Funcional Reativa, do inglês *Functional Reactive Programming* (FRP). Uma solução viável para o desenvolvimento de aplicações baseadas em eventos, tal como as aplicações SDN. Este paradigma permite que os algoritmos envolvidos em aplicações capturem a propriedade de tempo de fluxo pertinentes a redes SDN (NILSSON; COURTNEY; PETERSON, 2002). A característica reativa de FRP está diretamente

relacionada a um ambiente SDN, onde switches e controladores continuamente trocam informações sobre a chegada de pacotes e aplicam regras para os fluxos correspondentes. A programação funcional reativa (i.e., FRP) também torna possível expressar partes de aplicações como comportamentos que reagem a eventos externos (BAINOMUGISHA et al., 2013). Quando uma linguagem SDN segue o paradigma FRP, ela automaticamente se torna capaz de administrar o fluxo de eventos no decorrer do tempo e as dependências entre dados e computação. A principal ideia por trás do FRP é definir tudo em termos de sinais. Um sinal, neste caso, é um elemento que seus valores mudam no curso do tempo (e.g., se uma variável `v` é igual a `false`, seu valor pode mudar para `true` devido à emissão de um sinal).

Além destes, o paradigma de Linguagens Específicas de Domínio, ou DSL, está presente na maioria das linguagens SDN (LOPES et al., 2016) (mesmo naquelas que utilizam outros paradigmas, como o FRP ou declarativo). Neste paradigma, a especificação da linguagem ocorre focando na resolução de problemas para uma determinada área. Por exemplo, apesar de parte das linguagens de programação em SDN se basearem em GPLs (e.g., Python), estas linguagens servem apenas para a construção de um subconjunto de artefatos que serão parte de uma DSL. Obviamente, as partes da linguagem GPL que não são consideradas essenciais para criar aplicações SDN são removidas da linguagem SDN final. Secundariamente, o tradicional paradigma imperativo, presente em linguagens como Python, C e Java, também é utilizado para definir linguagens SDN. Visando resumir os paradigmas utilizados e seus pontos fortes e fracos com relação à construção de linguagens SDN, elaboramos o Quadro 1.

Esta discussão sobre os paradigmas de programação presentes em propostas para permitir o desenvolvimento de aplicações SDN, juntamente com a análise que será apresentada na Seção 3.2, responde a **Questão de Pesquisa 1**: *"Quais estratégias são mais indicadas para desenvolver aplicações SDN, de forma que requisitos como um alto nível de abstração, a validação de aplicações e a independência de controladores sejam atendidos?"* e atinge parcialmente o **OE1** que refere-se à revisão bibliográfica sobre ferramentas e linguagens (cf. Seção 1.3).

Quadro 1 – Resumo dos paradigmas de programação utilizados em SDN.

Paradigma: FRP	
Prós	**Contras**
Eficiência (da perspectiva de manipular eventos de rede em uma aplicação), permite modelagem de delays e estado da rede como eventos.	Complexidade em criar estruturas, vazamento de memória em caso de excesso de eventos.
Paradigma: DSL	
Prós	**Contras**
Alto nível de abstração, poucas linhas de código, permite a verificação e validação de aplicações, maior produtividade em problema de domínio específico, estrutura de camadas que pode levar a independência de infraestrutura.	Desempenho, o desenho da linguagem é complexo, aplicações sem utilidade fora do domínio para o qual foram implementadas.
Paradigma: Imperativo	
Prós	**Contras**
Flexibilidade, alto nível de abstração, permite ao desenvolvedor definir *como* ele deseja que uma funcionalidade de aplicação seja realizada.	Complexidade em criar estruturas, nem sempre possui nível de abstração adequado.
Paradigma: Declarativo	
Prós	**Contras**
Alto nível de abstração, poucas linhas de código, simplicidade ao focar em "*o que*" o desenvolvedor deseja de uma funcionalidade de aplicação.	Difícil para especificar condições, inflexível.

Fonte: O autor (2020).

2.2 ENGENHARIA DE SOFTWARE BASEADA EM MODELOS

A Engenharia de Software Baseada em Modelos, ou *Model-Driven Engineering* (MDE), é uma abordagem utilizada para reduzir a complexidade no desenvolvimento de software (SCHMIDT, 2006b). Ela é composta das etapas de **análise** e **processo** que se baseiam numa arquitetura de modelos como artefatos essenciais desta forma de desenvolvimento. Este paradigma é conhecido como Desenvolvimento Baseado em Modelos, ou *Model-Driven Development* (MDD), que é basicamente um conceito e processo de desenvolvimento que utiliza modelos, ao invés de unicamente código, na implementação de software.

No conceito de MDE, os modelos utilizados no MDD são definidos através de uma Linguagem de Modelagem Específica de Domínio, ou *Domain-Specific Modeling Language* (DSML). De acordo com Jackson e Sztipanovits (2009), uma DSML formaliza a estrutura e define o comportamento dos modelos dentro de domínios específicos (e.g., registros médicos online, gerenciamento de banco de dados, ou mesmo SDN). Além disso, para compor a abordagem MDE, uma DSML consiste de metamodelos para definir conceitos do domínio e o relacionamento entre eles.

Formalmente, de acordo com Harel e Rumpe (2004), uma DSML é uma 5-tupla composta de: sintaxe concreta (SC), sintaxe abstrata (SA), domínio semântico (DS), mapeamento semântico (MS) e mapeamento sintático (M_s), como definido em (2.1):

$$L =< SC, SA, DS, MS, M_s > \tag{2.1}$$

Onde,

SC: define a notação específica usada para expressar modelos, os quais podem ser gráficos, textuais, ou ambos;

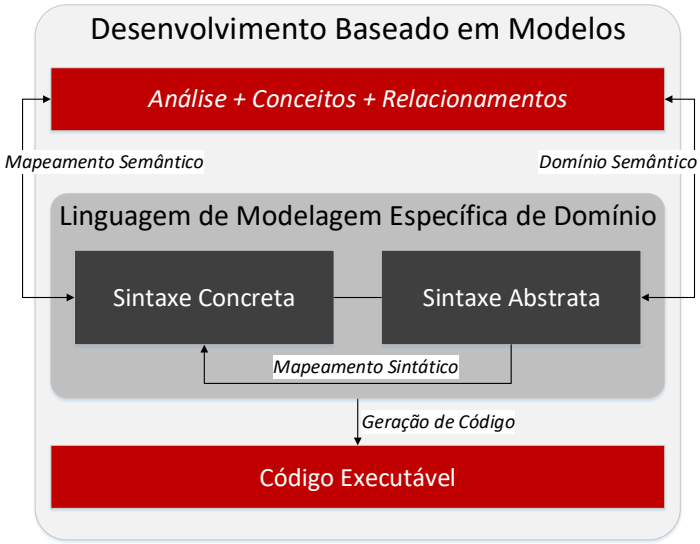
SA: define os conceitos, relacionamentos e restrições de integridade presentes na linguagem;

DS: apresenta o formalismo matemático significativo, nos termos que definem o significado dos modelos;

MS: refere-se à regra sintática de DS;

M_s: atribui construções sintáticas (e.g., gráficos, textos) para elementos de SA.

Figura 7 – Arquitetura envolvendo MDD e o papel da DSML.



Fonte: O autor (2020).

A Figura 7 ilustra o relacionamento entre estes termos e como eles se encaixam na arquitetura de uma DSML.

Um dos pontos benéficos de DSMLs é o suporte natural na utilização de ferramentas de auxílio à Engenharia de Software, ou ferramentas para *Computer-Aided Software Engineering* (CASE), as quais permitem a detecção de erros em aplicações ou mesmo fornecem a prevenção deles. Ferramentas CASE também guiam a realização de padrões de projeto, permitem a verificação da completude de um sistema, suportam a geração de código automática e provêem consistência para especificações (CASE, 1985).

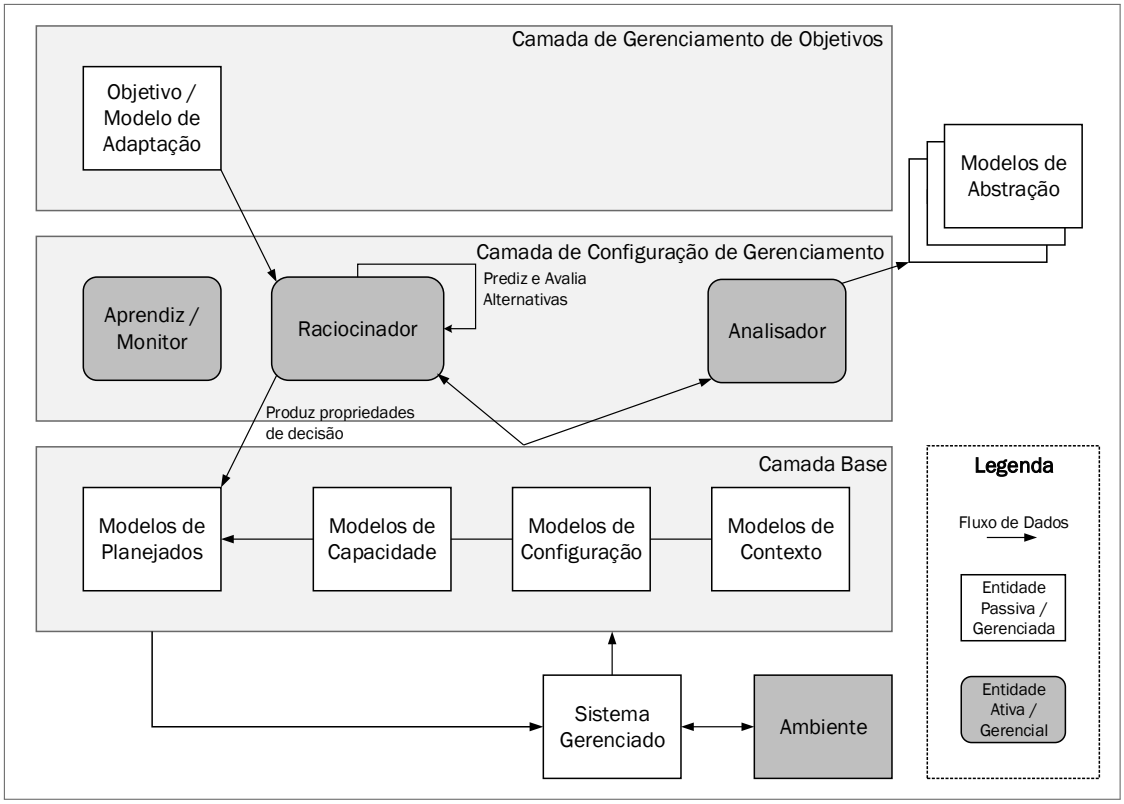
DSMLs podem ser considerados um tipo de DSLs que promovem uma camada de abstração a mais para realizar a geração de código a partir de modelos. Neste escopo, em MDE, tais modelos são mais que itens de documentação; eles são objetos executáveis (SCHMIDT, 2006b). Uma vez que modelos são criados, um código alvo pode ser gerado e então compilado ou interpretado para execução. Tudo isso baseado na especificação de metamodelos (i.e., sintaxe abstrata) que podem ser usados para desenvolver ferramentas CASE utilizando tecnologias livres (e.g., Eclipse Modeling Project (GRONBACK, 2009)) ou proprietárias (e.g., Microsoft Modeling SDK (ÖZGÜR, 2007)).

2.2.1 Modelos em Tempo de Execução

Tradicionalmente, pesquisas em MDE focavam principalmente na utilização de modelos nos estágios de design, implementação e verificação do desenvolvimento (GONZÁLEZ; CABOT, 2014). Estas pesquisas produziram técnicas relativamente maduras e ferramentas que estão atualmente sendo utilizadas tanto na indústria quanto na academia. Entretanto, modelos de software também possuem o potencial de serem usados em tempo de execução, para monitorar e verificar aspectos particulares do comportamento de sistemas de forma *online*, implementando ainda capacidades de auto adaptação (i.e., *self-**) como sistemas *self-healing*, *self-managing*, *self-optimizing* (BLAIR; BENCOMO; FRANCE, 2009). Esta utilização é conceituada como MART, e resulta em um benefício chave que é o fornecimento de uma base semântica mais rica para tomada de decisão em tempo real relacionada às preocupações de sistemas em tempo de execução, as quais são associadas com sistemas autonômicos e adaptativos.

Para estabelecer o conceito de MART na criação de sistemas de software modernos, seus casos de uso típicos e interesses fundamentais, várias arquiteturas de referência foram propostas (BLAIR; BENCOMO; FRANCE, 2009). Por exemplo, Aßmann et al. (2014) definem um arquitetura (ilustrada na Figura 8), onde um sistema MART sempre realiza uma interface com o sistema a ser gerenciado, o qual é monitorado e controlado pelo sistema de gerenciamento MART. Notavelmente, o sistema MART não é diretamente interfaceado ao ambiente. Ao invés disso, sensores e atuadores no sistema gerenciado são utilizados para esse propósito. O sistema gerenciado pode ser qualquer tipo de sistema observável e controlável (e.g., um computador pessoal, redes de sensores, um robô ou um sistema MART novamente).

Figura 8 – Arquitetura MART.



Fonte: Adaptado de (CHAPARADZA et al., 2013)

Nesta arquitetura, cada sistema MART compreende três camadas. De baixo para cima temos:

Uma **camada base** compreendendo modelos do sistema gerenciado;

Uma **camada de configuração de gerenciamento** compreendendo componentes ativos do sistema, realizando troca de informações com o sistema gerenciado; e

Uma **camada de gerenciamento de objetivo** compreendendo os objetivos desejados para o sistema, realizando um laço de repetição interno para trocar informações entre a camada de gerenciamento de objetivos e a camada de configuração de gerenciamento como elemento gerenciado.

A **camada base** compreende quatro tipos de modelos, os quais são abstrações de aspectos específicos do sistema para um dado propósito:

Modelos de Contexo: contém informações relevantes sobre o atual estado do ambiente do sistema gerenciado (e.g., utilização de largura de banda, taxa de utilização do processador). Para manter este tipo de modelo síncrono ao ambiente, são utilizados sensores no sistema gerenciado. Modelos de Contexto não cobrem informações sobre o sistema gerenciado, mas apenas sobre o estado observável do seu ambiente.

Modelos de Configuração: expressam a atual configuração do sistema gerenciado, i.e., seu estado atual. Abordagens MART atuais geralmente proveem uma arquitetura visual do sistema gerencial (i.e., quais serviços estão atualmente implantados e executando). Ambos modelos (i.e., contexto e configuração) cobrem o estado em tempo de execução em um nível mais alto de abstração, sujeito a reflexão traçável e preditiva.

Modelos de Capacidade: descrevem as funcionalidades disponíveis para influenciar o sistema gerenciado (e.g., se componentes de software podem ser adicionados/-removidos, se parâmetros de uma conexão podem ser ajustados), quais atuadores estão disponíveis e como eles podem afetar o ambiente. Geralmente este modelo é estático, mas pode ser atualizado em casos de novos elementos serem adicionados ao sistema (e.g., novos atuadores, novos switches ou rotas).

Modelos Planejados: descrevem uma série de ações (de acordo com os modelos de capacidade) a serem desempenhadas pelo sistema para realizar uma adaptação. Eles representam *scripts* de reconfiguração ou de ação, os quais descrevem como o sistema gerenciado deve ser reconfigurado e como os atuadores do sistema devem ser utilizados para afetar o ambiente.

Em um nível acima, a **camada de gerenciamento de configuração** contém as entidades ativas do sistema MART, as quais fazem uso dos modelos da camada base. Esta camada geralmente compreende um raciocinador, um analisador e, opcionalmente, um aprendiz.

O **raciocinador** avalia futuras configurações alternativas do sistema. Isso inclui (1) realizar análise preditiva, (2) identificar a melhor configuração relacionada aos objetivos especificados na camada acima, e (3) derivar planos de reconfiguração ou ação para estabelecer a configuração de sistema desejada.

O **analisador** possui duas tarefas. Primeiro, ele precisa detectar se todo o sistema deve ser reavaliado. Para isso, o atual estado do sistema precisa ser comparado com os objetivos desse sistema. Se o estado atual estiver diferente dos objetivos, o analisador irá invocar o raciocinador para computar um plano de reconfiguração. Segundo, o analisador ainda abstrai a informação contida nos modelos da camada base. Isso aumenta o nível de abstração dos modelos e, por sua vez, diminui a complexidade da reflexão preditiva. É o analisador que realiza a ponte entre o sistema MART em baixo e alto nível de abstração.

O **aprendiz** possui, também, duas tarefas. De um lado, o aprendiz é responsável por manter os modelos da camada base sincronizados com o sistema. Assim, o aprendiz utiliza sensores no sistema gerenciado para capturar o estado do ambiente,

continuamente observando o próprio sistema gerenciado para atualizar o contexto e o modelo de configuração na camada base. Do outro lado, o aprendiz pode observar o raciocinador para detectar se suas decisões são benéficas ao longo do tempo ou não.

Por fim, a **camada de gerenciamento de objetivos** compreende modelos de objetivo do sistema, os quais são usados pelo raciocinador para avaliar futuras configurações alternativas com relação ao alcance dos objetivos especificados. Notavelmente, estes modelos de objetivos podem e devem permitir mudanças ao longo do tempo. Isso ilustra a necessidade pelo último componente dessa arquitetura de referência.

Utilizamos esta arquitetura como referencial teórico por ela trazer similaridades de camadas em relação às arquiteturas de gerenciamento autonômico de redes que veremos na Seção 2.4.3. Por exemplo, a especificação de um modelo de objetivos, a definição dos componentes raciocinador e analisador, e assim por diante. Além disso, por prever, respectivamente:

1. **Reflexão Preditiva** permitindo a habilidade de raciocinar sobre futuras configurações do sistema.

2. **Tratabilidade por Abstração** que permite ao analisador abstrair a informação utilizada pela raciocinador (possivelmente várias vezes) e reduz a complexidade ao pensar sobre uma tarefa, e, assim, chegar a uma decisão que trate de todo o sistema.

2.3 MODELAGEM DE REDES

Ao longo dos anos, principalmente para redes tradicionais, muitas linguagens e modelos foram propostos para definir objetos, aplicações e o seu gerenciamento (MCCLOGHRIE; PERKINS; SCHOENWAELDER, 1999; MCCLOGHRIE et al., 2001; DMTF, 2012). Estas soluções focam na verificação e simplificação da forma com que as redes e seus serviços são implementados.

Entretanto, cada linguagem de modelagem, bem como os modelos, pode possuir propósitos distintos. Segundo Pras e Schoenwaelder (2003), cada solução pode ser associada a **Modelos de Informação** ou **Modelos de Dados**. Onde, enquanto o principal objetivo dos modelos de informação é modelar objetos gerenciados em um nível conceitual, independente de qualquer implementação específica ou protocolos utilizados para transportar dados, modelos de dados, por outro lado, definem um nível de abstração menor e incluem diversos detalhes sobre tipos e restrições de dados.

Outro aspecto importante é que modelos de informação permitem a definição de relacionamento entre vários objetos, o que pode ser usado na organização da modelagem de dados, delimitando suas funcionalidades. Entretanto, apesar de modelos de informação e

de dados servirem a propósitos diferentes, considera-se que existe uma área em que ambos se entrelaçam. Esta característica é semelhante aos modelos produzidos na engenharia de software orientada a objetos, i.e., fases de design de alto e baixo nível.

O entrelaçamento observado por Pras e Schoenwaelder (2003) também é descrito em Samaan e Karmouch (2009) que tenta separar as categorias de modelagem de redes em **modelos de estrutura**, **comportamento** e **controle**. Ainda assim, alguns tipos de modelos, como os modelos baseados em ontologias (STOJANOVIC et al., 2004), podem flutuar entre pelo menos duas destas categorias (no caso das ontologias: comportamento e estrutura). Nesta seção, englobamos estas categorias no conceito de redes definidas por modelos. Exemplos como o Common Information Model (CIM), o *framework* MDN e a linguagem YANG serão apresentados para descrever o conceito geral de Modelagem de Redes.

Vale ressaltar que, nesta seção, o termo modelo refere-se aos tipos de modelos já apresentados (e.g., informação, comportamento, controle). Nos demais casos, ao longo da proposta deste trabalho, o termo modelo será especificado quando se referir a outra finalidade, e.g., modelos estocásticos ou estatísticos.

2.3.1 Common Information Model

O CIM (DMTF, 2012) é um modelo conceitual de informação, definido pela *Distributed Management Task Force* (DMTF) para descrever entidades computacionais e de rede. A utilização do CIM torna possível modelar e trocar informações entre diferentes operadores e dispositivos. Por exemplo, ao utilizar o CIM, podemos modelar cada entidade (e.g., física, lógica) como um objeto CIM, além de definir o relacionamento entre ele e outras partes do modelo.

Uma informação modelada no CIM é organizada em *profiles*. Um *profile* é o modelo CIM e o comportamento associado a um domínio de gerenciamento particular (e.g., classes, abstrações e métodos). Tais *profiles* fornecem um caminho único para descrever o domínio de gerenciamento, o que contribui para interoperabilidade entre os elementos que compartilham o mesmo modelo.

Vale ressaltar que o CIM, além de permitir a troca e representação de informações, também possibilita o gerenciamento e controle dos elementos representados. Isso é feito através da implementação de um sistema que materialize as possíveis representações do modelo CIM. Por exemplo, existe uma instância do modelo CIM, chamada CIM Network (PILZ; SWOBODA, 2004), específica para representar elementos de rede tradicionais (e.g., rotas, protocolos) e outra para modelar redes SDN (PINHEIRO et al., 2013). Essa representação de informações pode ser feita de maneira textual ou gráfica (e.g., UML).

2.3.2 Model-Driven Networking

Basicamente, a abordagem de Redes Dirigidas por Modelos (MDN) (LOPES et al., 2015) realiza uma associação entre o paradigma SDN e técnicas de MDE, visando criar aplicações SDN através de um processo de desenvolvimento baseado em modelos. Esta associação é feita através da especificação de um metamodelo para SDN, da definição de elementos gráficos que representem conceitos em SDN e de *templates* para geração de código. Isso resulta em um processo de desenvolvimento MDD apoiado por uma DSML, assumindo uma característica de modelos informacionais (ainda que propriedades de dados também possam ser definidas). Tal abordagem é baseada nos seguintes conceitos:

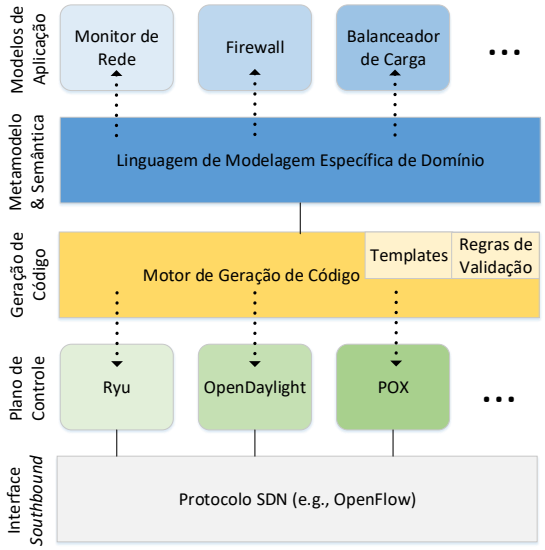
- **Tempo**: permitindo que operadores de rede implementem políticas onde o comportamento da rede depende de determinada data ou horário;
- **Uso de Dados**: especifica restrições onde o comportamento da rede depende da quantidade de dados utilizada pelos fluxos;
- **Nível**: a rede deve permitir que operadores especifiquem privilégios para diferentes níveis de usuários ou dispositivos;
- **Fluxo**: representam as informações que trafegam na rede, possuindo atributos que variam de acordo com a origem, destino e protocolos utilizados na comunicação entre dois ou mais pontos distintos da rede.

O conceito de MDN foi estabelecido seguindo dois princípios fundamentais: i) alto nível de abstração para o desenvolvimento de aplicações SDN; e ii) independência entre modelos de aplicações e controladores. A Figura 9 apresenta o resultado da arquitetura construída em cima destes princípios.

Na camada superior da arquitetura MDN temos a camada de Modelos de Aplicação, que envolve os modelos criados na interface com o usuário através de uma ferramenta CASE (nomeada *MDN Editor*). Estes modelos são criados de acordo com conceitos e relacionamentos que formam a camada inferior, chamada Metamodelo & Semântica. Depois da verificação de certa aplicação modelada (a partir de sua consistência com o metamodelo), a camada de Geração de Código é responsável por transformar modelos de alto nível em código fonte que irá interagir com um controlador alvo para execução da aplicação. Por fim, o MDN espera que o controlador utilize a SBI para, a partir da execução da aplicação, gerar comandos e regras OpenFlow, programando o plano de encaminhamento.

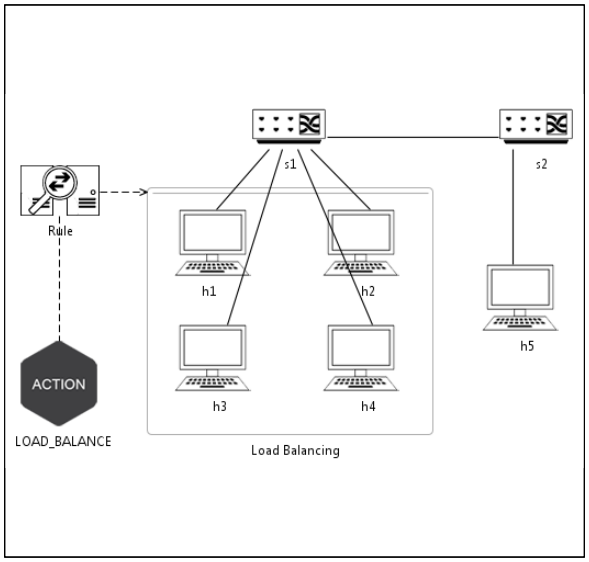
Visando ilustrar o funcionamento da abordagem MDN, no lado esquerdo da Figura 10, apresentamos um modelo de aplicação pra balanceamento de carga criado com o *MDN Editor*. Ao lado direito desta figura, encontra-se o trecho de um dos *templates* utilizados na geração do código que por fim é utilizado pelo controlador na implantação da aplicação.

Figura 9 – Arquitetura da abordagem MDN.



Fonte: O autor (2020).

Figura 10 – Na esquerda, exemplo de uma aplicação de balanceamento de carga modelada no MDN Editor. Na direita, trecho de um dos templates utilizados na geração de código.




```
1  [% operation actionLoadBalancing(rule : Any) : String { %]
2     virtual_ip=IPAddr("[%=rule.targetGroupRule.ip %]")
3     host = {}
4     [% var index = 0;
5     for (host in rule.targetGroupRule.hostsGroup.all) { %]
6        host[[%=index%]]={'ip':IPAddr("[%=host.ip%]")}
7        [% index++; %]
8     [% } %]
9     total_hosts = len(host)
10    host_index = 0
11    [...]
12    if (not event.parsed.find("ipv4")):
13       return EventContinue
14    if (msg.match.nw_dst != virtual_ip):
15       return EventContinue
16    [...]
17    index = host_index % total_hosts
18    selected_host_ip = host[index]['ip']
19    host_index += 1
20    [...]
21    msg.actions.append(of.ofp_action_nw_addr(
                  of.OFPAT_SET_NW_DST,selected_host_ip))
22    event.connection.send(msg)
23    [...]
24 [%}%]
```

Fonte: O autor (2020).

É no modelo que propriedades dos elementos, a lógica e as condições de execução são definidas.

Neste contexto, a DSML que compõe a abordagem MDN é baseada em um metamodelo. É a partir deste metamodelo que as demais tuplas da DSML são definidas. A combinação desta DSML e o processo MDD que baseiam o conceito por trás do MDN.

No Capítulo 3 descrevemos como os elementos da arquitetura SDN também são representados no metamodelo. Descrições como *Controller*, *Switch* e *Host*, compõem uma

Quadro 2 – Entidades que compõem o metamodelo MDN.

Entidades	Descrição
NetworkNode	Envolve o conceito geral de elementos da infraestrutura de rede (e.g., nó, controlador, switch).
Link	Componente que conecta elementos de rede.
Controller	A parte lógica do ambiente SDN, na qual switches são gerenciados, aplicações implantadas e o comportamento da rede programado.
Host	Clientes ou servidores da rede.
Traffic	Compreende os dados utilizados ou transitados através da rede.
Flow	Definição de propriedades para os fluxos de dados na rede (e.g., origem, destino, portas).
Policy	Diretrizes que definem o comportamento da rede.
RelationalOperator	Compõem regras condicionais nas aplicações.
Action	É a representação de parte da especificação OpenFlow que define um conjunto de ações possíveis para os fluxos.
Time	Representação de que o comportamento da rede pode mudar ao longo do tempo.
Conditions	Conjunto de condições que podem comparar o estado atual da rede antes da aplicação de alguma regra.

Fonte: O autor (2020).

entidade *Sdn*, i.e., o diagrama de modelos base para a ferramenta MDN, chamada *MDN Editor*. Os nós desse diagrama também possuem diversos relacionamentos, como *switchController*, *hostSwitch* e *switch*. Todos esses relacionamentos são baseados na especificação do protocolo OpenFlow (ONF, 2013) e na arquitetura SDN. A implementação da arquitetura definidas pelo MDN segue o fluxo do *Graphical Modeling Framework* (GMF) (KOLOVOS et al., 2009).

2.3.3 YANG/NETCONF

O *Network Configuration protocol* (NETCONF) e a linguagem de modelagem de dados YANG visam simplificar e agilizar a configuração de dispositivos de rede (SHAFER, 2011). Apesar de NETCONF não ser uma abordagem de modelagem de redes, sua associação com a linguagem de modelagem YANG ajuda a ter uma visão de como este tipo de abordagem permite facilitar o gerenciamento e a configuração de redes. Indo além, ao permitir alcançar característica de automação, já que, atualmente, tal característica encontra-se bloqueada pela necessidade de escrever *scripts* de configuração específicos para cada dispositivo ou pela dependência funcionalidades providas por fornecedores de dispositivos de rede. Por exemplo, o protocolo NETCONF é definido para realizar automaticamente transações seguras de configuração para uma variedade de dispositivos, modificando listas de controle de acesso, adicionando *Virtual Private Network* (VPN), mantendo flexibilidade e independência de fornecedores.

A seguir detalhamos a descrição de cada um destes protocolos.

2.3.3.1 YANG

Definida como uma linguagem utilizada para criar modelos de dados para o protocolo NETCONF (usado para configurar dispositivos de rede), YANG é uma referência da *Internet Engineering Task Force* (IETF) (BJORKLUND, 2010) que permite a implementação de dados hierárquicos para operações baseadas no NETCONF. Ela estrutura os modelos de dados em módulos e submódulos, permitindo ainda a descrição de restrições para serem aplicadas aos dados, restringindo a visualização ou valor dos nós na presença ou valor de outros nós da hierarquia. Além disso, os dados são descritos utilizando tipos bem definidos (e.g., `string`, `int`) ou tipos adicionais especificados pelo desenvolvedor. Por exemplo, a Figura 11 apresenta a implementação simples de um modelo YANG juntamente com um dos tipos de dados utilizados pela linguagem. Tal implementação se torna um arquivo *eXtensible Markup Language* (XML) para o NETCONF que, por sua vez, realiza a definição de nome para determinado elemento da rede.

Figura 11 – Exemplo de implementação YANG.

```
leaf host-name {
  type string;
  description "Nome do host para este sistema";
}
```

Fonte: O autor (2020).

A utilização de YANG pode, ainda, ser feita para definir formatos de notificação de eventos na rede, por exemplo, ao definir a assinatura de uma determinada chamada de procedimento remoto, invocada através do protocolo NETCONF. Uma de suas vantagens é a possibilidade de ser convertida em outros formatos, e.g., XML ou JSON, o que permite sua aplicação por outros protocolos.

Composta por uma arquitetura que balanceia uma modelagem de dados em alto nível com codificação de bits em baixo nível, YANG permite ao desenvolvedor a visualização de módulos com visões de alto nível enquanto ainda torna possível entender como os dados serão codificados em baixo nível pelas operações NETCONF.

Em uma visão geral, a linguagem é baseada em módulos e submódulos. Um módulo contém três tipos de declarações: o cabeçalho, que descreve o módulo e fornece informações; a revisão, que provê informações sobre a história do módulo; e a declaração de definição, que é o corpo do módulo onde os dados são definidos. Os submódulos, por sua vez, são baseados na necessidade de um módulo, caso este se torne muito complexo, é possível transcrever parte da definição de dados para um submódulo.

Sendo assim, YANG dispõe de uma arquitetura e componentes que a inserem no contexto de representação e definição da rede a partir de modelos. A seguir, visando exem-

plificar a aplicação do modelo YANG, detalhamos o protocolo NETCONF, mencionado em parágrafos anteriores.

2.3.3.2 NETCONF

O NETCONF (ENNS et al., 2011) é um protocolo definido pela IETF para instalar, manipular e excluir configurações de dispositivos de rede. As operações deste protocolo são baseadas em uma camada de *Remote Procedure Call* (RPC) (NELSON, 1981) utilizando codificação XML e fornecendo um conjunto básico de operações para editar e consultar configurações em um dispositivo de rede (ENNS et al., 2011).

Em Shafer (2011), apresenta-se uma arquitetura demonstrando como associar estes dois protocolos e a possibilidade de interoperabilidade caso diferentes fornecedores os utilizem nas especificações de seus dispositivos. Por exemplo, uma camada de gerenciamento mais simples entre os dispositivos de rede. Além disso, especificamente para o protocolo NETCONF, são descritas as seguintes características chave: configurações baseadas em transações (permitindo a configuração na amplitude da rede), suporte à *rollback* e a separação clara dos estados de operação e configuração.

Sob a perspectiva de SDN, uma instância do protocolo NETCONF é o OF-Config, um protocolo baseado no NETCONF para gerenciar a configuração de dispositivos OpenFlow físicos − para dispositivos virtuais, como o Open vSwitch (OVS), existe o OVSDB, ainda que existam esforços para utilizar o próprio OF-Config neste cenário (ČEJKA; KREJčí, 2016).

2.4 GERENCIAMENTO AUTONÔMICO DE REDES

A evolução nas tecnologias de comunicação nos últimos anos ocorreu em vários tipos de ambientes (e.g., com fio, sem fio, óptico), permitindo diversos tipos de aplicações e serviços demandados por um crescente número de usuários conectados a estes ambientes. Tal evolução foi acompanhada, também, por um crescente nível de heterogeneidade de dispositivos, protocolos e etc., tornando altamente complexo o gerenciamento dos ambientes de comunicação.

Para superar a complexidade de gerenciamento que originou-se nas redes, uma iniciativa chamada Gerenciamento Autonômico de Rede, ou ANM, similar ao conceito de Computação Autonômica, foi proposta como uma solução (MOVAHEDI et al., 2012). Porém, enquanto a Computação Autonômica tem por essência o desenvolvimento de sistemas computacionais auto-gerenciáveis (os quais gerenciam a si mesmos a partir de objetivos definidos por administradores) (KEPHART; CHESS, 2003), as Redes Autonômicas referem-se à capacidade da rede em realizar sua operação gerenciando a si mesma, mesmo no caso

de mudanças no ambiente.

De acordo com Samaan e Karmouch (2009), sistemas ANM possuem a capacidade de antecipar-se, diagnosticar e precaver-se de prejuízos que possam ser causados às funcionalidades da rede subjacente, de forma independente e autonômica, guiadas por um conjunto de objetivos de alto nível com a mínima intervenção humana. Esta visão é similar a de Movahedi et al. (2012), que complementa a definição mencionando que esta autonomicidade pode ser atingida através de uma arquitetura de gerenciamento distribuída, onde um conjunto de elementos cooperam uns com os outros para prover um gerenciamento autonômico convergente.

É importante mencionar que estas discussões sobre ANM surgiram após a iniciativa da IBM fomentando a computação autonômica (HORN, 2001). Logo após esta iniciativa, o famoso modelo para alcançar autonomia em sistemas, denominado modelo *Monitor, Analyze, Plan, Execute, and Knowledge* (MAPE-K), foi apresentado por Kephart e Chess (2003), juntamente com a sua definição de que elementos autonômicos devem possuir componentes de monitoramento, análise, planejamento e execução para que permitam a auto-adaptação à mudanças do ambiente. Neste cenário, tais componentes devem considerar uma base de conhecimento comum ao realizar suas ações.

Em geral, a adaptação refere-se às mudanças que tornam os componentes de um sistema mais adaptados ao seu ambiente (OREIZY et al., 1999). De forma mais abrangente, a **adaptação de sistemas** (ou, no caso deste trabalho, redes) diz respeito à aplicação de uma ou mais estratégias, algoritmos, regras ou configurações que visem modificar um ou mais aspectos do sistema com a finalidade de atingir um conjunto de objetivos de alto nível. A adaptação, então, é parte fundamental da ação de planejamento que discutimos aqui. Complementarmente, a **adaptação autonômica** define a habilidade do sistema de realizar operações de adaptação utilizando seu próprio conhecimento para decidir qual, quando, onde e como certa adaptação será desempenhada, sem qualquer intervenção externa no processo de decisão.

2.4.1 Visão Geral

O termo autonomicidade é assunto de diversas abordagens e soluções por diversas comunidades de computação. Visando dissociar o uso do termo autonomicidade em comparação ao conceito similar, chamado automaticidade, duas definições devem ser realizadas:

- **Automaticidade** (SCHMID; SIFALAKIS; HUTCHISON, 2006): refere-se à habilidade de realizar uma ou mais tarefas sem qualquer intervenção manual externa. Porém, isso não inclui questões de otimização de desempenho. O enfileiramento de pacotes, por exemplo, pode ser considerado uma execução automática.

- **Autonomicidade** (WHITE et al., 2004): é definida pela capacidade de auto-gerenciamento

dada por um conjunto de objetivos definidos por administradores. Ou seja, objetivos de alto nível definem para o sistema quais são suas metas e o sistema *autonomicamente* tenta atingi-las da melhor maneira possível. Consequentemente, um sistema autonômico é capaz de monitorar seu próprio desempenho e adaptar-se de acordo com o resultado do monitoramento, além de otimizar a utilização de recursos e superar possíveis eventos inesperados que ocorram.

Assim, em autonomicidade, as principais propriedades de auto-gerenciamento (MURCH, 2004; MOVAHEDI et al., 2012) são as seguintes:

- **Auto-Configuração**, ou *Self-Configuring*: esta propriedade refere-se à capacidade do sistema para configurar e reconfigurar a si mesmo de acordo com políticas de alto nível em um ambiente passível de mudança.
- **Auto-Otimização**, ou *Self-Optimizing*: o objetivo da auto-otimização é permitir uma operação eficiente do sistema, mesmo com a ocorrência de eventos inesperados.
- **Auto-Cura**, ou *Self-Healing*: consiste na capacidade de descoberta e reparo de potenciais problemas para garantir uma execução correta do sistema. Esta propriedade pode ser alcançada por algoritmos de predição e tomadas de decisão proativas ou prevendo falhas ou reduzindo o impacto destas.
- **Auto-Proteção**, ou *Self-Protecting*: define a habilidade do sistema em proteger-se daquilo que pode comprometer o alcance de seus objetivos. Envolve a proteção contra ataques maliciosos, tentativas de intrusão ou falhas inadvertidas.

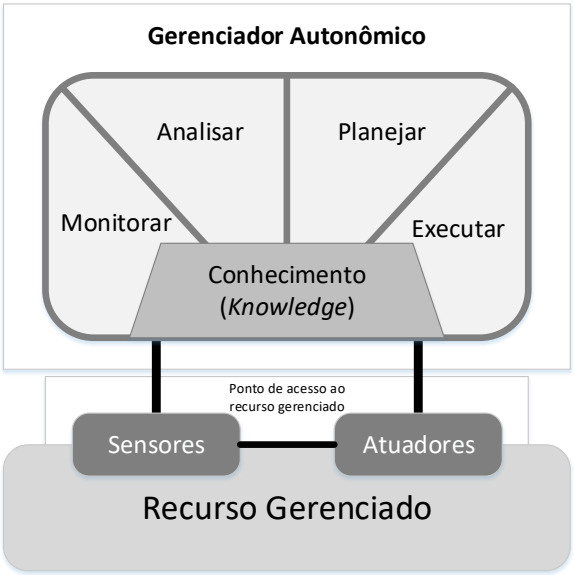
Em cenários onde a **Auto-Otimização** possui alta prioridade, como no contexto de SDN, Lemos et al. (2013) argumenta-se que uma abordagem centralizada é mais fácil de ser implementada e permite uma otimização mais direta. Por outro lado, em uma abordagem descentralizada, e.g., com diversos ciclos MAPE distribuídos, satisfazer a objetivos globais é tido como um problema complexo. Desta forma, devemos considerar o comprometimento da otimização na maioria dos casos. A literatura traz ainda outros modelos de implementação de sistemas autonômicos, como abordagens baseadas em *middlewares*, protocolos e descrições arquiteturais (SALEHIE; TAHVILDARI, 2009). Nesta tese, os conceitos do ciclo MAPE são levados em consideração dada a sua maturidade e adequabilidade para o contexto de um ciclo de adaptação dinâmico que estamos considerando.

2.4.2 Implementação do Gerenciamento Autonômico

Uma das formas de realização do gerenciamento autonômico de redes depende da implementação de blocos funcionais e requisitos que podem ser classificados de acordo

com a composição do modelo MAPE-K (cf. Figura 12). As seguintes subseções, baseando-se em Schmid, Sifalakis e Hutchison (2006), Samaan e Karmouch (2009) apresentam as possibilidades e direcionamentos de implementação destes blocos.

Figura 12 – Modelo conceitual MAPE-K.



Fonte: O autor (2020).

2.4.2.1 Base de Conhecimento

Este ponto chave na realização do gerenciamento autonômico deve permitir descrever um modelo do sistema gerenciado de forma correta e precisa. Nas arquiteturas ANM, a base de conhecimento é implementada através da criação de um *knowledge base system* (KBS) (SAMAAN; KARMOUCH, 2009), o que envolve desde a especificação do conhecimento que deve ser fornecido pelo sistema até a implementação de bancos de dados simples e mecanismos de raciocínio. Além disso, dois tipos de base de conhecimento podem existir no sistema autonômico: (1) conhecimento sobre o domínio, que envolve uma conceitualização do domínio gerenciado; e (2) conhecimento de controle, o qual representa os caminhos para gerenciar e controlar o sistema modelado, por exemplo, um conjunto de problemas relacionados a um conjunto de soluções.

Samaan e Karmouch (2009) citam a modelagem de redes, através de modelos de estrutura, controle e comportamento (discutidos na Seção 2.3) como uma das principais formas de construir uma base de conhecimento, ainda que modelos genéricos também possam ser utilizados (e.g., ontologias, modelos biológicos (DIAO et al., 2005)). Neste caso, modelos formais, e.g., metamodelos, também demonstram grande viabilidade para serem utilizados na implementação de uma base de conhecimento.

2.4.2.2 Monitoramento Autonômico

Uma ação de monitoramento de rede realiza uma coleção de medições necessárias para determinar a saúde da rede subjacente, a qualidade de serviço atingida, assim como a possibilidade de falhas ou ataques. Em geral, ferramentas de monitoramento baseiam-se em um conjunto sistemático de medições de parâmetros pré-determinados coletados em dispositivos da rede (e.g., contadores, registros, ou ainda métricas - como no caso protocolo OpenFlow). Porém, no monitoramento autonômico, as abordagens devem continuamente ajustar suas operações para manter o equilíbrio entre a visão precisa da rede e a sobrecarga que esta precisão pode causar no desempenho da rede.

Samaan e Karmouch (2009) definem as seguintes questões que devem ser levadas em consideração ao implementar um bloco de monitoramento autonômico:

- **Monitoramento ativo versus passivo**: Em ambas abordagens de rede (i.e., tradicional ou autonômica), o monitoramento pode ser realizado de forma ativa ou passiva. O monitoramento passivo é limitado a receber dados de um conjunto de sensores espalhados pela rede. Do outro lado, o monitoramento ativo obtém métricas da rede requisitando e enviando mensagens para verificação do estado da rede.

- **Monitoramento distribuído versus centralizado**: Funcionalidades do monitoramento autonômico podem ser iniciadas e controladas por gerenciadores autonômicos centralizados que coletam e consolidam os dados obtidos de diferentes locais. Uma outra abordagem é distribuir essa funcionalidade através de um conjunto de nós autonômicos individuais, os quais interagem entre si para alcançar o requisito de monitoramento desejado.

- **Granularidade do monitoramento**: A unidade de granularidade do monitoramento define o nível de detalhe ao capturar dados de utilização da rede gerenciada. Por exemplo, as medições podem ser realizadas em byte, pacotes, fluxos, e assim por diante.

- **Tempo de monitoramento**: A maioria das entidades de monitoramento realizam suas atividades em tempos fixados. De maneira diversa, existem abordagens que podem realizar as medições baseadas em eventos ou sob demanda, disparando requisições de informações somente quando necessário.

- **Programabilidade do monitoramento**: Espera-se de um monitoramento autonômico que ele possa, dinamicamente, modificar sua operação conforme necessário, requisitando um grau de auto-programabilidade. A maneira mais simples de alcançar essa propriedade é através de parâmetros dinâmicos. Por exemplo, o monitoramento pode ajustar-se para adaptar a sua granularidade conforme o nível de carga na rede, ou ainda transitar entre os medições ativas e passivas.

2.4.2.3 Análise Autonômica

A segunda atividade de um ANM, analisar as informações de monitoramento, já é realizada em redes tradicionais, embora limitada por apenas interpretar os dados coletados em uma descrição do estado da rede gerenciada. Essa atividade pode ser referenciada como diagnóstico da rede, e tem sua realização feita a partir de regras pré-existentes ou heurísticas. Além disso, de acordo com Lemos et al. (2013), no ciclo de controle, há cinco formas de interagir e analisar o estado do recurso gerenciado (cf. Figura 12). Na primeira forma, denominada **Controle Hierárquico**, os ciclos do MAPE-K estão presentes em todos os níveis da hierarquia do sistema ANM. Na segunda forma, o sistema autonômico é dividido em um **Padrão de Mestres e Escravos** (i.e., *master* e *slave*), onde os mestres realizam a análise autonômica discutida nessa seção (inclusive o planejamento das ações), enquanto os escravos apenas monitoram e executam ações. A terceira forma, estabelece um **Planejador Local** nos dispositivos que irão fazer parte do sistema ANM (separando-os por regiões). Estas regiões irão seguir a execução estabelecida por um elemento semelhante ao *master* definido para o padrão anterior. Na quarta forma, nomeada **Totalmente Descentralizada**, o sistema ANM é distribuído em toda a rede, e cada parte possui um ciclo MAPE próprio. Por fim, a quinta forma, chamada **Compartilhamento de Informação**, define que os componentes MAPE também serão distribuídos dentre as partes de um sistema, porém, apenas o componente de monitoramento (M) compartilha informações com seus pares.

2.4.2.4 Planejamento e Execução Autonômica

A terceira e a quarta funcionalidades de um gerenciador autonômico, respectivamente as funcionalidades de **planejar** as ações de adaptação e **executar** estas ações, também possuem peculiaridades que devem ser observadas em suas implementações. Desta forma, planejar, em operações de rede, refere-se à tarefa de configurar topologias de rede, operações e serviços e é realizada de maneira *offline* por administradores de rede (WICKBOLDT et al., 2015). Executar, por outro lado, refere-se à execução de operações e serviços de rede de acordo com a rede que foi planejada. Em redes tradicionais, estas funções são realizadas manualmente ou parcialmente automatizadas. Recentemente (YAO et al., 2018; TANGARI et al., 2018), a adaptação de rede tem sido utilizada para refletir o contínuo ajuste fino da rede visando atender ao conjunto de objetivos de gerenciamento.

2.4.3 Arquiteturas de Rede Autonômicas

Conforme já discutido, ao mesmo tempo em que o gerenciamento de redes se torna cada vez mais complexo, a computação autonômica vai se tornando essencial para manter este gerenciamento possível – dada a dinamicidade de redes SDN, envolvendo suas aplicações e eventos que podem ocorrer (WICKBOLDT et al., 2015). Neste cenário, o termo Redes Autonômicas, ou AN, é utilizado para definir a aplicação de princípios de autonomia ao gerenciamento de redes.

Por definição (MURCH, 2004), uma rede autonômica opera e **tenta** alcançar seus objetivos através do gerenciamento de si mesma, sem qualquer intervenção externa, mesmo nos casos de mudanças no ambiente. Neste ponto, diferentes autores apresentam múltiplas visões quanto às possíveis formas da arquitetura utilizada para alcançar a propriedade de autonomicidade. Por exemplo, enquanto Schmid, Sifalakis e Hutchison (2006) defendem uma arquitetura distribuída composta por um conjunto de gerenciadores autonômicos juntamente com os componentes gerenciados, Samaan e Karmouch (2009) mencionam que esta arquitetura também pode ter componentes centralizados, ao menos para algumas propriedades do sistema autonômico (e.g., monitoramento). Movahedi et al. (2012) discutem essa questão, categorizando arquiteturas ANM em horizontais e hierárquicas. Nesta categorização, arquiteturas horizontais definem a distribuição de elementos autonômicos na rede (semelhante à visão distribuída de Schmid, Sifalakis e Hutchison (2006)) cooperando entre si, enquanto arquiteturas hierárquicas consideram uma autonomia da rede realizada a partir de uma camada central acima dos elementos de rede, coordenando-os de forma autonômica e de acordo com os objetivos do gerenciamento.

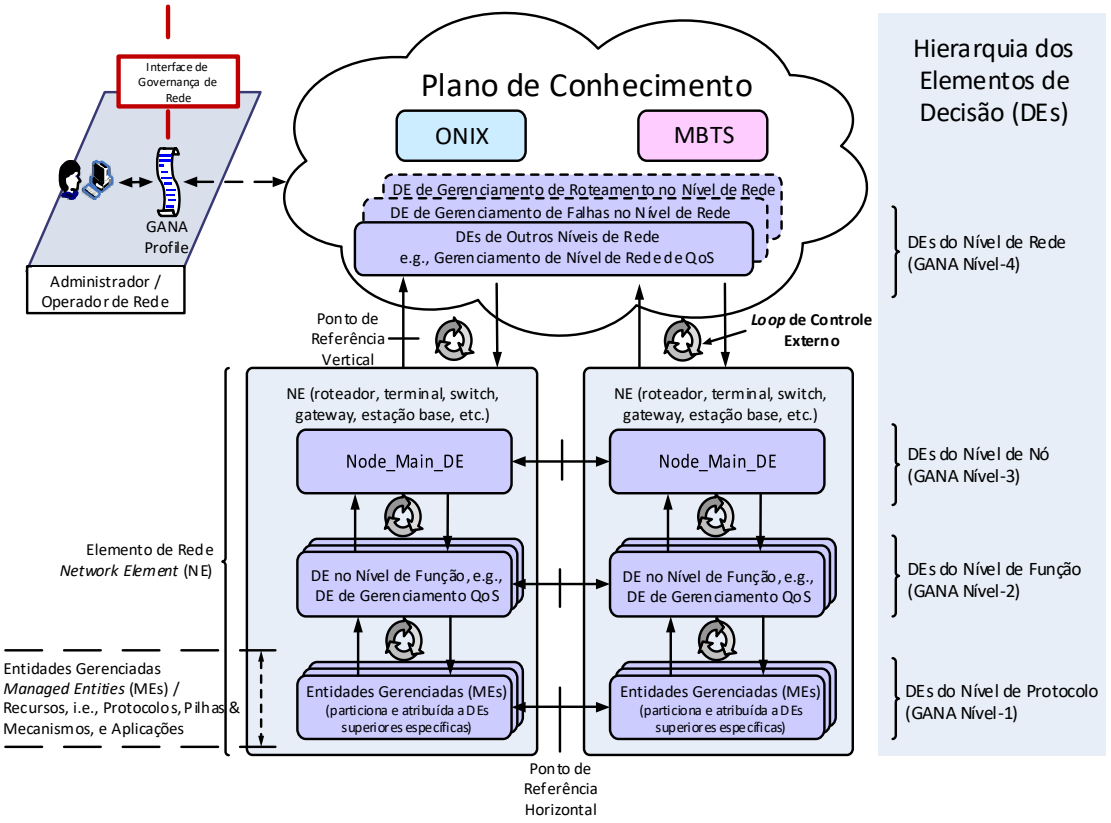
Neste contexto, algumas propostas de arquiteturas autonômicas de rede têm surgido visando conciliar diferentes visões de como a implementação e o gerenciamento destas redes devem ser feitos. Nas seções a seguir, discutiremos brevemente os detalhes de algumas destas arquiteturas, visando destacar aspectos relevantes e similaridades.

2.4.3.1 GANA

Projetada pelo projeto EFIPSANS do consórcio europeu FP7, a GANA é uma arquitetura que prevê o gerenciamento autonômico de redes de forma horizontal e hierárquica. Esse gerenciamento é baseado em blocos funcionais, pontos de referência associados a tais blocos e informações características através dos pontos de referência (WODCZAK et al., 2011).

GANA (CHAPARADZA et al., 2013) é considerada uma arquitetura genérica no sentido de que ela define e separa conceitos genéricos e princípios arquiteturais associados ao domínio de redes autonômicas, redes cognitivas e auto-gerenciamento das estratégias de implementação, detalhes e métodos que podem ser usados para implementá-los. São três

Figura 13 – Ilustração da arquitetura GANA.



Fonte: Adaptado de Chaparadza et al. (2013).

principais componentes: *Decision-making elements* (DEs), também chamados de funções autônomas, são responsáveis pela autonomicidade no gerenciamento do sistema e de serviços, parâmetros e recursos da rede; *Managed Entities* (MEs), que referem-se aos recursos gerenciados (incluindo elementos não físicos); e o *Knowledge Plane* (KP), que permite gerenciamento avançado e controle inteligente nos níveis de *Element Management* (EM), *Network Management* (NM) e *Operation and Support System* (OSS). Estes componentes comportam-se de acordo com os GANA *Profiles* que são especificados por operadores humanos e fornecidos como entrada para a rede autonômica através do KP. A Figura 13 ilustra estes e outros componentes desta arquitetura.

A descrição da arquitetura define que elementos de rede, os *Network Elements* (NEs), devem ser implementados com DEs, e, em uma comunicação *End-to-End* (E2E), pares de NEs e DEs interagem de forma distribuída entre em si, de acordo com o ciclo de controle. Nesta perspectiva, GANA define quatro níveis para os DEs (CHAPARADZA et al., 2013):

- **Nível de Rede**: possuem como entrada informações da rede em geral, desenhados para operar fora dos ciclos de controle com base na visão geral da rede. Neste nível, DEs interagem com o plano de conhecimento (i.e., KP) da arquitetura, que por sua vez é composto pelos componentes *Overlay Network for Information eX-*

change (ONIX), o qual permite a auto-descoberta de informações e recursos de uma rede autonômica, e o *Model-Based Translation Service* (MBTS), que serve como intermediário entre os DEs do KP e os NEs (físicos ou virtuais);

- **Nível de Nó**: refere-se a um DE que define aspectos do comportamento de um elemento de rede como um todo, e.g., segurança, gerenciamento de falhas, auto-configuração e descoberta, e resiliência;
- **Nível de Função**: é relacionado aos DEs que agrupam protocolos e mecanismos abstraídos por uma função de gerenciamento (e.g., roteamento, encaminhamento, QoS);
- **Nível de Protocolo**: refere-se a qualquer ME, tal como um protocolo ou outro mecanismo fundamental que pode exibir ciclos de controle intrínsecos e outros DE associados, tal como protocolos OSPF, o qual pode ser considerado uma instanciação de um DE no nível de protocolo.

2.4.3.2 Modelo de Referência AN

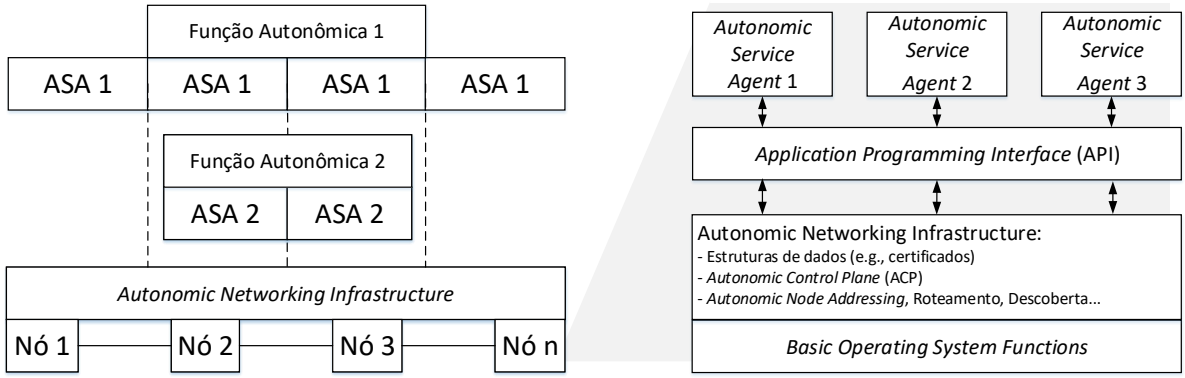
O grupo de trabalho ANIMA do IETF propôs o Modelo de Referência AN (BEHRINGER et al., 2017), o qual define vários elementos de uma rede com funções autônomas e como estas entidades devem trabalhar juntas, tornando possível encaixá-lo na categoria das arquiteturas horizontais. A Figura 14 ilustra a visão de uma AN para este modelo. Ela consiste de nós autonômicos, os quais interagem diretamente uns com os outros. Estes nós autonômicos fornecem um conjunto de capacidades ao longo da rede, chamado *Autonomic Networking Infrastructure* (ANI). O ANI fornece funções como endereçamento, negociação, sincronização e descoberta. Diferentemente da arquitetura GANA, este modelo de referência não prevê uma base de conhecimento para as funções autonômicas.

Funções autonômicas neste modelo geralmente abrangem diversos, senão todos, nós na rede. As entidades de função autonômica são chamadas *Autonomic Service Agents* (ASA), as quais são instanciadas nos nós da rede. Em uma visão horizontal, funções autonômicas abrangem a rede tanto quanto a ANI. Em uma visão vertical, um nó sempre implementa a ANI, e ainda pode ter uma ou mais entidades ASA.

Assim, neste modelo, ANI é a fundação para implementação de funções autonômicas. Em um nível mais baixo, essa implementação é baseada também nos elementos de rede autonômicos. Nestes elementos, os ASAs usam informações para prover auto-conhecimento, conhecimento da rede (através de descoberta), intenções e ciclos de informação. Em tais nós autonômicos, os ASAs estão definidos em um nível acima da ANI, a qual fornece serviços (e.g., roteamento, descoberta) para os agentes.

Essa organização pode ser visualizada na Figura 14, onde na esquerda temos a visão de alto nível de uma rede AN e na direita o modelo de um nó autonômico, segundo a arquitetura proposta pelo grupo ANIMA.

Figura 14 – Ilustração do Modelo de Referência AN.



Fonte: Adaptado de Behringer et al. (2017)

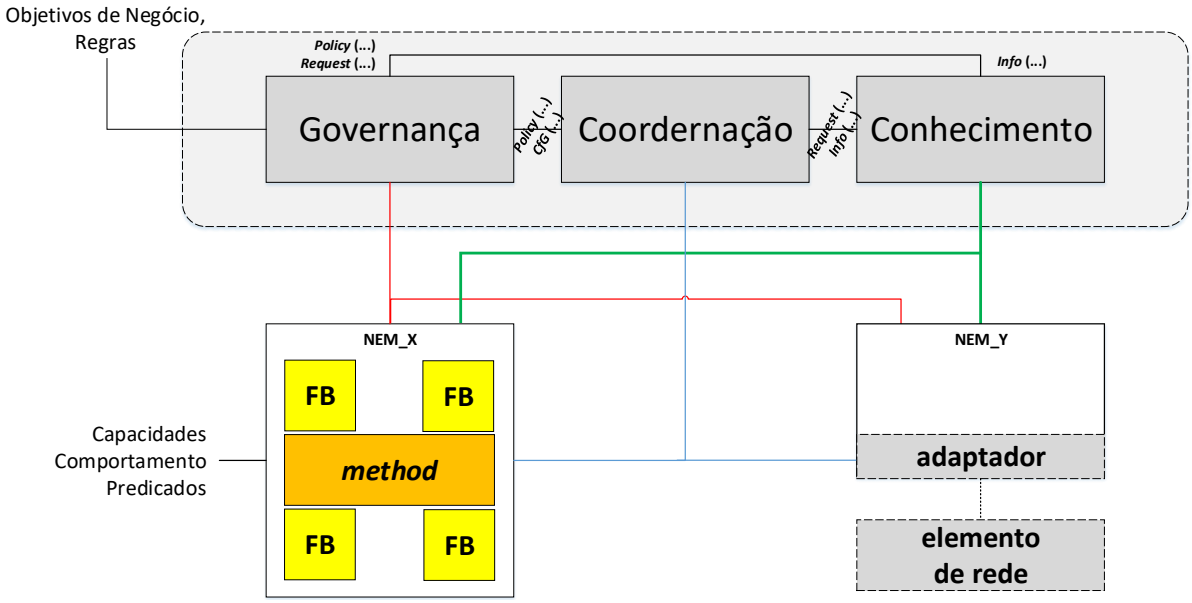
2.4.3.3 UMF

O UMF é um arcabouço na área de ANM. Proposto no projeto UNIVERSELF, é uma tentativa de atacar problemas de heterogeneidade em redes, principalmente aqueles que demandam por auto-gerenciamento. Por definição, o UMF ajuda na unificação, governança e soluções de rede autonômicas *plug and play* dentro de ecossistemas de gerenciamento atuais ou futuros (TSAGKARIS et al., 2012).

A inspiração para o UMF surgiu do conceito de *Self-Organizing Networks* (SONs) (KOUTSOURIS et al., 2013) e foi refletida na elaboração de uma arquitetura que suporte funções *self-**. Vale ressaltar que o conceito de SONs está próximo de ANM, porém ele foca em redes de rádio móveis. A arquitetura do UMF, ilustrada na Figura 15, foi desenhada para gerenciar múltiplos e diferentes ciclos autonômicos de uma maneira unificada. Além disso, o design da arquitetura considera um conjunto de algoritmos, ou métodos, como soluções autonômicas para resolver problemas de telecomunicações. Estes algoritmos são chamados de *Network Empowerment Mechanisms* (NEMs). NEMs são desenhados e implantados com um propósito específico: resolver um problema operacional, atingir um objetivo de desempenho ou realizar um serviço de rede. A composição dos NEMs é realizada através de *Funcional Blocks* (FBs) os quais podem conter funções de monitoramento, configuração, análise, e outras ações que podem ser vistas como instâncias do ciclo MAPE-K. Desta forma, os NEMs são definidos como componentes estratégicos do gerenciamento de rede autonômico na arquitetura UMF.

Além dos NEMs, podemos ver três outros principais blocos na arquitetura: a) gover-

Figura 15 – Ilustração da arquitetura do UMF.



Fonte: Adaptado de Tsagkaris et al. (2013).

nança, que objetiva fornecer a um operador um mecanismo baseado em políticas para controlar a rede a partir de um ponto de vista de alto nível; b) coordenação, responsável por garantir uma interoperação apropriada dos NEMs; e c) conhecimento, que é responsável por prover um gerenciamento unificado da informação em um sistema UMF.

2.4.3.4 Discussão

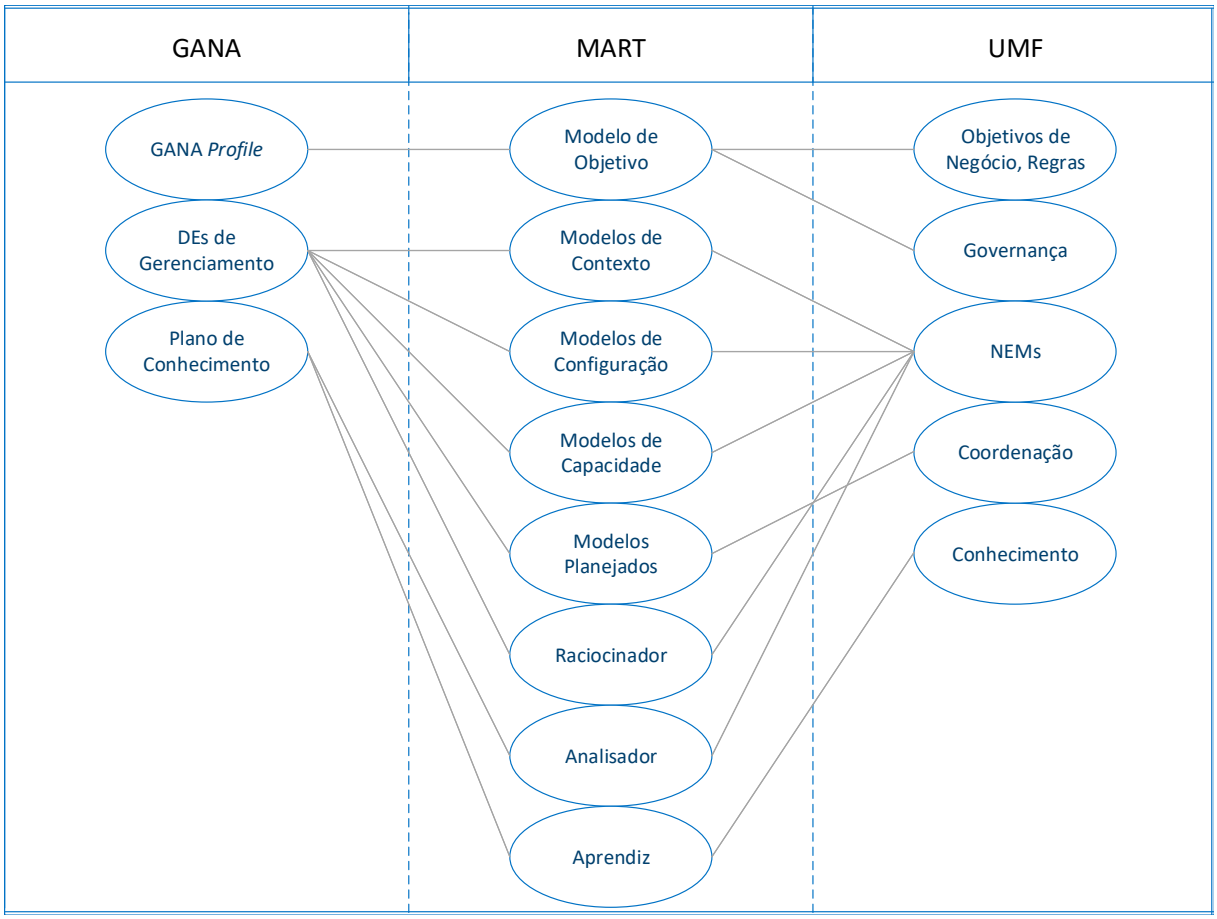
Apesar de diferentes nomenclaturas e posicionamentos, os exemplos de ANMs compartilham diversas semelhanças entre si, tais como: realização de conceitos do MAPE-K, separação de blocos funcionais para base de conhecimento e elementos com funções autonômicas. Ou seja, avaliar os pontos positivos e negativos de cada arquitetura depende, invariavelmente, da implementação e do objetivo desejado pelos operadores de rede.

Nesta tese, partindo das similaridades presentes nas arquitetura discutidas até aqui – principalmente entre as arquiteturas GANA e UMF) – e ilustradas na Figura 16, faz-se uso de alguns destes conceitos, tais como: definição de objetivos, descrição de estado do sistema a ser gerenciado, elementos de gerenciamento e plano de conhecimento. Estes conceitos já foram, em parte, validados em outros trabalhos (como será discutido na Seção 2.5). Assim, serão associados à uma arquitetura MART (cf. Subseção 2.2.1). Essa possibilidade de associação, vista na Figura 16, mostra a ligação entre conceitos de arquiteturas ANM (i.e., GANA, UMF) e uma arquitetura MART. Um exemplo dessa associação está no **plano de conhecimento** da arquitetura GANA, que tem descrição similar ao **bloco de conhecimento** da arquitetura UMF, e ambos podem ser associados

a entidade **aprendiz** da arquitetura MART. Além disso, especificamente da GANA, o componente MBTS (serviço de tradução baseado em modelos) pode perfeitamente ser representado por componentes de modelagem de uma arquitetura MART.

Outro aspecto similar é que tanto GANA quanto UMF definem blocos ou interfaces para a definição de objetivos de rede. Esse aspecto de definição de objetivos também está presente na especificação de MART através de seus modelos de objetivos ou adaptação.

Figura 16 – Relacionamento entre conceitos de arquiteturas ANM e MART.



Ou seja, a partir da identificação dos componentes destas arquiteturas de referência, percebe-se que é razoável realizar a especificação e implementação de uma arquitetura ANM através de modelos em tempo de execução. Os desafios, neste sentido, referem-se à como estes blocos devem ser criados, de forma que pontos positivos de qualidade e performance sejam alcançados (e.g., melhores algoritmos, utilização correta de recursos, escalabilidade, nível de abstração e autonomicidade), sendo necessária uma avaliação que forneça essas informações.

Esta avaliação, tanto qualitativa quanto quantitativa, da arquitetura ANM e seus blocos é discutida por Samaan e Karmouch (2009) e Movahedi et al. (2012), que apresentam as primeiras direções que podem ser utilizadas para realizar a avaliação de uma arquitetura ANM, e foram adotadas no decorrer do desenvolvimento deste. Por exemplo, é

possível avaliar os graus de autonomia, adaptabilidade e inteligência de uma solução (SAMAAN; KARMOUCH, 2009), ou, ainda, avaliar métricas de funcionalidade e desempenho como escalabilidade, largura de banda e gerenciamento de heterogeneidade (MOVAHEDI et al., 2012).

2.5 ESTADO DA ARTE

De forma correspondente aos conceitos descritos na fundamentação teórica, a proposta deste trabalho tem relação com diversas abordagens que compõem o estado da arte. Nesta seção, são descritos trabalhos relacionados às duas principais linhas de pesquisa com as quais este trabalho se relaciona: (1) modelagem e desenvolvimento de aplicações SDN; e (2) gerenciamento autonômico de SDN. Ainda assim, ressaltamos que a solução trazida neste trabalho é a primeira que aplica os conceitos de Modelos em Tempo de Execução para gerenciar redes SDN de forma autonômica.

2.5.1 Modelagem e Desenvolvimento de Aplicações SDN

Uma das primeiras linguagens de programação propostas para SDN foi a *Flow Management Language* (FML) (HINRICHS et al., 2009), que utilizou um paradigma declarativo para expressar políticas e objetivos de rede. Além dela, diversas outras foram propostas, como a Nettle (VOELLMY; HUDAK, 2011) que foi uma das primeiras a aplicar conceitos de DSL para desenvolver aplicações SDN, juntamente com o paradigma FRP. Após isso, as preocupações com consistência de aplicações começaram a surgir, e trabalhos como os de Voellmy, Kim e Feamster (2012) e Monsanto et al. (2012) ofereceram as primeiras soluções para contornar problemas de inconsistência entre políticas. E, finalmente, linguagens focadas na validação de aplicações, como Merlin (SOULÉ et al., 2014) e Kinetic (KIM et al., 2015c) surgiram no cenário de linguagens SDN.

Entretanto, devido aos paradigmas que seguem, estas linguagens compartilham entre si, na maioria das vezes, uma alta dependência dos controladores para os quais foram projetadas, além da falta de mecanismos que auxiliem na validação das aplicações criadas e no gerenciamento das mesmas.

Da perspectiva de modelagem, complementarmente ao que foi discutido na seção 2.3, as ferramentas da literatura mais próximas deste trabalho (focando especificamente na modelagem) são: Miniedit[4], *Visual Network Descriptor* (VND) (FONTES et al., 2014a) e o Common Information Model for SDN (CIM-SDN) (PINHEIRO et al., 2013). O Miniedit refere-se a criação de topologias virtuais para simulações SDN no Mininet através de uma interface gráfica. Outro editor gráfico é o VND, que permite descrever a estrutura

[4] Miniedit - https://github.com/mininet/

e parte do comportamento de uma rede SDN através de modelos gráficos utilizando o *framework* Network Scenario Description Language (NSDL) para definir seus elementos. E, por último, mas não menos importante, o CIM-SDN é um projeto que oferece um modelo de abstração e um editor cliente para gerenciamento de SDN a partir da modelagem de elementos de rede (e.g., hosts, controladores, switches). De forma similar à um dos componentes do MOSES, o CIM-SDN também utiliza Object Constraint Language (OCL) para realizar validações e encontrar inconsistências na construção da rede.

2.5.2 Gerenciamento Autonômico em SDN

Em contraste às arquiteturas de rede tradicionais, funções de controle locais são movidas dos elementos de rede para controladores remotos em SDN. Como resultado, isso pode levar à criação de novos gargalos e potenciais sobrecargas, dependendo do tipo de aplicações de gerenciamento a considerar (YEGANEH; TOOTOONCHIAN; GANJALI, 2013a). Enquanto utilizar um controlador centralizado com uma ampla visão da rede tem o benefício de facilitar a implementação da lógica de controle, isso também apresenta limitações em termos de escalabilidade. Então, diferentes abordagens surgiram para superar os possíveis problemas de implementação e gerenciamento de um modelo único de controlador centralizado. Nesta seção, veremos tanto propostas que se concentraram apenas na arquitetura do gerenciamento (e.g., centralizada ou distribuída) quanto aquelas que adicionaram o viés da autonomicidade na sua realização.

Dentre as soluções hierárquicas e centralizadas, temos, por exemplo, a abordagem de Yeganeh, Tootoonchian e Ganjali (2013a) que baseiam sua proposta em dois níveis de controladores: nível distribuído e nível centralizado. Nesta proposta, os controladores distribuídos operam com informações locais, enquanto o controlador que compõe o nível centralizado possui amplo conhecimento da rede. Outra solução hierárquica é apresentada por Ahmed e Boutaba (2014), porém foca em *Wide Area Networks* (WAN), dividindo a rede em múltiplas zonas de controle, com uma camada de gerenciamento centralizada acima destas zonas.

Na contramão de soluções hierárquicas, abordagens totalmente distribuídas foram propostas por Koponen et al. (2010) e Tootoonchian e Ganjali (2010). A diferença entre elas está na implementação, enquanto Koponen et al. (2010) utilizou um plano de controle distribuído abstraindo os recursos de rede como objetos de dados armazenados em bases de informação de rede, em Tootoonchian e Ganjali (2010) os autores distribuíram fisicamente um plano de controle logicamente centralizado, o qual necessita de mecanismos de sincronização.

É interessante notar que as abordagens discutidas até aqui focaram em oferecer um gerenciamento distribuído ou centralizado apenas criando interfaces entre o plano de dados e o plano de controle. Isso mudou com o trabalho de Kim e Feamster (2013), o qual focou

na interface entre a plataforma de controle e a lógica de gerenciamento. Outra questão é que nenhuma destas propostas discutiu um gerenciamento autonômico da rede.

Este foco em autonomicidade começa por propostas de arquiteturas (algumas já discutidas na subseção 2.4.3) e abordagens que investigam partes do gerenciamento (e.g., utilização de recursos, QoS). Por exemplo, em relação às arquiteturas, Li et al. (2013) propõem a *Autonomic Management Architecture* (AMA), uma arquitetura focada especificamente em SDN e no oferecimento de múltiplos serviços de rede. Em Koutsouris et al. (2013), os autores utilizam uma das arquiteturas discutidas anteriormente (cf. Seção 2.4) implementando a UMF para alcançar propriedades autonômicas no gerenciamento de redes. Gelenbe (2013), por outro lado, propõe uma outra arquitetura, dentro do paradigma SDN, que é ciente de suas capacidades e que adiciona pacotes inteligentes para alcançar autonomicidade na rede. Em uma perspectiva diferente, Qi et al. (2014) propõe outra arquitetura autonômica, mas focada em redes SDN virtualizadas.

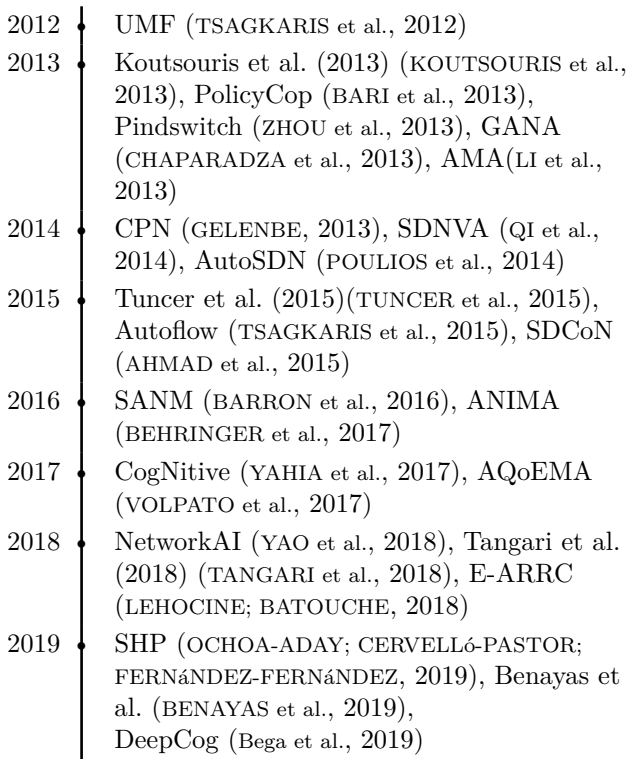
Mais recentemente, Yahia et al. (2017) propuseram uma arquitetura chamada CogNitive para prover gerenciamento autonômico em redes 5G definidas por software, utilizando o ciclo MAPE-K e técnicas de aprendizagem de máquina. Por outro lado, focando em QoE, Volpato et al. (2017) propõem mais uma arquitetura ANM (a qual demos o acrônimo de AQoEMA - *Autonomic QoE Management Architecture*) para fornecer serviços cientes dos níveis de QoE.

Outros autores optam por realizar partes do gerenciamento autonômico a partir da implementação de algoritmos (TUNCER et al., 2015), protocolos (ZHOU et al., 2013), controladores (POULIOS et al., 2014) e frameworks (BARRON et al., 2016; AHMAD et al., 2015; TSAGKARIS et al., 2015). Uma linha do tempo demonstrando o surgimento destes trabalhos (e das outras arquiteturas já discutidas até aqui) está ilustrada na figura 17.

Especificamente estas propostas podem ser resumidas da seguinte forma. Bari et al. (2013) discutem a realização de requisitos de QoS realizado de forma autonômica, enquanto Zhou et al. (2013) propõem um novo protocolo SDN, o PindSwitch, para alcançar autonomicidade. Neste trabalho, os autores estendem a arquitetura GANA (cf. subseção 2.4.3.1) com um protocolo SDN independente visando prover o gerenciamento autonômico em redes complexas. Outro aspecto identificado é o problema do posicionamento de controladores, que começa a ser discutido em Heller, Sherwood e McKeown (2012). Deste problema, Tuncer et al. (2015) propõem algoritmos adaptativos para posicionar controladores, integrando-os ao gerenciamento de SDN. Poulios et al. (2014) por outro lado, propõem que o próprio controlador seja autonômico, implementando o AutoSDN, um controlador para permitir características *self-** em redes SDN.

Por fim, temos propostas que envolvem a definição e implementação de frameworks para prover ANM. Por exemplo, em Barron et al. (2016), os autores propõem um framework de gerenciamento auto-adaptativo, visando alcançar níveis de QoS ótimos para serviços rede. No mesmo caminho, Ahmad et al. (2015) propõem o SDCoN, um framework

Figura 17 – Linha do tempo de pesquisas em ANM para SDN.



Fonte: O autor (2020).

que visa permitir ações cognitivas na rede. Tsagkaris et al. (2015) também lançaram seu próprio framework, o qual é composto por uma versão autonômica do protocolo OpenFlow, chamada AUTOFLOW, e que combina SDN com ANM.

Desta forma, é possível perceber que o movimento em direção à gerência autonômica começou e foi aumentado pelo paradigma SDN aliado à diferentes ideias, embora ainda não haja um consenso sobre as melhores práticas ou padrões a serem adotados para a sua realização. Neste contexto, a presente tese permite a implementação de uma solução de gerenciamento autonômico para SDN, combinando os conceitos básicos discutidos até aqui, os quais viabilizam esta solução. Em comparação aos trabalhos relacionados, esta tese complementa o estado da arte no gerenciamento autonômico de redes SDN, possibilitando uma forma integrada de definição de objetivos e auto-adaptação ainda não explorada pelos trabalhos anteriores. Além disso, contribuições pontuais no aumento do nível de abstração para o desenvolvimento de um sistema ANM para SDN, na compatibilidade em redes SDN heterogêneas e na investigação da utilização de aprendizagem por reforço profunda também colocam a solução aqui proposta com diferencial importante na literatura.

3 MODELOS DE ALTO NÍVEL PARA DESENVOLVER E VALIDAR APLICAÇÕES EM REDES DEFINIDAS POR SOFTWARE

Ao analisar as mais populares linguagens de programação para SDN, nota-se que menos da metade destas linguagens fornecem meios de verificações dinâmica ou estática das aplicações criadas. Adicionalmente, as linguagens baseadas no paradigma DSL (e.g., Pyretic (REICH et al., 2013b) e Flog (NELSON et al., 2014)) ainda deixam espaço para melhoramentos, tais como: liberar a dependência da linguagem em relação a um controlador, permitir a verificação do software e aumentar a compatibilidade entre aplicações e infraestrutura.

Uma das principais funcionalidades de SDN é permitir que gerenciadores de rede ofertem aos desenvolvedores de software uma *Application Programming Interface* (API). A *Northbound* API, por exemplo – presente na arquitetura SDN (JARSCHEL et al., 2014), permite aos desenvolvedores a criação de aplicação para uma variedade de tarefas de gerenciamento de rede baseadas em políticas. Uma prática comum no escopo de criar linguagens para *Northbound* API é o acoplamento com um determinado controlador SDN subjacente (e.g., POX (MCCAULEY et al., 2015), Beacon(ERICKSON, 2013b), OpenDaylight (MEDVED et al., 2014)). Por exemplo, a linguagem Pyretic (REICH et al., 2013b) foi criada para executar em um controlador POX. Desta forma, uma aplicação desenvolvida em Pyretic não irá funcionar em outro controlador, como, por exemplo, o Beacon.

Nesse sentido, apesar de o paradigma SDN oferecer algum nível de abstração para programação, ainda falta um caminho conciso e de alto nível para expressar políticas de rede e suportar o uso de aplicações desenvolvidas de forma agnóstica em relação ao controlador. Há evidências em (LOPES et al., 2016) de que não há um caminho claro em direção à criação de aplicações SDN em alto nível de abstração. Assim, nesta seção, complementa-se a resposta da Questão de Pesquisa 1 (cf. Seção 1.2) sobre *quais estratégias são mais indicadas para desenvolver aplicações SDN, de forma que requisitos como um alto nível de abstração, a validação e a independência de controladores sejam atendidos?*

Para isso, esta etapa do presente trabalho estende um estudo anterior (LOPES et al., 2015), indo além na análise do conceito de MDN que defende o desenvolvimento de aplicações SDN utilizando modelos de alto nível, i.e., uma abordagem MDD, e é instanciado a partir de um *framework* composto de uma ferramenta CASE e de uma DSML. Nesse contexto, este capítulo traz os seguintes resultados: (i) observa-se que o *framework* MDN pode superar a maioria das questões em aberto no desenvolvimento de aplicações SDN – utilizando níveis altos de abstração e realizando a separação entre linguagens e controladores; e (ii) fica demonstrado que a geração de código do MDN é até duas vezes mais rápida que outras abordagens, possibilitando seu uso em cenários reais.

O restante desse capítulo é organizado da seguinte forma: a Seção 3.1 apresenta os requisitos para o desenvolvimento de aplicações SDN. A Seção 3.2 apresenta a metodologia e infraestrutura do MDN utilizada neste trabalho. A avaliação e discussão dos resultados

estão presentes na Seção 3.3. As considerações finais deste capítulo são apresentadas na Seção 3.4.

3.1 DESENVOLVIMENTO DE APLICAÇÕES PARA REDES DEFINIDAS POR SOFTWARE

Considerando que o foco desta capítulo é no desenvolvimento de aplicações SDN, é preciso definir que este desenvolvimento pode ser visto como o processo de escrita de código em uma linguagem de programação que interage com a *Northbound* API. Porém, essa visão simplística esconde a complexidade que programadores possuem ao lidar com a *Northbound* API manualmente. Como uma aplicação SDN envolve diversas estruturas interdependentes (e.g., regras condicionais, controle de acesso), o seus desenvolvimento é, essencialmente, propenso a erros – já que é responsabilidade do programador garantir que as regras especificadas em nível de aplicação sejam executadas de acordo no plano de encaminhamento. É comum, neste caso, para programadores ter que lidar com uma variedade de detalhes de baixo nível da *Southbound* API e do plano de encaminhamento. Por exemplo, eles têm que lidar com acoplamento de módulos, detalhes de regras de álgebra dos switches (e.g., padrões de máscaras de bits) e diversos tipos de condição de corrida impostos pelo aspecto distribuído de qualquer rede de switches (FOSTER et al., 2011b).

3.1.1 Desafios no Desenvolvimento de Aplicações em Redes Definidas por Software

As dificuldades em implementar aplicações para SDN de forma manual geram diversos desafios. De acordo com Foster et al. (2011b), Lopes et al. (2016) e Kim et al. (2015a), é possível destacar os desafios em quatro cenários:

- Elementos de rede desempenham tarefas múltiplas e paralelas, tais como: controle de acesso, roteamento e monitoramento de tráfego. Criar uma aplicação independente de elementos de rede para estas tarefas é difícil, dada a relação que pode existir entre as regras envolvidas em cada uma delas.

- A comunicação entre os planos de dados e de controle tem um nível de abstração baixo especificado por um protocolo SDN (e.g., OpenFlow (ONF, 2013), POF (SONG, 2013b)). Este baixo nível de abstração, essencialmente, compõe as interfaces de controladores SDN (i.e., *Southbound* e *Northbound*), tornando complexa a programação para tais controladores. Ainda que as *Northbound Interfaces* devam ser implementadas para abstrair detalhes de baixo nível, isso ainda não é uma realidade.

- Aplicações SDN desenvolvidas para um certo fornecedor de controlador não irão funcionar em um ambiente diferente que possua outro fornecedor. Essa questão re-

sulta em uma indesejada dependência de controladores para implantar um ambiente SDN. Por exemplo, um controlador baseado na linguagem de programação Java tem suas aplicações e módulos escritos em uma linguagem de programação específica, geralmente a mesma utilizada para escrever o controlador. Tal dependência se torna problemática quando o tipo do controlador (ou fornecedor) precisa ser alterado, já que, é importante ressaltar, o desempenho de controladores pode ser altamente distinto (YEGANEH; TOOTOONCHIAN; GANJALI, 2013b; ALENCAR et al., 2014).

- Gerentes de rede podem não estar familiarizados com detalhes de implementação de baixo nível utilizados em linguagens de programação SDN. A utilização de modelos pode, assim, melhorar a comunicação entre gerentes e desenvolvedores (MOHAGHEGHI et al., 2011).

As ferramentas baseadas em DSML, as quais servem tanto como itens de documentação quanto como objetos executáveis, possuem o potencial de aumentar o nível de abstração para o desenvolvimento de aplicações SDN.

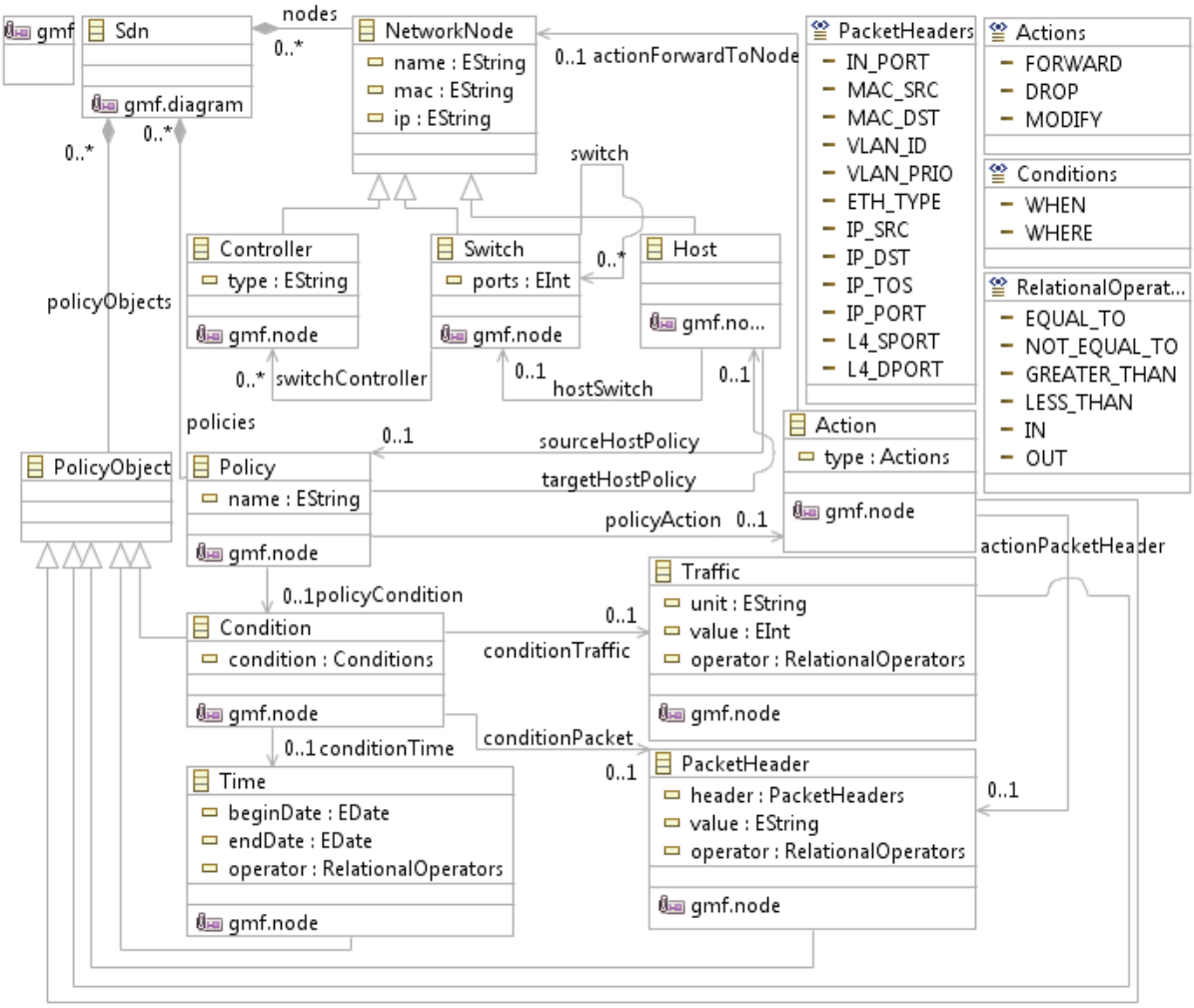
3.2 METODOLOGIA

3.2.1 Definição das Sintaxes Abstrata e Concreta

Para realizar o desenvolvimento e a validação de aplicações utilizando modelos de alto nível foi utilizado o trabalho sobre MDN publicado em (LOPES et al., 2015), expandido-o para compreender a validação de modelos. O *framework* MDN fornece uma DSML subjacente para permitir a modelagem de aplicações. Assim, em sua infraestrutura, as duas sintaxes que compõem toda DSML foram utilizadas: a sintaxe abstrata (i.e., metamodelo) e a sintaxe concreta (i.e., representação gráfica) (SCHMIDT, 2006a). Para implementar estas sintaxes, a especificação de protocolo OpenFlow (ONF, 2013) foi adotada. Tal especificação estabelece, por exemplo, que um controlador pode ser ligado com vários switches, bem como um switch pode ser conectado com vários controladores – um exemplo direto de relacionamento muitos-para-muitos.

Outro exemplo de relacionamento presente no metamodelo MDN (cf. Figura 18) é a associação entre políticas de rede e os respectivos fluxos. Esta modelagem é possibilitada pelo elemento *Policy* adicionado no metamodelo do MDN, relacionando-o com os elementos *Controller* e *NetworkNode*. Desta forma, o elemento *Policy* pode definir o comportamento de um determinado fluxo. Entretanto, vale ressaltar que políticas de rede em SDN são algoritmos que podem estar relacionados a vários elementos de rede (e.g., tráfego, switches, hosts). Assim, estes elementos também fazem parte do metamodelo, e.g., *Traffic*, *Flow*, *Policy*, *Concession*, *RelationalOperator*, *Action*, *Time* e *Condition*. Elemen-

Figura 18 – Metamodelo da abordagem MDN.



Fonte: O autor (2020).

tos da arquitetura SDN, i.e., *Controller*, *Switch* e *Host* são nós do elemento *Sdn* – que representa, basicamente o diagrama para modelar tais elementos.

Com relação à sintaxe concreta do MDN, para cada elemento do metamodelo existem formas gráficas para representá-los. Isso inclui os tipos de ligações existentes no modelo de uma aplicação SDN (e.g., linhas sólidas para links físicos e linhas tracejadas para associação de regras) visando diferenciar as conexões entre elementos de rede e regras. O metamodelo, presente na Figura 18, também permite a modelagem de cabeçalhos de pacotes, tráfego e operadores relacionais para permitir aos desenvolvedores uma maior granularidade na especificação de regras para suas aplicações. Este metamodelo surgiu a partir da análise da especificação do protocolo OpenFlow (ONF, 2013), considerando as características de um switch baseado neste protocolo bem como os cabeçalhos de pacotes que podem compor regras nas tabelas de fluxo.

Quadro 3 – Restrição EVL para validar o nome de *NetworkNode*. Neste caso, a *Regra #3* valida que o elemento *NetworkNode* possua um MAC.

```
1   context NetworkNode {
      constraint hasMAC { // Regra #3
3       check : self.mac.isDefined();
        message : 'Undefined ' + self.eClass().mac + 'not allowed';
5       fix {
          title : 'Define the node MAC';
7         do {
            var type := UserInput.prompt('What is the MAC?');
9           if (type.isDefined()) self.type := type;
          }
11      }
      }
13  }
```

Fonte: O autor (2020).

3.2.2 Regras de Restrição

Visando a validação das aplicações modeladas no MDN Editor, a definição do domínio semântico da DSML em questão foi baseada na utilização de linguagem natural para representar regras de restrição bem formuladas, escritas na linguagem *Eclipse Validation Language* (EVL) (KOLOVOS; ROSE; PAIGE, 2013). Estas regras são avaliadas sempre que um usuário (e.g., desenvolvedor, gerente de rede) tenta validar um modelo de aplicação. A necessidade para utilizar uma linguagem como a EVL é garantir as restrições de modelagem que um metamodelo não pode representar. Por exemplo, de acordo com a especificação do protocolo OpenFlow, existem regras de restrições que não foram possíveis realizar através apenas de metamodelagem, tais como: elementos do tipo *NetworkNode* devem possuir um nome e um endereço IP válido, elementos *Host* não podem possuir o mesmo endereço MAC.

Nos scripts EVL neste trabalho, cada contexto é associado à uma instância de classe (e.g., *NetworkNode*, *Host*, *Policy*). Dentro do contexto especificado, existe o conceito de invariância, i.e., restrição e crítica, o qual define a aplicabilidade de regras para um subconjunto de instâncias especificadas pelo contexto. Essa definição foi realizada pela escrita de expressões de **guarda** e **verificação**. Se alguma instância se enquadra na guarda e não satisfaz a expressão de verificação o modelo não é válido e um ajuste precisa ser realizado.

O exemplo presente no Quadro 3 mostra a definição de uma regra de restrição utilizando linguagem natural e EVL. Regras semelhantes, baseadas em EVL, foram criadas e utilizadas para definir o domínio semântico do metamodelo estendido originalmente do MDN.

3.2.2.1 Geração de Código

A geração de código do MDN Editor torna a construção automática e declaração de diversas variáveis, estruturas e métodos sem a necessidade de direto envolvimento do usuário final (e.g., gerente de rede). O Quadro 4 exemplifica a utilização do que o processo definido no GMF chama de *templates* de código, ilustrando um dos templates, i.e., *sdn.egl*, utilizado para gerar código da aplicação modelada. Este *template* é composto de uma combinação de tags da linguagem EGL (e.g., operações, comandos condicionais) e textos estáticos, i.e., texto fora dos delimitadores [% %]. O resultado da execução destes scripts é a substituição das tags EGL pela informação presente no diagrama SDN criado pelo desenvolvedor, gerando os scripts que são executados pelo controlador SDN.

O *template sdn.egl* (ilustrado no Quadro 4) descreva a integração entre *templates*, onde cada um deles responsável por realizar ações específicas no código final da aplicação. Por exemplo, um dos *templates* realiza a importação das possíveis bibliotecas de um controlador alvo a serem utilizadas pela aplicação modelada. Além disso, supondo que o controlador POX (MCCAULEY et al., 2015) seja o alvo da aplicação, ele utiliza o método `mdn_handler(event)` na linha 3 do Quadro 4. Em resumo, este método avalia as regras condicionais presentes na aplicação modelada, identificando as ações requeridas. É a partir das ações de cada regra (cf. Quadro 4 - linha 5) que o corpo do código é gerado, recuperando informações nos elementos *NetworkElements* e *Conditions*. A verificação de contadores (cf. Quadro 4 - linha 10) é utilizada para evitar duplicidade de código ao implementar tais regras. Por exemplo, o método `addListenerByName` na linha 11 (cf. Quadro 4) não pode ser definido mais de uma vez com os mesmos parâmetros, `PacketIn` e `mdn_handler`, ou iria causar uma falha em tempo de execução como consequência da duplicidade de assinatura do método.

Vale mencionar que os *templates* utilizados para gerar código devem ser definidos não apenas de acordo com as características das aplicações modeladas, mas também com as es-

Quadro 4 – Trecho do *template sdn.egl*. Este *template* define as bibliotecas e métodos utilizados na geração de modelos MDN.

```
1   [% import "header.egl"; [...] %] // Importa de bibliotecas
    [% var counterActionDrop : Integer = 0; %]
3   def mdn_handler(event):
    [%
5       for (rule in Rule.all) {
            core.openFlow.addFlowRule(rule);
7       }
    %]
9   def launch():
    [% if (counterActionDrop > 0) { %
11      core.openflow.addListenerByName("PacketIn", mdn_handler);
    [% } %]
```

Fonte: O autor (2020).

pecificações dos controladores subjacentes. Por exemplo, apesar de os exemplos de geração de código serem apresentados para o controlador POX (baseado em Python), é possível utilizar outros *templates* para gerar código compatível com diferentes controladores (e.g., OpenDaylight).

Esta característica extensível permite a compatibilidade de modelos MDN com diversos controladores SDN sem a necessidade de realizar qualquer tipo de modificação nas sintaxes e mapeamentos da DSML subjacente. Assim, a aplicação criada seguindo a abordagem MDN pode ser migrada para diferentes cenários SDN (e.g., para testar o desempenho de aplicações em controladores distintos) sem a necessidade de refatoração de código. Existe apenas a necessidade de gerar código para o controlador alvo.

3.3 AVALIAÇÃO E DISCUSSÃO

Visando avaliar quantitativamente a utilização da abordagem MDN para modelar e validar aplicações SDN, verificou-se o desempenho do *framework* proposto ao gerar código de aplicações e validar os modelos. Os experimentos desta subseção foram realizados em um hardware equipado com um processador Intel Core i7-5500U 2.40Ghz e 8GB de memória.

3.3.1 Sobrecarga de Restrições Semânticas

O primeiro experimento visou mensurar a sobrecarga que diversas restrições semânticas teriam na validação de aplicações e topologias complexas. A ideia é verificar a **Hipótese 1** de que a utilização de modelos permite o desenvolvimento e gerenciamento de redes SDN. Para este experimento, uma aplicação de controle de acesso foi modelada e precisou ser validada para três topologias reais, escolhidas arbitrariamente do Topology Zoo [1] (cf. Tabela 2). O experimento mediu o **tempo de validação**, utilizando como fatores de desempenho a **quantidade de restrições semânticas** e a **complexidade da topologia** (i.e., quantidade de nós e links).

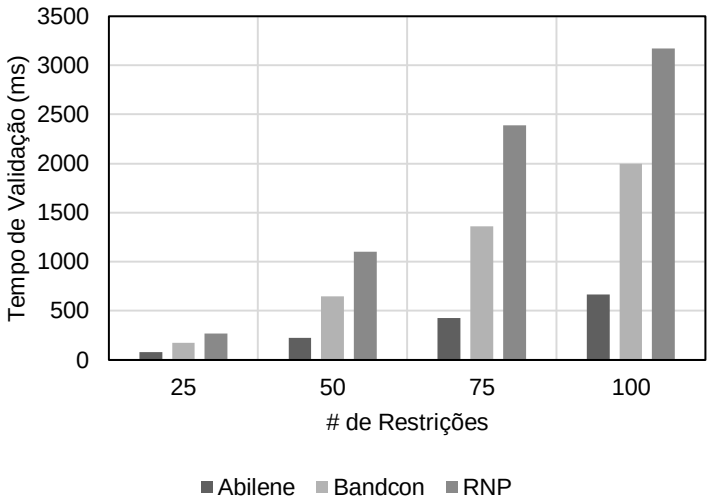
[1] http://www.topology-zoo.org/dataset.html

Tabela 2 – Descrição das topologias analisadas

Nome da Topologia	# de Nós	# de Links
Abilene	11	14
Bandcon	21	28
RNP	27	30

Fonte: O autor (2020).

Figura 19 – Tempo médio de validação para um número crescente de restrições e diferentes topologias de rede.



Fonte: O autor (2020).

Para executar esse teste, o número de restrições semânticas nos níveis 25, 50, 75 e 100 foi definido de forma arbitrária, visando verificar como estes aumentos no número de restrições impacta no desempenho da validação – considerando que a aplicação de controle de acesso poderia ter, então, até 100 regras semânticas a serem verificadas – para cada topologia, medindo o **tempo de validação** para a aplicação. Para obter significância estatística de 95%, o experimento foi repetido 30 vezes para cada conjunto de fatores, obtendo os resultados ilustrados na Figura 19

Os resultados indicam que o tempo de validação é mais afetado pelo número de restrições do que pelo aumento no tamanho da rede. Isso torna-se evidente quando foi observado que cada restrição adicional requer a verificação de diversos elementos do modelo. Por outro lado, a inserção de cada *NetworkNode* requer apenas a verificação das restrições associadas com este novo elemento em particular.

3.3.1.1 Tempo de Validação para uma Aplicação de Balanceamento de Carga

A fim de verificar o comportamento da validação identificado no experimento anterior, um segundo teste de desempenho visou verificar a adequabilidade de utilizar MDN na validação de aplicações a serem utilizadas em redes com alta quantidade de nós que estejam envolvidos em um determinado conjunto de restrições. Para isso, a aplicação de balanceamento de carga foi modeladada como ilustra a Figura 20, utilizando a sintaxe concreta associada às entidades do metamodelo descrito anteriormente.

A aplicação de balanceamento de carga recebe requisições HTTP a partir de um cliente (*host* `h5`), o qual está conectado ao *switch* `s2`. A aplicação deve distribuir as requisições vindas de `h5` para um grupo de *hosts* conectados ao *switch* `h1` (i.e., *hosts* `h1`, `h2`, `h3` e `h4`)

seguindo o algoritmo round-robin. Com o objetivo de realizar este balanceamento, uma ação chamada LOAD_BALANCE foi modelada, a qual é responsável por realizar a tarefa de balancear a carga na aplicação gerada a partir do modelo.

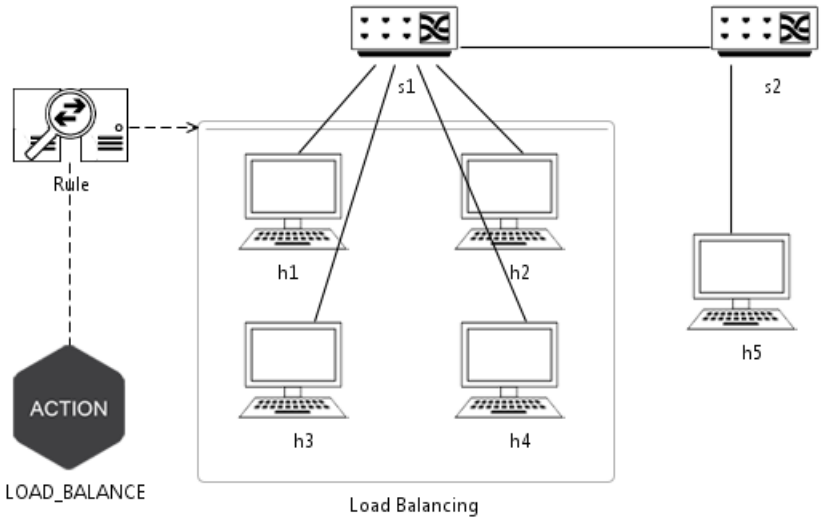
Quadro 5 – Trecho do *template* EGL utilizado na aplicação de balanceamento de carga.

```
[% operation actionLoadBalancing(rule : Any) : String { %]
virtual_ip=IPAddr("[%=rule.targetGroupRule.ip%]")
host = {}
[%  var index = 0;
for (host in rule.targetGroupRule.hostsGroup.all) { %]
   host[[%=index%]]={'ip':IPAddr("[%=host.ip%]")}
   [% index++; %]
[% } %]
total_hosts = len(host)
host_index = 0
[...]
if (not event.parsed.find("ipv4")):
   return EventContinue
if (msg.match.nw_dst != virtual_ip):
   return EventContinue
[...]
index = host_index % total_hosts
selected_host_ip = host[index]['ip']
host_index += 1
[...]
msg.actions.append(of.ofp_action_nw_addr(of.OFPAT_SET_NW_DST,
    selected_host_ip))
event.connection.send(msg)
[...]
[%}%]
```

Fonte: O autor (2020).

Quando o desenvolvedor define a ação de LOAD_BALANCE, um dos *templates* que compõem a infraestrutura do MDN captura os *hosts* agrupados, os quais são usados na geração

Figura 20 – Modelo para a aplicação de balanceamento de carga.

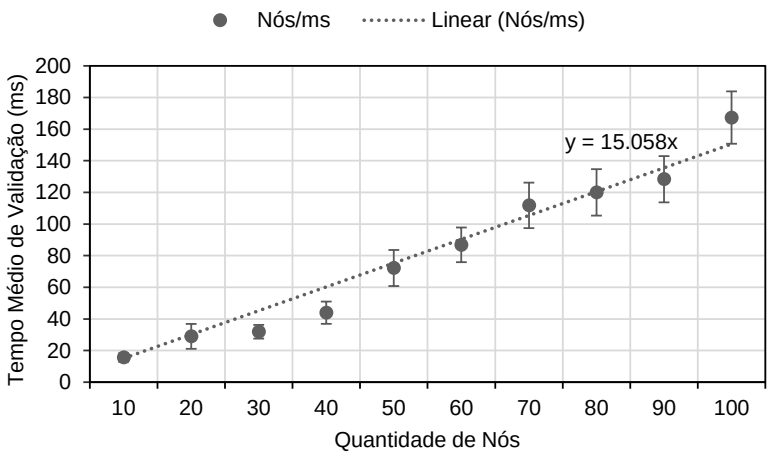


Fonte: O autor (2020).

de código. Se existir tal ação no modelo, a geração de código seleciona os endereços IPs dos *hosts* agrupados e os insere em uma lista. A partir desta lista, o algoritmo round-robin presente no *template* seleciona um *host* seguidamente do outro, redirecionando o tráfego para eles. O Quadro 5 mostra o *template* utilizado nessa geração de código. Na linha 2, a variável `virtual_ip` recebe da entidade `rule` um endereço IP virtual do balanceamento de carga; nas linhas 12 e 14 ocorre o processo em que a aplicação irá balancear apenas o tráfego que tem o IP virtual como destino; a partir da linha 17, o algoritmo round-robin inicia, definindo a variável `selected_host_ip`; e nas linhas 21 e 22, a API do controlador POX é utilizada para definir a rota ao `selected_host_ip`.

Para obter a métrica **tempo de validação** em milissegundos necessários para validar a aplicação de balanceamento de carga, foi realizada uma variação no fator **quantidade de nós** envolvido no balanceamento. Esta variação foi feita incrementando a **quantidade de nós** de 10 em 10, até chegar em uma topologia com 100 elementos. Para obter significância estatística, o experimento foi repetido 30 vezes para cada quantidade de nós, medindo o **tempo de validação** gasto no processo. A Figura 21 ilustra como o tempo de validação varia com a quantidade crescente de nós. Os resultados indicam, com 95% de nível de confiança, a ocorrência de uma variação média de 32% no tempo de validação para cada 10 nós adicionados à topologia subjacente da aplicação. É possível inferir que há uma correlação linear entre o tempo de validação e a quantidade de nós presente no modelo para o intervalo analisado (i.e., entre 10 e 100 nós). A variação observada na linha de tendência pode ser explicada pela mudança no formato da topologia, já que os nós foram adicionados aleatoriamente.

Figura 21 – Média do tempo de validação para um número crescente de nós em uma aplicação de balanceamento de carga.



Fonte: O autor (2020).

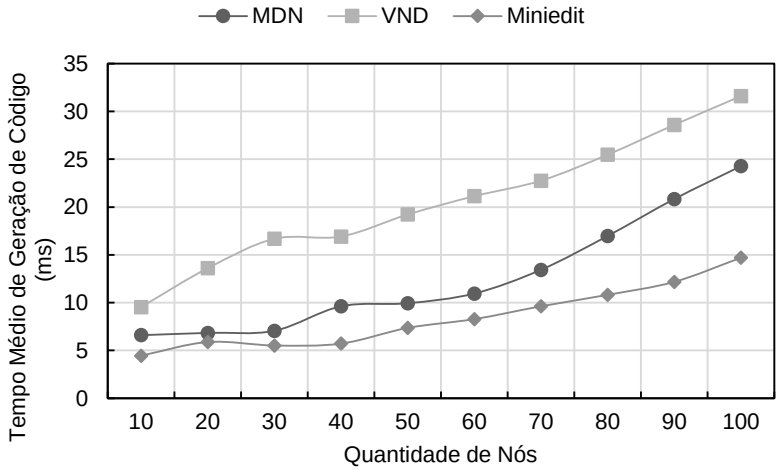
3.3.2 Geração de Código

O último cenário de avaliação teve como objetivo verificar a geração de código do MDN através do editor gráfico de modelagem que o compõe, comparando as métricas de desempenho deste com outras duas ferramentas de modelagem: Miniedit(LANTZ; HELLER; MCKEOWN, 2010) e VND (FONTES et al., 2014b). Porém, como estas ferramentas não permitem a modelagem de aplicações SDN, o teste foi adaptado para gerar código que configura topologias no Mininet. Neste experimento, a métrica utilizada foi o **tempo de geração de código** para criar topologias no Mininet e os fatores considerados foram a **quantidade de nós** e a **ferramenta utilizada**. Mensurar este tempo de geração de código será essencial na utilização de modelos em tempo de execução que serão descritos nos próximos capítulos da tese. O cenário de modelado considera que as topologias criadas em todas as ferramentas consistiram de dois *switches* conectando um número crescente de nós. Para cada fator, foram realizadas 30 execuções adotando o nível de confiança de 95%, medindo o **tempo de geração de código**. A Figura 22 mostra que o MDN com as extensões realizadas alcançou um tempo médio que se equipara ao de outras abordagens, ainda que realize validações semânticas durante a geração de código.

Neste cenário, o leitor deve observar, a partir do resultado obtido (cf. Figura 22), que o Miniedit é mais rápido que o MDN Editor. Porém, vale enfatizar que a geração de código envolvida no MDN Editor possui uma semântica bem definida e verifica diversas restrições antes de gerar o código final. Além disso, modelos criados no MDN Editor não são diretamente acoplados aos comandos do Mininet. Na análise realizada, essa característica causa uma pequena sobrecarga no desempenho do gerador de código, mas garante a independência dos modelos com relação à infraestruturas subjacentes (e.g., controladores, simuladores). O resultado também mostra que o MDN Editor é até duas vezes mais rápido que o VND.

A importância em analisar esta métrica fica evidente ao ser considerado um cenário de MART, onde as adaptações feitas nos modelos não podem demorar na reação aos eventos que possam ocorrer na rede.

Figura 22 – Tempo médio necessário para gerar código em cada ferramenta analisada, considerando um número crescente de nós em uma topologia arbitrária.



Fonte: O autor (2020).

3.4 CONSIDERAÇÕES FINAIS

Este capítulo discutiu como o *framework* MDN permite que um processo de desenvolvimento baseado em modelos seja utilizado para desenvolver e validar aplicações SDN, evitando implementações propensas a erros e fornecendo modelos executáveis consistentes, além de aplicações agnósticas em relação à infraestrutura SDN subjacente.

As funcionalidades do *framework* foram avaliadas em três diferentes cenários, verificando o tempo de validação de diferentes aplicações e topologias, além de mensurar o tempo de geração de código a partir dos modelos criados – comparando este tempo com outras duas ferramentas de modelagem da literatura.

As métricas obtidas nos experimentos e na avaliação pertmitem confirmar a **Hipótese 1** desta tese, dando um passo em direção à resposta da **Questão de Pesquisa 3** (cf. Seção 1.2), fornecendo indícios sobre como uma abordagem deste tipo pode permitir a implementação de aplicações e gerenciamento autonômico em SDN.

4 UTILIZAÇÃO DE MODELOS PARA LIDAR COM HETEROGENEIDADE DE DE RECURSOS EM REDES DEFINIDAS POR SOFTWARE

Este capítulo trata da investigação sobre como lidar com a heterogeneidade da infraestrutura SDN para desenvolver e implantar aplicações. Em particular, complementa-se a resposta à **Questão de Pesquisa 3** (cf. Seção 1.2) – mostrando como uma abordagem baseada em modelos pode impactar positivamente o gerenciamento e desenvolvimento de SDN, se aproveitando de todas as capacidades da infraestrutura de rede (e.g., protocolos suportados, *bandwidth*). A solução proposta aqui realiza mais uma extensão ao *framework* MDN discutido no Capítulo 3 e que foi inspirada pela ideia de **delegação de fluxo** (DF) (BAUER; ZITTERBART, 2016): encaminhamento de fluxos para serem processados por *switches* auxiliares, aproveitando recursos de *switches* ao longo da rede para alcançar um determinado objetivo. Essa extensão envolve permitir à modelagem do MDN que o conceito de Delegação de Fluxo (DF) seja parte das aplicações finais geradas.

Outras abordagens tentaram resolver a questão da compatibilidade em redes heterogêneas através da utilização de controladores SDN diferentes para cada grupo de elementos de rede (ARORA et al., 2016; RINALDI et al., 2015), padronizando a descrição de serviços de rede (DARADKEH et al., 2016; SCHONWALDER; BJORKLUND; SHAFER, 2010), ou mesmo pela programação direta do plano de dados (SONG, 2013a). Alternativamente, este capítulo propõe uma solução de delegação de fluxo baseada em modelos para utilizar o conjunto de capacidades da infraestrutura de rede, lidando, assim, com a heterogeneidade na perspectiva de capacidades dos dispositivos de rede.

Em resumo, este capítulo foca no melhoramento da utilização de recursos e compatibilidade entre aplicações e infraestrutura de rede. Este melhoramento é realizado a partir da adição de dois componentes novos ao *framework* MDN e sua camada de Geração de Código: o componente de avaliação de compatibilidade e o componente de delegação de fluxo em si. Estes componentes são responsáveis por: (i) redirecionar os fluxos de um *switch* – incapaz de atender a algum requisito de aplicação – para um outro *switch* que seja capaz de atender tal requisito; e (ii) definir caminhos alternativos para atender requisitos de desempenho (e.g., largura de banda). Em outras palavras, considere, por exemplo, uma aplicação de monitoramento de rede que utiliza o protocolo P (e.g., IPFIX, sFlow) para monitorar os fluxos passando através de um *switch* S. Se S não suporta P, esta aplicação não irá ser executada apropriadamente. Um cenário similar é descrito em Song (2013a) mas relacionado à compatibilidade entre versões do protocolo OpenFlow. A ideia aqui é replicar os fluxos que passam através de S para um *switch* $S´$ que suporte P.

Com base nos experimentos realizados e que serão descritos neste capítulo, a abordagem baseada em modelos demonstra ser viável para: (i) executar aplicações que não seriam diretamente suportadas por algum dispositivo de rede (e.g., diferentes tipos de protocolos); e (ii) executar aplicações que podem apenas operar sob certas condições de

desempenho (e.g., atraso máximo, largura de banda mínima). Os experimentos mostram que o mecanismo de DF melhora a compatibilidade entre aplicações e *switches* SDN a um custo de ∼15% no aumento do atraso. Por outro lado, considerando cenários de QoS, o mecanismo proposto alcançou menor perda de pacotes e melhores taxas de transferência quando comparado com abordagens baseadas em filas tradicionais.

A organização deste Capítulo é a seguinte: nas Seções 4.1 e 4.2 destacam-se as soluções existentes e o problema abordado. A Seção 4.3 descreve a integração da ideia de delegação de fluxo no *framework* MDN. A Seção 4.4 mostra os experimentos e resultados desta integração. Por fim, a Seção 4.5 faz as considerações finais.

4.1 TRABALHOS RELACIONADOS

Como a solução apresentada aqui se enquadra em duas questões na área de SDN: compatibilidade e desempenho. Os trabalhos relacionados foram agrupados em dois grupos. O primeiro diz respeito aos trabalhos feitos para melhorar a compatibilidade da rede, e o segundo refere-se aos trabalhos que visam garantir níveis de QoS.

4.1.1 Compatibilidade da Rede

A composição de ambientes SDN por diferentes tipos de dispositivos de rede e protocolos não é algo novo. Porém, tal heterogeneidade se tornou mais evidente com o surgimento de novas versões de protocolos (i.e., lançamentos do protocolo OpenFlow), controladores SDN e *switches*. Esse cenário torna-se um problema quando a compatibilidade entre estes elementos de rede é analisada. Não é coincidência que diversos pesquisadores e fornecedores de equipamentos de rede têm proposto mecanismos para que operadores de rede e desenvolvedores possam lidar com questões de compatibilidade. Neste contexto, tais mecanismos podem ser colocados em três categorias: abordagens orientadas ao plano de encaminhamento, abordagens orientadas ao plano de controle e abordagens orientadas a serviço.

Das **abordagens orientadas ao plano de encaminhamento**, Mendonca, Obraczka e Turletti (2012) publicaram um dos primeiros trabalhos focados em analisar a relação entre SDN e ambientes de rede heterogêneos. Este trabalho iniciou uma discussão sobre a necessidade de um *framework* que abstraísse o plano de encaminhamento, abrangendo diferentes dispositivos de rede para habilitar um ambiente SDN. Um outro trabalho focado no plano de dados para resolver a questão da compatibilidade é o *Protocol Oblivious Forwarding* (POF) – um *framework* que remove a dependência de configurações de protocolo específicas (e.g., OpenFlow) dos dispositivos de encaminhamento.

Quanto às **abordagens orientadas ao plano de controle**, A partir da perspec-

tiva do plano de controle, o trabalho apresentado por Rinaldi et al. (2015) propõe um controlador SDN adaptativo genérico baseado no protocolo OpenFlow como uma solução para manipular infraestruturas heterogêneas de *smart grids*. Entretanto, este trabalho não deixa claro como lidar com possíveis diferentes capacidades destas infraestruturas subjacentes. Em Anwer et al. (2013), os autores também propõem um controlador SDN, mas visam prover um plano de controle para *middleboxes* de rede (i.e., abstraindo o plano de encaminhamento) que suportem infraestruturas heterogêneas (e.g., NetFPGAs, GPUs).

Por fim, as **abordagens orientadas a serviço** envolvem a descrição padronizada de serviços disponibilizados dentre os dispositivos da rede. O principal exemplo desta categoria é o projeto NETCONF/YANG do IETF que propõe uma forma padronizada para especificar e controlar dispositivos de rede (SCHONWALDER; BJORKLUND; SHAFER, 2010). Um outro exemplo é o Heter-sdn (SHU, 2017) – controlador SDN que fornece um modelo em nível de serviço para aplicações de rede em redes de nuvem heterogêneas.

A extensão apresentada neste capítulo pode ser categorizada como uma abordagem voltada para o plano de encaminhamento, apesar de envolver elementos dos dois outros níveis (i.e., serviço e controle). Porém, diferencia-se dos trabalhos anteriores por fazer uso da implementação padrão do protocolo OpenFlow para aumentar a contabilidade entre aplicações e dispositivos de encaminhamento. Assim, como a maioria das propostas da literatura, este trabalho foca em redes SDN baseadas no protocolo OpenFlow. Entretanto, a ideia de delegação de fluxo associada à modelagem de aplicações poderia ser estendida para qualquer rede programável baseada em fluxos.

4.1.2 Qualidade de Serviço em SDN

Muitos anos antes de SDN surgir como um paradigma de rede, dois *frameworks* principais de QoS dominaram a discussão na literatura para redes IP cabeadas (GHORBANZADEH; ABDELHADI; CLANCY, 2017): **IntServ** e **DiffServ**. O primeiro realiza a reserva de recursos para cada fluxo passando através dos dispositivos de rede, e o segundo classifica os fluxos para definir o tráfego. Com SDN, as discussões sobre estes *frameworks* ganharam notabilidade novamente (KARAKUS; DURRESI, 2017). Pesquisadores agora possuem uma outra perspectiva sobre como implementar *frameworks* QoS para redes programáveis. Tal possibilidade foi permitida desde a primeira versão do protocolo OpenFlow (i.e., o padrão *de facto* para SDN), o qual apresentou um mecanismo de enfileiramento para fluxos específicos.

Neste contexto, de forma não surpreendente, diversas propostas relevantes para o provisionamento de QoS foram identificadas neste trabalho. A maioria delas aborda a questão de QoS pela reserva de recursos (SHARMA et al., 2014; AKELLA; XIONG, 2014; TOMOVIC; PRASAD; RADUSINOVIC, 2014; AFAQ; REHMAN; SONG, 2015) ou pelo enfileiramento de tráfego (WANG et al., 2015; CABA; SOLER, 2015). Em (SHARMA et al., 2014), os autores

propuseram um *framework* baseado em um sistema autônomo para entregar tráfego de alta prioridade. Akella e Xiong (2014) aplicaram uma outra abordagem de enfileiramento para obter reserva de largura de banda em uma rede de fornecedor de nuvem. Tomovic, Prasad e Radusinovic (2014) também alocaram recursos pelo enfileiramento de fluxos para satisfazer níveis de QoS de aplicações multimídia, enquanto Afaq, Rehman e Song (2015) focaram no fornecimento de QoS na presença de *elephant flows*. Uma característica comum destas abordagens é a utilização do OVS e sua funcionalidade de *queue*, a qual não é mais suportada por versões recentes do OpenFlow (i.e., versões acima da 1.3). A solução descrita aqui difere-se das anteriores por evitar o uso de técnicas de enfileiramento (i.e., utilizando medições e estatísticas no lugar) e aplicando apenas regras padrão do OpenFlow para fornecer QoS.

Entretanto, a solução discutida aqui não é a única a realizar este tipo de implementação. As propostas OpenE2EQoS (LIN et al., 2016), OpenQoS(EGILMEZ et al., 2012) e *Recover-aware Steiner Tree* (RST) (SHEN et al., 2015) são exemplos que focam no roteamento dinâmico para alcançar QoS. OpenE2EQoS utiliza dois algoritmos em um controlador SDN para obter as rotas de acordo com a largura de banda disponível dos *links* da rede e a predição da largura de banda necessária para uma aplicação. O OpenQoS também propõe um controlador para obter roteamento QoS dinâmico baseado em um algoritmo Lagrangeano. Em Shen et al. (2015), os autores apresentam o RST como um *framework* para re-rotear tráfego multimídia considerando rotas melhores.

O mecanismo de DF para QoS apresentado aqui é mais próximo da ideia do OpenE2EQoS, embora este trabalho utilize um método diferente para definir as rotas QoS. Entretanto, ambas as propostas (i.e., mecanismo DF e OpenE2EQoS) diferem do OpenQoS e RST por não determinar nova rota para tráfego prioritário. Ao invés disso, o mecanismo DF e o *framework* OpenE2EQoS re-roteiam o tráfego com prioridade mais baixa. Comparada com os trabalhos anteriores, a solução apresentada aqui tem o potencial para superar a perda de pacotes, já que o mecanismo DF recebe informações do modelo de rede e verifica se existem rotas melhores na topologia antes de realizar a delegação de fluxo.

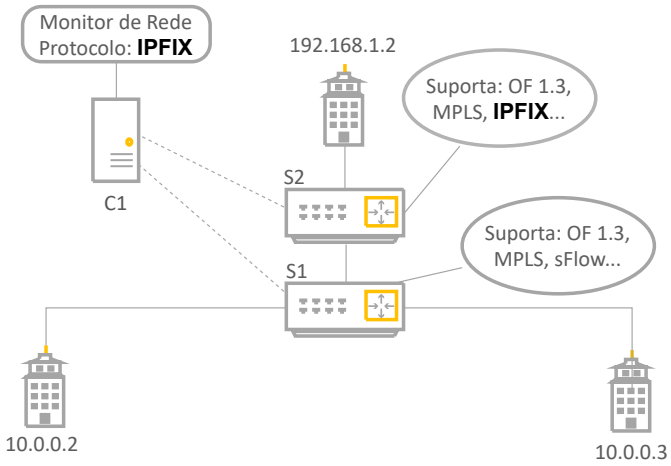
4.2 DESCRIÇÃO DO PROBLEMA

Para clarear a discussão deste capítulo, o problema abordado foi dividido em duas questões: a primeira envolve a compatibilidade em redes heterogêneas; a segunda discute como satisfazer algum requisito de aplicação por nível de QoS. Para a primeira parte, o caso de uso de uma aplicação de monitoramento de rede que deve ser executada em uma rede SDN heterogênea é usado como exemplo. Suponha que este monitor de rede utiliza o protocolo IPFIX para realizar seu procedimento de monitoramento. Além disso, assuma que o fluxo a ser monitorado é definido por pacotes com o endereço IP 10.0.0.2 como origem e 192.168.1.2 como destino.

A Figura 23 ilustra este cenário. Neste caso, o *switch S1* (i.e., dispositivo envolvido no monitoramento) não suporta o protocolo IPFIX requirido pela aplicação hipotética. Entretanto, quando considera-se a capacidade da rede, existem switches (*S2*) que suportam tal protocolo. Então, a primeira parte do problema em discussão é uma aplicação que não é diretamente suportada pelo ambiente (i.e., monitoramento dos fluxos *S1* como alvo), embora esta rede possua dispositivos capazes (i.e., *S2*) de suportá-la. Agora, considere que é possível utilizar as capacidades do *switch S2* para realizar os procedimentos de monitoramento para o *switch S1*. Este é um mecanismo a ser discutido com mais detalhes na Seção 4.3. Por enquanto, é suficiente saber que essa capacidade é alcançada a partir da replicação de fluxos (ou ao menos seus cabeçalhos) para um *switch* auxiliar (i.e., *S2*) dentre aqueles capazes de atender ao requisito da aplicação.

Esta mesma ideia pode ser aplicada para satisfazer requisitos de aplicações por níveis de QoS. Por exemplo, supondo que a largura de banda requisitada por uma aplicação de *streaming* de vídeo não esteja disponível no caso dos elementos ou caminhos de rede estarem sobrecarregados. Pode-se fornecer melhores garantias de QoS a partir do uso de rotas alternativas ou *switches* para processar o tráfego primário ou secundário (da perspectiva da aplicação de *streaming*) (WANG et al., 2015). O desafio é manter os fluxos QoS isolados de outros fluxos e remover a dependência tradicional de reserva de recursos (i.e., descarte de fluxos secundários) e filas de prioridade, já que estas abordagens prejudicam outros requisitos de aplicação (KARAKUS; DURRESI, 2017; EGILMEZ; CIVANLAR; TEKALP, 2012). A eliminação de tal dependência pode evitar efeitos adversos (e.g., perda de pacotes e atraso) em fluxos não-QoS (e.g., tráfego de monitoramento), como discutido por Wang et al. (2015), Egilmez, Civanlar e Tekalp (2012), Caba e Soler (2015).

Figura 23 – Topologia hipotética e capacidades da rede.



Fonte: O autor (2020).

4.3 DELEGAÇÃO DE FLUXO BASEADA EM MODELOS

Para melhorar a compatibilidade entre aplicações SDN e sua infraestrutura subjacente, propõe-se uma delegação de fluxo baseada em modelos para lidar com aplicações que uma rede SDN pode não suportar diretamente, embora a capacidade para este suporte exista em determinados pontos (ou pelo conjunto deles) na rede. Para realizar isso, essa proposta envolve a adição de dois componentes na arquitetura do *framework* MDN (cf. 3.2): (i) um componente chamado **Avaliador de Capacidade e Suporte**, que avalia a compatibilidade da rede a partir da comparação entre modelos de aplicação e infraestrutura; e (ii) um componente chamado **Delegação de Fluxo**, que possui dois tipos de comportamento a depender das aplicações modeladas – este mecanismo provê uma estratégia de espelhamento para melhorar a compatibilidade e buscar por rotas alternativas para satisfazer requisitos de QoS.

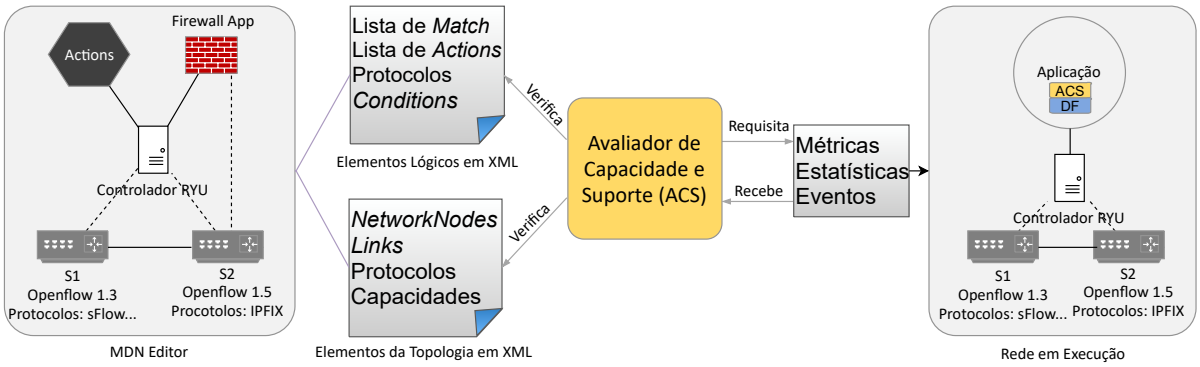
4.3.1 Avaliador de Capacidade e Suporte

A comparação entre os elementos utilizados por uma aplicação (e.g., protocolos, cabeçalhos) e aqueles suportados pela rede é realizada no componente Avaliador de Capacidade e Suporte (ACS). O ACS desempenha este processo linearmente buscando o modelo MDN – composto da lógica das aplicações, requisitos e capacidades da rede. No cenário de pior caso, este processo possui um desempenho $O(N)$, onde N é a quantidade de dispositivos a verificar. A Figura 24 mostra que um modelo MDN pode possuir os dados sobre as aplicações modeladas, bem como a especificação da infraestrutura de rede. Esta informação alimenta o componente ACS, o qual compara a compatibilidade entre os requisitos de aplicação tanto com os elementos diretamente envolvidos nos procedimentos desta aplicação quanto com elementos vizinhos. Depois do processo de geração de código (cf. Seção 3.1), o componente ACS torna-se parte da camada de aplicação (com *threads* monitorando a rede) para lidar com comparações de métricas QoS ou mudanças nas capacidades da rede. Desta forma, o componente ACS também recebe estatísticas e informações de capacidade dos *switches* candidatos ou caminhos alternativos em tempo de execução para oferecer as melhores alternativas para uma decisão de delegação de fluxo.

4.3.2 Delegação de Fluxo para Compatibilidade

Enquanto o componente ACS apenas avalia a compatibilidade entre aplicações de rede e a infraestrutura (retornando possíveis *switches* candidatos se houver uma questão de incompatibilidade), o mecanismo DF, em seu comportamento orientado à compatibilidade, visa prover caminhos alternativos para executar aplicações na rede. O papel do

Figura 24 – Informação utilizada pelo componente ACS e como, juntamente com o componente DF, são implantados na rede.



Fonte: O autor (2020).

ACS começa quando o componente ACS avalia a rede como capaz de suportar – ainda que indiretamente – a aplicação alvo (i.e., existem elementos de rede que suportam a aplicação). Este suporte é aproveitado a partir da utilização do mecanismo de DF, o qual espelha os fluxos de um *switch* incapaz para um outro *switch* capaz de atender aos requisitos da aplicação, e.g., protocolos suportados. Por exemplo, suponha que uma aplicação esteja tentando desempenhar um procedimento com fluxos passando por um *switch* que não suporte um de seus requisitos, e.g., versão do protocolo OpenFlow. Neste caso, o mecanismo DF realiza o espelhamento de fluxos ou dos cabeçalhos dos pacotes que passam por este *switch*, roteando-os para um outro *switch* capaz de suportar o requisito. Atualmente, este espelhamento é realizado utilizando: (i) o endereço do *switch* capaz fornecido pelo componente ACS; (ii) o caminho que os fluxos espelhados devem seguir; e (iii) a identificação dos cabeçalhos dos pacotes – para separar o tráfego normal do tráfego que foi delegado. Desta forma, o impacto da delegação de fluxo afeta apenas os fluxos e cabeçalhos envolvidos ou requisitados pela aplicação. Para os *switches* auxiliares (aqueles que assumirão o processamento dos fluxos espelhados), obviamente, esta delegação de fluxo deve introduzir um tempo de configuração e tráfego adicionais, dependendo do tamanho e da duração dos fluxos espelhados – um cenário complexo para predizer.

4.3.3 Delegação de Fluxo para Garantias QoS

O mecanismo DF também permite que o MDN forneça conformidade de QoS entre aplicações e infraestrutura da rede. Por exemplo, considere uma aplicação de *streaming* de vídeo que requer uma largura de banda mínima de 1Mbps. Tal cenário é viabilizado pelo mecanismo DF computando caminhos alternativos que satisfaçam o requisito da aplicação, utilizando informação estatística obtida com o componente ACS (verificando a largura de banda disponível nas rotas da rede). Este comportamento para tentar alcançar garantias

QoS é alcançado por: (i) identificar caminhos alternativos que satisfaçam os requisitos da aplicação; e (ii) especificar regras de fluxos para definir a rota do tráfego priorizando fluxos de aplicações baseadas em QoS. Esta abordagem é similar à ideia do *framework* IntServ (BRADEN; CLARK; SHENKER, 1994) para redes tradicionais, juntamente com o *Bandwidth Broker* (NICHOLS et al., 1999). Entretanto, não precisa fazer uso de protocolos de reserva de recursos (e.g., *Resource Reservation Protocol* (RSVP)) (BRADEN et al., 1997) nem lidar com questões de escalabilidade presentes no IntServ – já que os fluxos são redirecionados utilizando regras detalhadas ou genéricas, i.e., implementando os *wildcards* do protocolo OpenFlow e *tags* de metadados (para marcar os fluxos espelhados), dentre os elementos de rede para realizar o *traffic shaping*. O uso de metadados marcando os fluxos também permite que a flexibilidade do *framework* DiffServ (NICHOLS et al., 1999) seja alcançada. A limitação neste mecanismo proposto é a memória TCAM dos *switches* OpenFlow (YEGANEH; TOOTOONCHIAN; GANJALI, 2013b), mas que pode ser superada utilizando mecanismos para também delegar fluxos de um *switch* sobrecarregado para um outro com mais espaço de memória (BAUER; ZITTERBART, 2016).

4.3.4 Implementação dos Componentes ACS e DF no MDN

O primeiro passo na realização da proposta apresentada neste capítulo foi estender o metamodelo do MDN, adicionando e modificando suas entidades, da seguinte forma:

- *App*: modificada para fornecer maior flexibilidade e detalhamento sobre os protocolos e requisitos utilizados pelas aplicações;
- *OpenFlowSwitch*: uma especialização da entidade `switch`, criada para descrever especificações dos *switches* OpenFlow presentes na rede modelada;
- *Instruction*: entidade que reflete o conceito de instrução presente no protocolo OpenFlow;
- *QoS*: entidade criada para habilitar esta funcionalidade para fluxos e regras relacionadas a estes;
- *PacketHeaders*: entidade modificada para permitir a compatibilidade com diferentes versões do protocolo OpenFlow;
- *Rule*: entidade modificada, separada das propriedades da aplicação para permitir a modelagem de regras independentes; e
- *Flow*: inserida para permitir a modelagem de fluxos envolvidos em cenários QoS.

Estas modificações permitiram a descrição semântica por trás da delegação de fluxos para compatibilidade (e.g., capacidades dos *switches*, protocolos de aplicação) e garantias

QoS (e.g., largura de banda, rotas preferenciais). Também foi feita a adição de modificação dos *templates* utilizados na geração de código (cf. Seção 3.2) para refletir estes novos elementos e possibilidades de modelagem de aplicações (e.g., monitores de rede, firewall, balanceamento de carga), além de regras de fluxo independentes (e.g., *forward*, *drop*, *copy*). Nas seguintes subseções serão apresentados os detalhes sobre a implementação dos componentes.

4.3.4.1 Componente ACS

O componente ACS foi implementado como uma funcionalidade compondo a modelagem MDN e sua subsequente geração de código. Esta implementação envolveu a modificação dos *templates* de geração de código anteriores. De forma similar ao que foi explicitado no Capítulo 3, a linguagem EGL foi utilizada para escrever estes *templates*. Assim, se um modelo habilita o componente ACS, os *templates* utilizados na geração de código agora chamam um método para comparar linearmente os requisitos de cada aplicação modelada com as capacidades de cada dispositivo de encaminhamento presente na rede ou no caminho envolvido na execução da aplicação, i.e., o componente ACS verifica cada instância das entidades do metamodelo, comparando seus requisitos com as capacidades da rede (cf. Figura 24). Quando os requisitos QoS são utilizados na modelagem, tal comparação também ocorre em tempo de execução enviando mensagens `FEATURE_REQUEST` para os *switches* envolvidos. Isto é necessário porque, por exemplo, elementos de encaminhamento podem ser modelados com uma largura de banda suficiente para suportar a aplicação, mas em determinado instante de execução da rede eles podem estar sobrecarregados.

4.3.4.2 Componente DF

O núcleo da abordagem discutida aqui faz uso das possibilidades de compatibilidade fornecidas pelo componente ACS e o potencial de programabilidade de redes SDN. O mecanismo do componente DF realiza a delegação de fluxo gerando e inserindo regras OpenFlow no plano de encaminhamento. Para cenário apenas de compatibilidade, este mecanismo recebe a identificação do(s) *switch*(es) incapaz(es) e capaz(es), i.e., $sIncapaz_{id}$ e $sCapaz_{id}$, respectivamente, e obtém uma rota de $sIncapaz_id$ para $sCapaz_id$. Atualmente, esta rota é obtida utilizando o algoritmo de Dijkstra (cf. linha 3). Em seguida, a função `installFlowRules` instala regras `FORWARD` ordenadas baseadas em uma lista de *switches* e portas obtidas pela função `getPath`, utilizando o cabeçalho *metadata* do OpenFlow para identificar os fluxos delegados. Esta identificação é necessária para identificar fluxos delegados e prioritários ao longo da rede.

A linha 11 do Algoritmo 1 lida com aplicações QoS que precisam passar por alguma

rota específica. Por exemplo, aplicações de *streaming* que utilizam *Multiple Description Coding* (MDC) codificam os dados de origem em diferentes descrições (i.e., *bitstreams*). Cada uma destas descrições possui um caminho independente. Para alcançar isso, o motor de geração de código do MDN pode verificar um caminho estático para cada descrição na aplicação modelada e passá-lo como parâmetro para o procedimento `DF`.

Algoritmo 1: Instalando regras ordenadas para delegação de fluxo

```
1  Function flowDelegation(sIncapable_id, sCapable_id, qos, path) : boolean is
2      if path = null then
3          path ← getPath(sIncapable_id, sCapable_id, qos);
4          if path ≠ false then
5              installFlowRules(path);
6          end
7          else
8              return false;
9          end
10     end
11     else if path ≠ null then
12         installFlowRules(path);
13         return true;
14     end
15 end
```

Quando há restrições QoS envolvidas, ocorre uma mudança no processo de obtenção de rotas. Como o componente DF somente executa caso o ACS forneça ao menos uma possibilidade, i.e., caminho ou *switch* alternativo, é preciso tentar alcançar essas garantias QoS da seguinte forma: (i) delegando tráfego não-prioritário para liberar recursos; e (ii) definindo rotas de fluxos prioritários para *switches* ou caminhos capazes. Neste contexto, quando o método `getPath` (cf. Algoritmo 1) recebe parâmetros QoS, ele balanceia o tráfego (i.e., delegando fluxo para rotas alternativos) e analisa a topologia utilizando o algoritmo de Wang-Crowcroft, *Source Routing Algorithm* (SRA) (WANG; CROWCROFT, 1996). Este algoritmo baseia-se na composição de restrições (i.e., largura de banda e atraso) para obter uma rota viável a partir de uma origem que satisfaça à parâmetros QoS. A escolha pelo SRA baseia-se na adequação deste algoritmo para cenários SDN (e.g., visão global da rede) e sua avaliação presente em Tomovic, Radusinovic e Prasad (2015), que mostra um menor tempo de computação alcançado pelo algoritmo em comparação à outras abordagens.

O SRA foi implementado considerando que, durante a delegação de fluxo, existe uma origem (i.e., rota incapaz) e um destino (i.e., rota capaz). Apesar de o núcleo do algoritmo não ter mudado, neste trabalho, foi feita uma adaptação para redirecionar fluxos não-prioritários para rotas alternativas. Tal estratégia resulta na liberação de largura de banda e uma menor perda de pacotes (um problema inerente do algoritmo SRA). O Algoritmo 2, criado para receber rotas prioritárias ou obter rotas alternativas que atendam a requisitos

de QoS, mostra que a implementação pode receber informações sobre o atraso e a largura de banda como parâmetros do modelo MDN (cf. linha 1, $[d, bw]$). Além disso, nas linhas 4 e 5, a largura de banda bem como as taxas de atraso podem ser definidas em tempo de execução pela aplicação gerada (i.e., métodos `getDelayForAll` e `getBandwidthForAll`). Depois de definir estes valores e seguir o algoritmo SRA, pode-se descartar *links* (cf. linhas 8-9) que não atendam ao requisito de largura de banda mesmo depois de um mecanismo DF (cf. linha 7). Esse descarte é feito definindo o valor de atraso ($d_{i,j}$) para MAX no(s) *switch*(es) relacionado(s). As linhas seguintes encontram e balanceiam o caminho contendo o menor atraso D para o nó $dst_i d$ em uma sequência lógica similar ao que faz o algoritmo de Dijkstra (porém, considerando a taxa de atraso). O SRA termina sua execução quando (i) ou o nó dst_{id} é inserido no caminho P ou quando a taxa de atraso excede o limiar antes de atingir o nó dst_{id}. Assim como na versão original do SRA, a implementação exibida aqui implementação possui um tempo de computação de $O(N^2)$ no pior caso.

Algoritmo 2: Obtendo uma rota orientada à QoS

```
1  Function getQoSPath(dp_id, dst_id, qos, [d, bw]): optional is
2      i, j ← getSwitchIds();
3      delay_list ← getDelayForAll(d);
4      bandwidth_list ← getBandwidthForAll(bw);
5      forall d_{i,j} in delay_list and b_{i,j} in bandwidth_list do
6          if b_{i,j} < qos_bandwidth then
7              if releaseBandwidth(b_{i,j}) = false then
8                  d_{i,j} = MAX;
9              end
10         end
11     end
12     P ← {dp_id};
13     forall i ≠ dp_id do
14         D_i = b_{dpid,i};
15     end
16     k ← i;
17     forall k not in P do
18         D_k ← min_{i∉P} D_i;
19         if D_k > qos_delay then
20             if balanceTraffic(k, qos_delay) = false then
21                 return false;
22             end
23         end
24         else if dst_id in P then
25             return P;
26         end
27         else
28             P ← P ∪ {k};
29         end
30     end
31     forall i not in P do
32         D_i ← min[D_i, D_k + d_{k,i}];
33     end
34 end
```

4.4 AVALIAÇÃO E DISCUSSÃO

Para avaliar os componentes introduzidos neste Capítulo, foram realizados os experimentos descritos nas subseções a seguir. O hardware utilizado para os experimentos estava equipado com um processador Intel Core i705500U 2.40GHz e 8GB de memória. Estes experimentos consistiram na verificação dos seguintes cenários: **avaliação da compatibilidade** e desempenho do mecanismo DF para **compatibilidade** e **QoS**. Estes cenários foram avaliados considerando diferentes topologias (cf. Tabela 3) – exceto para a avaliação de compatibilidade. Os resultados foram obtidos em um nível de confiança de 95%, calculando o valor médio de cada métrica para cada topologia: compatibilidade, sobrecarga do mecanismo DF e comparação com solução baseada em fila. Foram realizadas 30 repetições para cada avaliação, totalizando 900 experimentos.

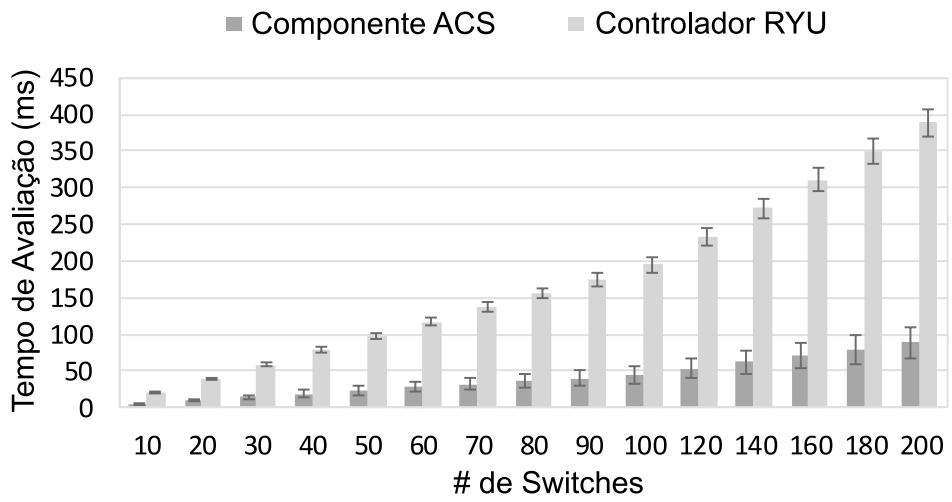
4.4.1 Avaliação de Compatibilidade

Para verificar o desempenho do componente ACS em seu procedimento de comparar requisitos de aplicação e infraestrutura, a avaliação considerou um número crescente de *switches* em uma topologia arbitrariamente definida em anel modelada com o MDN Editor, lidando sempre com as rotas envolvidas no pior caso (i.e., *switch* auxiliar posicionado mais distante possível do *switch* incapaz). O mesmo procedimento do ACS foi realizado em uma rede Mininet simulada (sem tráfego externo).

Neste cenário, um requisito específico de aplicação foi buscado dentre os *switches* gerenciados por um controlador Ryu. Em ambos os experimentos, o cenário de pior caso foi considerado (i.e., na segunda perspectiva, apenas o último *switch* avaliado seria capaz de satisfazer o requisito da aplicação). Os resultados (cf. Figura 25) mostram que o componente ACS avaliando 200 *switches* em até 85 milissegundos, enquanto a mesma avaliação poderia ser 4x mais lenta se realizada diretamente na rede simulada através do Mininet. Este resultado, obviamente, é devido à dependência de controladores SDN – baseados no protoclo OpenFlow – de mensagens `FEATURE_REQUEST` para verificar as capacidades dos *switches*, as quais estão sujeitas ao *handshake* do protocolo OpenFlow e de atrasos na rede, enquanto o componente ACS verifica computacionalmente o modelo da rede. Em caso de mudanças na rede, o componente ACS recebe notificações da rede subjacente e atualiza o modelo da rede.

Os resultados deste experimento mostram uma primeira vantagem de possuir informações em tempo de execução que ajudem a gerenciar a rede. Essa vantagem evidencia como MART podem auxiliar neste gerenciamento, e compõe parcialmente a resposta da **Questão de Pesquisa 2** sobre quais formas de realizar o gerenciamento autonômico de redes SDN.

Figura 25 – Desempenho do pior caso na comparação do tempo de validação médio entre o componente ACS e a verificação padrão do controlador Ryu.



Fonte: O autor (2020).

4.4.2 Delegação de Fluxo para Compatibilidade

O segundo componente, i.e., mecanismo DF, deste capítulo foi avaliado com base em três métricas: (i) tempo para realizar a delegação de fluxo, incluindo o tempo de descoberta de rotas partindo de *switches* **incapazes** para *switches* **capazes**; (ii) e sobrecarga no tempo, i.e., atraso médio, ao usar este mecanismo. Estas três métricas foram definidas para verificar a viabilidade de utilizar o mecanismo DF no cenário modelado. Para obtenção destas métricas, o MDN Editor foi utilizado para modelar uma aplicação de monitoramento de rede (M) baseada no protocolo IPFIX, com o controlador Ryu como alvo. Além disso, dez topologias arbitrariamente escolhidas do Topology Zoo (KNIGHT et al., 2011) foram modeladas. Antes de iniciar os experimentos, a complexidades das topologias escolhidas foi analisada utilizando o algoritmo de Johnson para contar a quantidade de *loops* e a distância máxima de cada topologia (cf. Tabela 3). Em cada experimento e topologia, o Mininet foi utilizado para simular as topologias e o requisito mínimo necessário para utilizar o componente de delegação de fluxo: ao menos um *switch* suportando M (i.e., capaz de manipular o protocolo IPFIX).

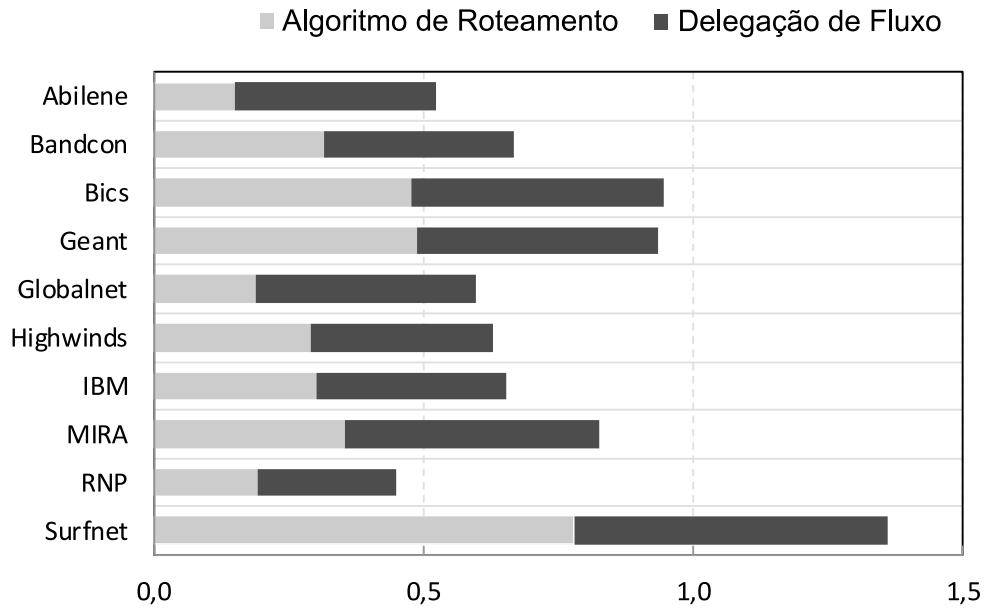
A métrica do tempo para realizar a delegação de fluxo é baseada em dois instantes do tempo (T): T_1 é marcado quando um *switch* incapaz recebe o primeiro pacote de fluxo (i.e., protocolo de monitoramento não suportado); e T_2 é definido quando o *switch* capaz recebe o primeiro pacote dos fluxos espelhados. O tempo delegação de fluxo é, então, a diferença simples entre $T_2 - T_1$. A Figura 26 ilustra o resultado deste experimento. Para colher as métricas desejadas, mediu-se o tempo de execução para obter o caminho até dst_{id} e o tempo de execução para realizar a própria delegação de fluxo separadamente. É preciso destacar que o mecanismo DF somente executa o algoritmo de roteamento quando

Tabela 3 – Descrição das topologias utilizadas nos experimentos.

Nome da Topologia	# de Switches	# de Links	# de Loops	Diâmetro
Abilene	11	14	10	5
Bandcon	21	28	53	6
Bics	33	48	1529	8
Geant	40	61	12379	8
Globalnet	8	7	0	4
Highwinds	18	31	455	6
IBM	18	24	76	6
MIRA	15	28	1491	5
RNP	27	30	673	10
Surfnet	50	68	2562	11

Fonte: O autor (2020).

Figura 26 – Tempo de execução do mecanismo DF em diferentes topologias.



Fonte: O autor (2020).

não existem rotas válidas instaladas nos *switches*. Além disso, o tempo de delegação de fluxo tende a diminuir após a primeira execução, já que ele não precisará instalar regras de fluxos novamente. Então, é possível observar um tempo de execução razoável mesmo para topologias complexas, i.e., com alto número de *loops*, como Surfnet e Geant.

Os resultados também sugerem que a complexidade da rede não é o único fator afetando o tempo de execução. Os tamanhos das topologias e, obviamente, as distâncias entre os *switches* incapazes e capazes foram os principais fatores identificados que aumentam o tempo de execução. Por exemplo, o maior tempo de execução para realizar a delegação de fluxo na topologia Surfnet comparada com a Geant, embora a Surfnet possua um menor número de *loops*, é devido ao número de saltos necessários para delegar fluxo em cada

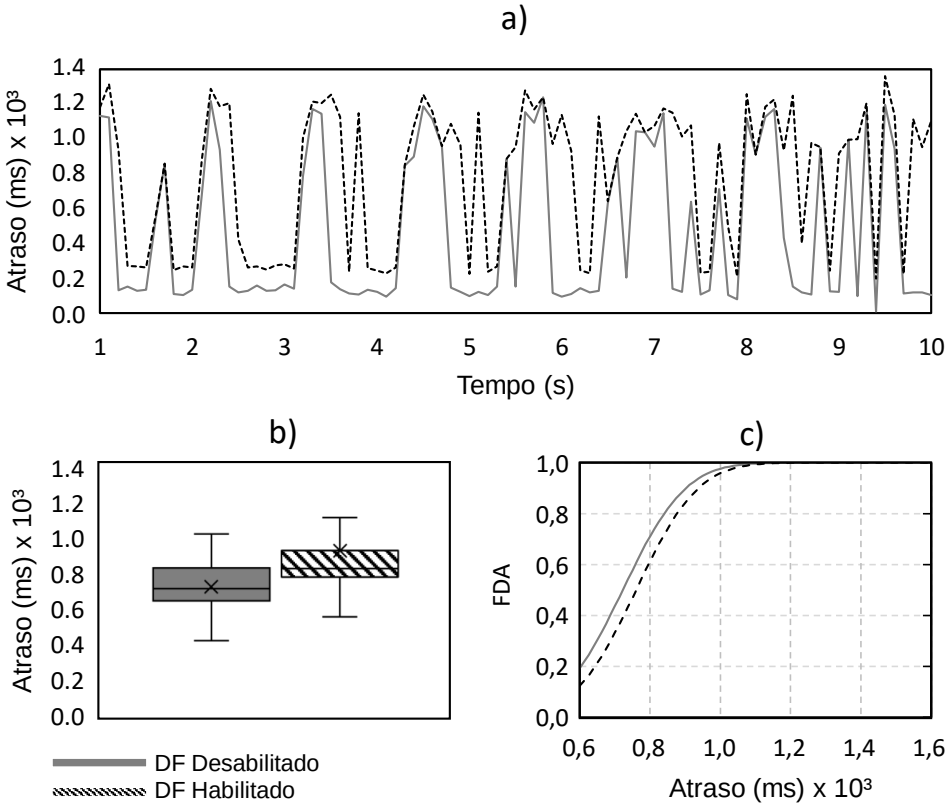
uma delas. Note que a máxima distância entre *switches* capazes e incapazes foi forçada em todos os experimentos.

O segundo experimento deste cenário verificou a sobrecarga do mecanismo DF. O *Distributed Internet Traffic Generator* (D-ITG) (BOTTA; DAINOTTI; PESCAPÈ, 2012) foi utilizado para gerar tráfego e obter granularidade em nível de pacote. Além disso, o Mininet e o OVS foram utilizados para simular o processamento dos fluxos. Para este cenário de avaliação, dois *hosts* virtuais i.e., h_1 e h_2 foram adicionados às topologias (cf. Tabela 3), conectando estes *hosts* a um *switch* incapaz S_1, enquanto a aplicação de monitoramento M verifica o fluxo entre eles. Além disso, todas as topologias possuem apenas um *switch* S_2 capaz de suportar M, e este *switch* foi posicionado na distância máxima em relação à S_1 (i.e., a distância entre S_1 e S_2 é igual à coluna Diâmetro na Tabela 3). Por fim, os fluxos de h_1 para h_2 foram enviados utilizando a seguinte configuração: 100 fluxos consecutivos (UDP, taxa constante de envio de pacotes em 100Mbit/s) com 10 segundos de duração (cf. Figura 27a). Vale mencionar que a janela de tempo não considera o tempo de *warm-up* relativo à instalação das regras OpenFlow e à descoberta de rota. Assim, para se avaliar a sobrecarga causada pelo mecanismo DF (i.e., a cópia de fluxos de S_1 para S_2), mensurou-se o quanto este mecanismo aumenta o atraso para os fluxos monitorados por M. Os picos na Figura 27a referem-se às trocas de mensagens OpenFlow e D-ITG (e.g., `Flow-Mod` e notificações para fluxos encerrados). No fim deste experimento, foi possível observar que utilizar o mecanismo DF em uma aplicação aumenta em até 15% a taxa de atraso para os fluxos envolvidos (cf. Figura 27b). Esse aumento na taxa de atraso deve ser observado para que haja um balanceamento adequado entre perda de desempenho e aumento de compatibilidade. A variabilidade que ocorre é devido ao número crescente de mensagens causada pelo próprio mecanismo de delegação de fluxos. A Figura 27c mostra a Função de Distribuição Acumulada (FDA) para esta última métrica, onde é possível perceber como se comporta o aumento do atraso obtido com o mecanismo DF habilitado.

4.4.3 Delegação de Fluxo para QoS

O objetivo nesta etapa é demonstrar a viabilidade de utilizar o mecanismo DF para obter garantias de QoS. Esta avaliação ocorreu no seguinte cenário: o MDN Editor foi utilizado para modelar a topologia MIRA (cf. Figura 28), escolhida arbitrariamente, e, assim, gerar código para criar uma simulação no Mininet. Neste modelo, alguns *links* foram definidos com capacidade de 15Mbps e outros com capacidade de 10Mbps. A Figura 28 ilustra essa definição. Também foram criados dois *hosts*, h_1 e h_2, e um servidor S. Além disso, um requisito de QoS foi modelado, supondo uma aplicação de transmissão de vídeo (VS) baseada em TCP, que requer uma largura de banda mínima de 7Mbps para transmitir um vídeo com resolução de 1080p durante 30 segundos de S para h_1 (veja o Fluxo B na Figura 28). Então, um outro fluxo foi simulado para inserir tráfego cruzado na

Figura 27 – O gráfico a) mostra como o atraso médio dos fluxos D-ITG são afetados pelo mecanismo DF. Em b), um box plot ilustra o atraso médio para o primeiro e terceiro quartil das amostras: 750ms com o DF desabilitado e 860ms quando habilitado. A Figura c) apresenta a FDA para a mesma comparação.

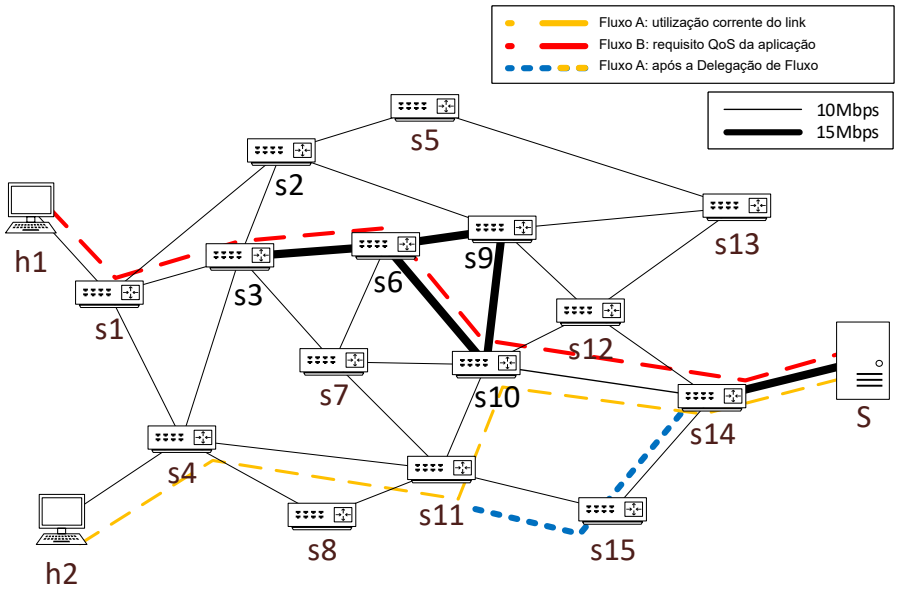


Fonte: O autor (2020).

rede, com a transmissão de um vídeo com resolução de 720p (veja o Fluxo A na Figura 28) durante 25 segundos, entre S e h_2 – sem garantias de QoS. Esta última transmissão utiliza ∼4Mbps do *link* $l(s_{14}, s_{10})$. Como l é também parte da rota $S \rightarrow h_1$, este é o gargalo que o mecanismo DF precisa resolver. Ambos os fluxos foram gerados utilizado a aplicação *iperf*. Assim, no experimento foi possível observar o comportamento destes fluxos antes, durante e depois do processo de delegação de fluxo. A transmissão entre $S \rightarrow h_2$ inicia primeiro. Após 5 segundos, o fluxo $S \rightarrow h_1$ tem início. Assim, já que $l(s_{14}, s_{10})$ não é capaz de satisfazer o requisito QoS, o mecanismo DF realiza a instalação das regras para definir o tráfego de acordo com o procedimento `getQoSPath` e as informações fornecidas pelo componente ACS. A linhas tracejadas em azul e vermelho, na Figura 28, ilustram o resultado deste processo, depois de balancear o tráfego removendo e instalando novas regras de fluxo.

Avaliou-se ainda o desempenho do mecanismo de delegação de fluxo para satisfazer ao requisito de QoS de VS, comparando o desempenho obtido com a abordagem padrão baseada em filas do controlador Ryu. Ambos os métodos foram definidos para satisfazer ao requisito do Fluxo B (i.e., 7Mbps de largura de banda). A Figura 29 mostra os resul-

Figura 28 – Delegação de fluxo na topologia MIRA.

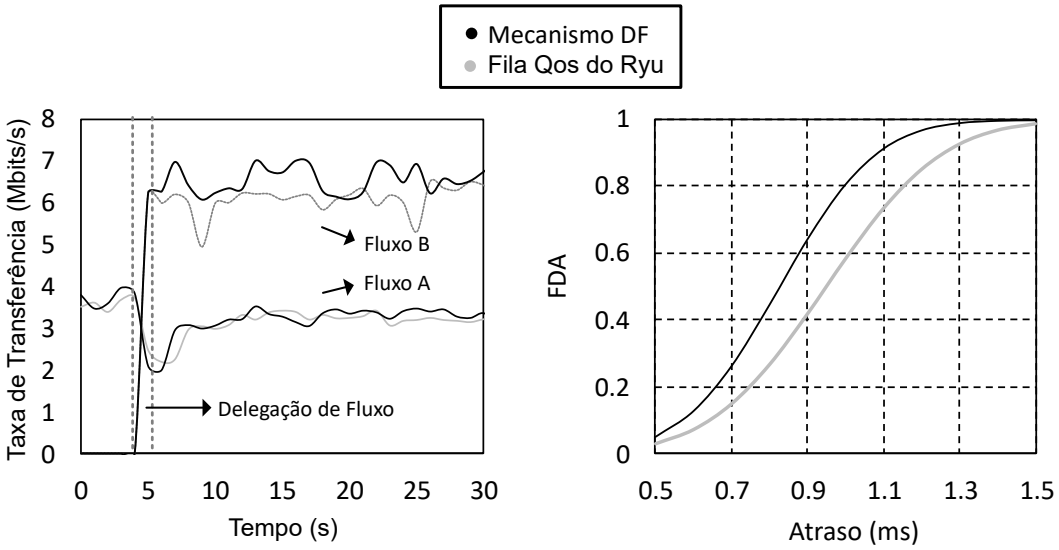


Fonte: O autor (2020).

tados desta avaliação para cada abordagem: (i) o gráfico da esquerda apresenta a taxa de transferência obtida durante o experimento para cada fluxo; e (ii) o gráfico da direita ilustra a FDA obtida utilizando o valor de variância no atraso ao utilizar o mecanismo DF comparado com a utilização da técnica de enfileiramento padrão do controlador Ryu. A partir destes resultado, ficou constatado que, apesar de o foco inicial não ser o melhoramento de métricas de desempenho – mas sim em demonstrar a aplicabilidade de modelos e do mecanismo de delegação de fluxo neste cenário – utilizar regras de fluxo para delegar fluxos prioritários, ao invés de adotar uma estratégia baseada em fila, melhorou a utilização da taxa de transferência dos fluxos QoS em até 9%. Além disso, ocorreu uma menor variação de atraso na mesma comparação.

A explicação para estes resultados é que, ao utilizar o mecanismo DF, o *switch* precisa apenas realizar a combinação de regras habitual, tornando o processo mais direto. Abordagens baseadas em fila, por outro lado, necessitam comparar os fluxos entrantes com cada fila e seus valores condicionais (em adição ao processo de combinação de regras e fluxos, i.e., *flow matching*).

Figura 29 – Esquerda: comparação da utilização de largura de banda por cada abordagem para satisfazer o requisito de QoS. Direita: FDA de taxa de atraso para ambos os fluxos, comparando cada abordagem. O mecanismo DF fornece menor atraso, já que considera a largura de banda disponível ao longo da rota na instalação de regras, enquanto a abordagem baseada em filas realiza a priorização utilizando verificações de filas e condições no *switch*.



Fonte: O autor (2020).

4.5 CONSIDERAÇÕES FINAIS

Neste Capítulo, o *framework* MDN foi estendido para aproveitar as capacidades totais de uma infraestrutura SDN, melhorando a compatibilidade entre aplicações e a infraestrutura subjacente. Tal extensão envolveu a adição de dois novos componentes, os quais realizam a avaliação da rede a uma possível delegação de fluxos, permitindo que aplicações aparentemente não suportadas por um dispositivo incapaz possam utilizar as capacidades de um outro elemento de rede para desempenhar seus procedimentos. A mesma ideia foi aplicada com sucesso para prover garantias QoS. Neste caso, ao invés de utilizar um dispositivo específico, um dos componentes adicionados molda o tráfego para satisfazer a um certo requisito de qualidade de serviço.

Os experimentos demonstraram que a solução apresentada neste capítulo atingiu resultados razoáveis para melhorar a compatibilidade entre aplicações e infraestrutura, enquanto mantém níveis aceitáveis de desempenho, permitindo a utilização dos componentes apresentados em soluções de desenvolvedores ou operadores de rede para facilitar a implementação de aplicações ao lidar com ambientes heterogêneos. Além disso, quando o mecanismo DF foi utilizado para prover garantias QoS, a solução proposta alcançou resultados melhores que implementações padrão.

5 TRANFORMAÇÃO DE MODELOS PARA PREDIÇÃO DE DESEMPENHO DE REDES DEFINIDAS POR SOFTWARE

No capítulo anterior, foi descrita a possibilidade de adicionar componentes na geração de código a partir de modelos, permitindo a reação à mudanças em rede SDN (e.g., delegar fluxo para atender requisitos de QoS). Porém, cenários de heterogeneidade e em constante mudança possuem desafios que a avaliação linear das capacidades da rede feita anteriormente não consegue superar. Por exemplo, é possível que a delegação e fluxo ocorra para liberar uma rota alternativa, mas essa rota não atinja a largura de banda necessária para que uma determinada aplicação funcione adequadamente. Predizer o desempenho da rede em **tempo de *design*** (i.e., antes da implantação dos serviços e dispositivos de rede) é um desafio em aberto. Por exemplo, o desempenho de uma rede SDN pode ser representado em múltiplos modelos, e.g., Redes de Petri (RdP), modelos analíticos e através de Teoria das Filas. Também existem ferramentas de simulação (e.g., Mininet) e *benchmarking*, porém cada ferramenta possui seu próprio formalismo, composição e métrica – o que: (i) requer diferentes conhecimentos e experiências para criar a simulação; e (ii) geram resultados diferentes.

Este cenário torna-se um problema para operadores de rede ao projetarem suas redes. Escolher a representação mais adequada não é uma tarefa trivial, e a melhor decisão (i.e., aquela que provê simulações mais realistas) deve variar de acordo com as características da rede (e.g., serviços em execução, topologia). Neste sentido, buscando responder às questões de pesquisa definidas para esta tese, mais especificamente a **Questão de Pesquisa 3** (cf. Seção 1.2), sobre *o impacto de utilizar uma abordagem baseada em modelos no desenvolvimento e gerenciamento de SDN*, a utilização de modelos em *design time* permite a correção de erros de projeto antes da implantação da rede real, o que evita comportamentos inesperados do ambiente de rede. Para responder a esta questão de pesquisa, uma das possibilidade é verificar a utilização de modelos como uma solução que unifique a descrição de diferentes formalismos de desempenho, obtendo uma avaliação de desempenho melhorada. Além disso, para possibilitar o uso de MART, a predição de desempenho e comportamento da rede é um importante requisito para implementar características *self-** (WICKBOLDT et al., 2015) que norteiam esta tese.

Assim, estende-se aqui a DSML criada para o *framework* MDN, tornando-a capaz de abstrair mais de um modelo de desempenho em uma única descrição de alto nível. Ainda que essa ideia tenha sido investigada anteriormente (FERNANDES, 2017; RYGIELSKI; SELIUCHENKO; KOUNEV, 2016a), os trabalhos anteriores não consideraram a programabilidade permitida pelo plano de controle de redes SDN. Então, um novo formalismo foi definido ao estender a DSML, nomeando essa DSML estendida como *Network Modeling Language* (NML), aplicando o conceito de **transformação de modelos** e permitindo a utilização de uma única descrição de alto nível para obter métricas de desempenho. Considerando

que modelos estocásticos podem ser utilizados para obter estas métricas, os formalismos *Queueing Petri Net* (QPN) e *Stochastic Petri Net* (SPN) serão utilizados nas transformações. Gerar modelos QPN e SPN a partir da linguagem NML permite, então, que a predição de desempenho leve em conta não apenas as capacidades da infraestrutura, mas também a lógica presente nas aplicações que serão executadas no plano de controle.

A partir das ideias e resultados apresentados neste capítulo, as seguintes contribuições são dadas ao estados da arte: (i) uma nova transformação modelo-para-modelo que permite predizer métricas de desempenho baseada em diferentes modelos através de uma única instância de alto nível; (ii) a avaliação de dois modelos estocásticos, considerando a acurácia dos mesmos ao predizer o desempenho de uma rede SDN; e (iii) a demonstração dos desafios e benefícios de utilizar diferentes modelos de simulação para representar cenários SDN distintos.

O restante deste Capítulo está organizado da seguinte forma: na Seção 5.1, delimita-se o problema a ser resolvido utilizando transformação modelo-para-modelo. A Seção 5.2 discute os trabalhos relacionados especificamente na linha de predição de desempenho e modelagem. Na Seção 5.3, apresenta-se o método de modelar cenários SDN para predizer seus desempenhos. E, por fim, as Seções 5.4 e 5.5 apresentam, respectivamente, avaliam experimentalmente a proposta deste Capítulo e as considerações finais.

5.1 DESCRIÇÃO DO PROBLEMA

Diversos formalismos de modelagem para desempenho podem representar redes de computadores (e.g., Teoria das filas, modelos de simulação de domínio específico, RdP). Porém cada formalismo requer as próprias etapas de modelagem e possui diferentes características e benefícios (e.g., tempo de execução/simulação, acurácia). Este cenário resulta na necessidade de mais conhecimento e experiência, uma lacuna que usualmente não pode ser preenchida por projetistas de redes. Além disso, para que o gerenciamento da rede possa ocorrer da melhor forma, diferentes modelos podem ser usados para realizar a análise de desempenho desta rede.

Ao considerar redes SDN, escolher e utilizar um formalismo de modelagem para predizer desempenho torna-se um desafio ainda maior. Como o plano de controle destas redes possui um papel crucial no desempenho, tal plano precisa ser modelado de acordo com o seu comportamento, o qual é definido pelas aplicações executadas pelo controlador SDN. Assim, modelar redes SDN utilizando apenas abordagens tradicionais (e.g., Teoria das Filas, QPN) resulta em uma abstração excessiva para representar o comportamento dos controladores (e suas aplicações). Ou seja, ainda que seja possível predizer o desempenho de redes utilizando estas abordagens, ao basear-se em apenas uma destas abordagens, características lógicas da rede podem ser perdidas e o tempo de simulação pode não ser o menor possível. Neste contexto, SPN é um formalismo que permite descrever de forma

mais acurada o comportamento do plano de controle SDN. Por exemplo, condições de uma aplicação de balanceamento de carga executando no controlador podem ser representadas por um modelo SPN e abstraídas em um modelo QPN. Entretanto, utilizar mais de um formalismo traz de volta o problema dos múltiplos formalismos para representar redes de computadores. Assim, o desafio encarado neste capítulo refere-se à representação de uma rede SDN de tal forma que esta possa ser avaliada por diferentes modelos de desempenho a partir de um único formalismo.

Neste caso, escolher o nível correto de abstração para representar os elementos SDN de tal maneira que estes elementos possam ser transformados em outros formalismos não é uma tarefa fácil. Por exemplo, um *switch* OpenFlow poderia ser representado como uma *posição* nos modelos QPN e SPN. Por outro lado, quando tráfego de redes é envolvido, características como atraso e carga possuem diferentes representações em ambos os modelos. Transformar um modelo de rede SDN de alto nível em modelos de desempenho como RdPs precisa envolver um processo que permita representar conceitos de alto nível em modelos de desempenho subjacentes corretamente.

5.2 TRABALHOS RELACIONADOS

Existe um grande número de trabalhos em modelos de desempenho para predizer o desempenho de redes. As primeiras propostas definiram formalismos baseados em modelos estocásticos (WONG, 1978; KELLY, 1985; HARRISON; PATEL, 1992) para descrever o comportamento da rede e predizer seu desempenho. Abordagens recentes também utilizam modelos estocásticos (e.g., Teoria das Filas em camadas, RdP) para representar redes e predizer seu desempenho. Entretanto, eles focam em partes específicas destas redes (e.g., protocolos, tráfego) (MOKDAD et al., 2014; MAHENDRAN; GUNASEKARAN; MURTHY, 2014) ou em novas arquiteturas (e.g., SDN, *mobile cloud*) (RYGIELSKI; SELIUCHENKO; KOUNEV, 2016a). Assim, como sugerido por Rygielski, Kounev e Tran-Gia (2015), é possível dividir a maioria destas abordagens em modelos de domínio específico (e.g., ferramentas de simulação de rede) e modelos de propósito geral (e.g., cadeias de Markov, RdPs) que são aplicados à modelagem de redes.

Por vários anos, pesquisadores de redes têm usado Ns-2/3 (HENDERSON et al., 2008) e OMNeT++ (VARGA; HORNIG, 2008) como simuladores de eventos discretos para modelar a pilha TCP/IP e outros protocolos de rede populares. Entretanto, apesar de estes simuladores permitirem a modelagem de uma rede e seus protocolos, ambos requerem habilidades avançadas em programação C++ e uma curva de aprendizado acentuada para modelar e implementar modelos de protocolos específicos. No campo de SDN, o Mininet (OLIVEIRA et al., 2014) surgiu como um emulador que simula um ambiente SDN mas possui o mesmo contraponto presente no Ns-2/3 e no OMNeT++, i.e., requer um conhecimento razoável de programação Python para implementar e modelar suas simulações.

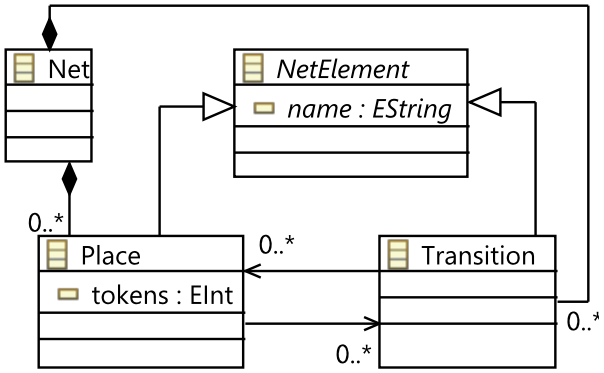
Em outro contexto, a aplicação de modelos de desempenho de propósito geral na modelagem de redes ocorre em dezenas de abordagens. Por exemplo, modelos de RdPs têm sido usados para modelas e analisar a segurança de redes SDN (YAO et al., 2016). Em Xiong et al. (2016), os autores constroem modelo de filas para representar redes baseadas no protocolo OpenFlow e avaliar seu desempenho. Em Fofack et al. (2014), os autores definem um modelo de Markov para predizer o desempenho de redes de cache baseadas no valor de *Time-to-live* (TTL).

Outros trabalhos relacionados utilizam ambas as ideias (i.e., combinam modelos de desempenho de propósito geral em soluções de domínio específico). Em Li et al. (2017), os autores apresentam uma ferramenta para transformar modelos baseados em *Unified Modeling Language* (UML), representando aplicações de dados intensivos, em SPN para realizar a análise de desempenho. Com um conceito similar, Rygielski, Seliuchenko e Kounev (2016a) propõem utilizar um formalismo de alto nível que gera modelos QPN para predizer o desempenho de redes SDN. A solução proposta neste capítulo diferencia-se destas anteriores ao propor uma nova transformação **modelo-para-modelo** superando as limitações relacionadas a uma única representação de ambientes SDN como modelos QPN (e.g., descrição limitada do plano de controle).

5.3 TRANSFORMAÇÕES MODELO-PARA-MODELO

A NML, linguagem proposta neste capítulo, para criar e avaliar o desempenho de modelos SDN, é baseada na extensão do metamodelo MDN (LOPES et al., 2015). Considera-se aqui que, ao usar modelos de alto nível no desenvolvimento ou gerenciamento de SDN, projetistas de rede devem ser capazes não apenas de modelar aplicações SDN, mas também de verificar o desempenho de seus modelos antes da implantação (em *design time*). No próximo capítulo, essa característica será vista como uma aproximação do requisito de predição do gerenciamento autonômico de redes SDN (WICKBOLDT et al., 2015), demonstrando como MART pode viabilizar a implementação deste tipo de gerenciamento. Assim, assume-se que instâncias NML podem ser transformadas em modelos estocásticos, evitando a necessidade de conhecimento dos projetistas de rede sobre modelagem estocástica. Para demonstrar o impacto e benefício da utilização do conceito de transformações modelo-para-modelo (CZARNECKI; HELSEN, 2003), as transformações dos modelos NML para modelos QPN e SPN serão apresentadas – sendo importante ressaltar que a NML suporta transformações adicionais (e.g., modelos de Teoria das Filas).

Figura 30 – Metamodelo de uma rede de Petri.



Fonte: O autor (2020).

5.3.1 Metamodelos

Para realizar as transformações discutidas até aqui, a disciplina de MDE prevê diversos tipos de transformações. Porém, a que mais se enquadra no objetivo deste trabalho, considerando a estrutura estendida do MDN, é a transformação modelo-para-modelo. Nesta transformação ocorre uma "tradução"entre modelos de origem e destino, os quais podem ser instâncias do mesmo ou de diferentes metamodelos (CZARNECKI; HELSEN, 2003). No presente caso, o objetivo é transformar modelos criados com NML em modelos SPN e QPN. Para isso, foi necessário: (i) estender o metamodelo original do MDN com novas entidades para representar o desempenho da rede; e (ii) criar um novo metamodelo para descrever os modelos destino (i.e., QPN e SPN).

Desta forma, em adição ao metamodelo da abordagem MDN, as seguintes entidades foram modificadas: *Link*, adicionando atributos de largura de banda e *delay*; *Traffic*, para permitir a modelagem de distribuição de tráfego na rede; e *Switch*, modificado para descrever capacidades de memória e capacidade de portas. Além disso, foi necessário implementar um novo metamodelo, o qual representa conceitos de RdPs (cf. Figura 30). Este metamodelo foi baseado na Definição 1, a qual apresenta uma versão resumida do formalismo de redes de Petri (ZHOU; ZAIN, 2016):

Definição 1. (Rede de Petri) RP é uma tupla: $RP = \{P, T, F, K, W, M_0\}$ onde:

1. P é um conjunto finito de posições: uma posição é representada por um cículo no modelo RdP;

2. T é um conjunto finito de transições: a transição é representada por um retângulo no modelo RdP;

3. $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto finito de arcos que parte de uma posição para uma transição e vice-versa;

4. $K = \{1, 2, 3, ...\}$ é a capacidade de uma posição p. $K(p)$ representa a quantidade de recursos armazenados na posição (p);

5. $W : F \rightarrow \{1, 2, 3, ...\}$ é uma função de ajuste de peso que representa a quantidade de recursos consumida de uma "posição para uma transição"ou criada a partir uma "transição para uma posição";

6. $M_0 : P \rightarrow \{1, 2, 3, ...\}$ é uma marcação inicial (M_0) que representa a distribuição de recursos para cada posição na fase inicial do modelo RdP. Tais recursos são chamados de *tokens* na teoria de RdP.

Nas seguintes subseções, será possível observar como os modelos NML são transformados nos modelos de RdPs específicos.

5.3.2 Transformação NML-para-QPN

O formalismo QPN possui mudanças substanciais comparadas às redes de Petri originais. QPN adiciona os aspectos de enfileiramento e temporização à definição das posições, as quais também são compostas de *tokens* coloridos. Esta adição separa uma posição em dois tipos: **posições ordinárias** e **posições de enfileiramento**. O primeiro tipo possui um comportamento idêntico ao das posições de RdPs originais. A última integra os conceitos de **fila** e **depósito**, os quais representam o processamento dos *tokens* e seus serviços na fila.

Utilizando a definição de QPN apresentada em Kounev, Spinner e Meier (2012), formalmente considera-se aqui que uma QPN é definida da seguinte forma:

Definição 2. (Queueing Petri Net) QPN é uma tupla: $QPN = \{P, T, C, I^-, I^+, M_0, Q, W\}$ onde:

1. A declaração de P e T é a mesma da Definição 1;

2. C é uma função de cor que atribui um conjunto de cores finito e não-vazio à cada posição e transição;

3. I^- e I^+ são funções de incidência para trás e para frente definidas em $P \times T$, tal que: $I^-(p,t), I^+(p,t) \in [C(t) \rightarrow C(p)_{MS}], \forall (p,t) \in P \times T^1$

4. $M_0 : P \rightarrow \{1, 2, 3, ...\}$ é uma função definidas em P descrevendo a marcação inicial tal que $M_0(p) \in C(p)_{MS}$;

5. $Q = \{\tilde{Q}_1, \tilde{Q}_2, (q_1, ..., q_{|P|})\}$ onde:

   $\tilde{Q}_1 \subseteq P$ é o conjunto de posições de enfileiramento temporizadas,

   $\tilde{Q}_2 \subseteq P$ é o conjunto de posições de enfileiramento imediato, $\tilde{Q}_1 \cap \tilde{Q}_2 = \emptyset$ e

   q_i é a descrição de uma fila levando todas as cores $C(p_i)$ em consideração.

6. $W = \{\tilde{W}_1, \tilde{W}_2, (w_1, ..., w_{|T|})\}$ onde:

   $\tilde{W}_1 \subseteq T$ é o conjunto de transições temporizadas,

   $\tilde{W}_2 \subseteq T$ é o conjunto de transições imediatas, $\tilde{W}_1 \cap \tilde{W}_2 = \emptyset$, $\tilde{W}_1 \cup \tilde{W}_2 = T$, e

   $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ tal que $\forall c \in C(t_i) : w_i(c) \in \mathbb{R}$ é interpretada como a taxa de um distribuição exponencial negativa especificando o atraso de disparo devido à cor c, se $t_i \in \tilde{W}_1$ ou um peso de disparo especificando a frequência de disparo relativa dada a cor c, se $t_i \in \tilde{W}_2$.

Assim, a partir da Definição 2, a transformação a partir de instâncias NML para modelos QPN foi desenhada da seguinte forma (considere entidades NML escritas na fonte sans serif e entidades QPN escritas em *itálico*):

- Cada instância Host representa uma máquina (física ou virtual) que gera (e/ou recebe) tráfego. Tais instâncias são transformadas em elementos dos conjuntos P e Q. Um único Host é representado por uma posição ordinária p conectada à duas transições t. Estas transições conectam p à uma posição de enfileiramento q representando a fila de recebimento presente em interfaces de rede.

- As instâncias Link e Switch, responsáveis por conectar entidades Host, são transformadas em posições q e transições t. A capacidade da entidade `Link` é representada como um peso w_i, e $w_i(c)$ representa o atraso de *Link*.

- Instâncias das entidades Traffic e Flow definem o conjunto de cores C (de acordo com a quantidade de fluxos modelada), bem como a quantidade de *tokens* (de acordom com o tamanho do fluxo).

- Como o formalismo QPN não é capaz de modelar regras condicionais, as entidades Controller e Rule são transformadas em posições de enfileiramento q e transições imediatas $\tilde{W}_2$.

Vale destacar que os modelos QPN gerados aqui têm a ferramenta QPME (KOUNEV; SPINNER; MEIER, 2012) como alvo de compatibilidade para teste dos modelos gerados. E, da mesma forma que foram utilizados *templates* de geração de código (cf. Seção 3.1) para gerar aplicações SDN anteriormente, *templates* foram criados para gerar os modelos XML suportados pela ferramenta QPME.

5.3.3 Transformação NML-para-SPN

A linguagem de modelagem NML é capaz de modelar cenários SDN complexos, nos quais a existência de algumas regras de fluxo ou condições lógicas (definidas pela aplicação SDN) podem modificar o comportamento da rede (bem com o seu desempenho). Neste contexto de incerteza, SPN é um formalismo que fornece meios de modelar o desempenho de cenários SDN complexos. Esta capacidade é fácil de notar ao observar a definição formal de SPN a seguir:

Definição 3. (Stochastic Petri Net) SPN é uma tupla: $SPN = \{P, T, I, O, M_0, \Lambda\}$, onde:

1. A declaração de P, T, M_0 é a mesma da Definição 2.

2. $I : P \times T \rightarrow \{0, 1\}$ é uma matriz $n \times m$ definindo o arco direcionado de uma posição p para uma transição t:

$$\begin{cases} I(p_i, t_j) = 1 \text{ se existir um arco direcionado de } p_i \text{ para } t_j \\ I(p_i, t_j) = 0 \text{ senão} \end{cases}$$

3. $O : P \times T \rightarrow \{0, 1\}$ é uma matriz de saída $n \times m$ output definindo o arco direcionado de transições para posições:

$$\begin{cases} O(p_i, t_j) = 1 \text{ se existir um arco direcionado de } t_j \text{ para } p_i \\ O(p_i, t_j) = 0 \text{ senão} \end{cases}$$

4. $\Lambda : T \rightarrow R^-$ é uma função de disparo. Além disso, λ_{ith} é a taxa de disparo da n-ésima transiçãoonde λ_i denota a taxa de disparo de (t_i) e R^+ é o conjunto todos os números reais positivos.

Assim, considerando a Definição 3, as entidades NML foram transformadas em tuplas SPN da seguinte forma (mais uma vez, considere as entidades NML escritas na fonte sans serif e as entidades SPN escritas em *itálico*):

- Cada Host relacionado à uma instância Flow é transformado em um elemento de P.
- Os *switches* e *links* que conectam elementos Host ao longo do fluxo, são transforados em transições t. A quantidade de transições de disparo é definidas pelo número de saltos ao longo da rota da posição de origem p_n até o destino p_{n+1}.
- Toda Condition e Rule relacionadas a um Flow é transformada em uma guarda para transições específicas de T. Uma combinação de regras e condições podem gerar posições auxiliares para prover maior acurácia do comportamento do modelo.

- A quantidade de *tokens* e a distribuição de probabilidade são definidas de acordo com características das instâncias Traffic (e.g., tamanho do pacote, tamanho do fluxo).

Atualmente, a ferramenta alvo de simulação utilizada pela geração de código da solução proposta é a Mercury, já que esta suporta distribuições de tempo e condições lógicas como guardas para as transições SPN.

5.4 AVALIAÇÃO E DISCUSSÃO

Para avaliar a o que foi proposto nesse capítulo, um estudo de caso foi conduzido em uma rede SDN executando uma aplicação de balanceamento de carga. A aplicação foi executada utilizando a linguagem de modelagem NML, e, então, a funcionalidade de geração automática de código para obter as versões QPN e SPN deste cenário. Além disso, utilizando a ferramenta Mininet (OLIVEIRA et al., 2014), foi simulada a rede SDN modelada, obtendo a *baseline* para o desempenho do estudo de caso. A Figura 31 apresenta este cenário utilizando a notação NML.

Neste estudo de caso, o cenário SDN possui dois *hosts*, h1 e h2, conectados por quatro *switches* que formam duas diferentes rotas. A comunicação entre h1 e h2 deve ocorrer através de uma destas rotas. O Controller e sua aplicação Load Balancing são responsáveis por decidir qual caminho o fluxo *Flow*:A deve seguir – baseando-se na instância da regra *Rule*:A e sua condição (i.e., o tráfego deve ser balanceado se a saída do *switch*) s_2 for menor que 7Mbps). Por simplicidade, de forma arbitrária, os *links* foram definidos da seguinte forma: (s1, s2), (s1, s3), (s3, s4 e (s2, s4)) possuem 7Mbps de largura de banda, enquanto os *links* (h1, s1) e (s4, h2) transmitem pacotes em 10Mbps. Todos os *links* possuem 10μ de atraso.

Considerando este cenário, dois modelos de predição foram gerados, i.e., QPN e SPN. A partir destes modelos, dois experimentos foram realizados para verificar: (i) a acurácia da predição para cada modelo; e (ii) o tempo de simulação. O *testbed* do experimento foi um hardware equipado com um processador Intel Core i7-5500U de 2.40GHz e 8GB de memória.

5.4.1 Acurácia da Predição

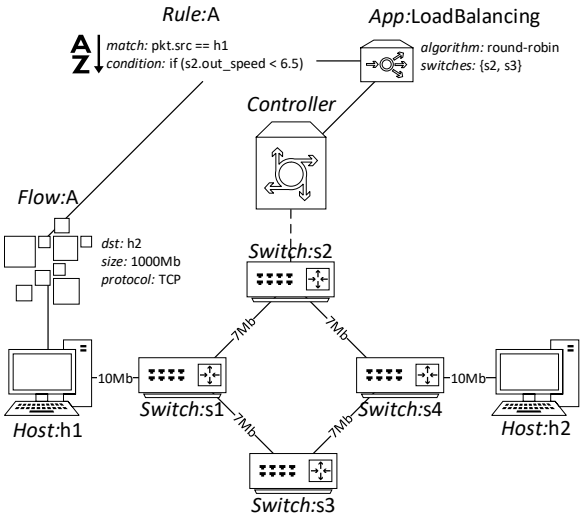
O primeiro experimento para o estudo de caso definido na avaliação visa predizer a taxa de transferência para o `Flow:A` baseado nas transformações discutidas anteriormente. A Figura 32 ilustra o resultado das transformações obtidas a partir do modelo criado na Figura 31. O leitor pode visualizar na Figura 32a (i.e., o modelo QPN gerado) que não foi considerada a condição lógica da aplicação para realizar o balanceamento de carga. Na

Figura 32b, a condição lógica da `Rule:A` foi transformada em duas guardas de transição (i.e., s1-output-s2*, s1-output-s3*) e duas posições adicionais. Para as medições, as taxas de todos os fluxos foram modeladas como exponencialmente distribuídos e os intervalos de confiança são calculados com uma significância de $\alpha = 0.05$.

A Figura 33 mostra a FDA dos resultados a partir de cada modelo comparado à *baseline* obtida a partir da simulação realizada no Mininet. Para o mesmo experimento, a Tabela 4 exibe as probabilidades de cada modelo de simulação em relação à mesma *baseline*.

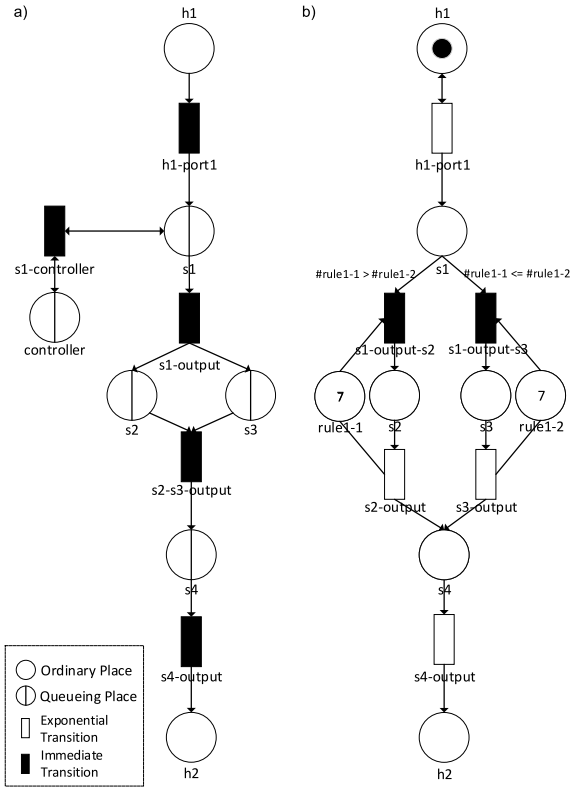
É possível perceber que a predição das taxas de transferências – analisando a taxa de chegada em `h2` – não são exatas (em relação ao *baseline*), mas essa predição ainda provê resultados com uma margem de erro razoável. Uma observação interessante a partir deste gráfico é que, apesar de o modelo QPN alcançar uma predição mais acurada quando comparada ao SPN, quando o consumo da largura de banda está no intervalo 7,0-8,0 (cf. Tabela 4), o modelo SPN prediz de forma mais realista o cenário – já que este modelo pode representar algoritmos de balanceamento de carga, e.g., *round robin*, especificados na instância *Rule:*A (cf. Figura 31). Entretanto, é preciso enfatizar que o modelo QPN proporciona uma taxa de transferência média de 7,48Mbps, o que é mais próximo do resultado da *baseline* média de 7,48Mbps obtida no Mininet (principalmente quando observa-se o modelo SPN obtendo uma média de 6,78Mbps). Nesta comparação, o modelo QPN gerado alcança uma taxa de erro de apenas 3% na predição, um resultado similar ao obtido por Rygielski, Seliuchenko e Kounev (2016b).

Figura 31 – Modelo NML para a aplicação de balanceamento de carga.



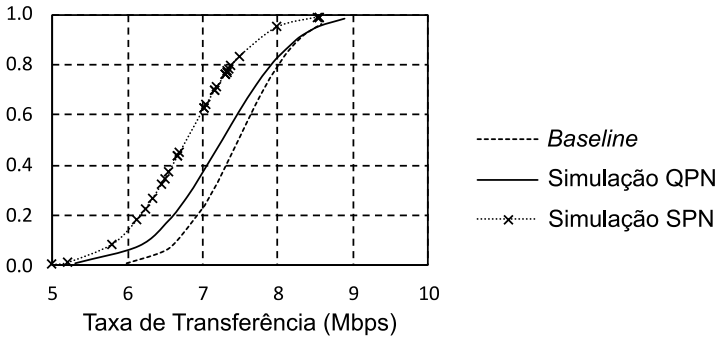
Fonte: O autor (2020).

Figura 32 – Modelos gerados. Figura a) mostra o modelo QPN resultante, enquanto a Figura b) ilustra o modelo SPN resultante.



Fonte: O autor (2020).

Figura 33 – Comparando a FDA de cada simulação à *baseline* medida no Mininet.



Fonte: O autor (2020).

Tabela 4 – Probabilidade de taxas de transferência para o fluxo modelado.

Taxa de Transferência (Mbps)	Probabilidade (Medida)	Probabilidade (QPN)	Probabilidade (SPN)
5.0-6.0	0.07	0.03	0.10
6.0-7.0	0.07	0.47	0.30
7.0-8.0	0.70	0.37	0.50
8.0-9.0	0.16	0.13	0.10
9.0-10.0	0.00	0.00	0.00

Fonte: O autor (2020).

5.4.2 Tempo de Simulação

Considerando que as transformações apresentadas aqui podem ser utilizadas no gerenciamento de SDN, dependendo do cenário, o projetista de rede pode precisar de uma predição menos acurada mas mais rápida, ao invés de uma predição precisa mas tardia. Assim, durante a execução do caso de uso ilustrado na Figura 31, foi mensurado o tempo de simulação de cada modelo.

Os resultados obtidos mostram que o projetista de rede precisa escolher entre velocidade e acurácia ao selecionar o modelo de predição baseado em simulação. Por exemplo, o modelo gerado mais acurado, i.e., QPN, levou em média 5,16 segundos para finalizar sua simulação, enquanto o SDN finalizou a simulação com uma média de 3,2 segundos. Esta diferença é, notoriamente, devido ao crescente número de *tokens* requisitados pela simulação QPN em sua análise, enquanto o modelo SPN não é sensitivo à intensidade de tráfego em sua simulação.

5.5 CONSIDERAÇÕES FINAIS

Neste Capítulo, predições baseadas em simulações foram apresentadas para demonstrar o comportamento da rede e seu desempenho. Embora os modelos abordados aqui tenham limitações conhecidas da literatura (e.g., explosão de estados e capacidade de memória), as simulações discutidas podem servir como base para melhorar o modelo de rede – incluindo a tomada de decisões em *design time* ou ainda em tempo de execução (i.e., modelos MART). Assim, este trabalho estendeu o metamodelo apresentado no Capítulo 3, propondo aqui a linguagem de modelagem NML, que agora considera aspectos de desempenho no processo de projetar redes SDN.

A partir da NML e sua capacidade de expressar características de desempenho, visando obter resultados da simulação de desempenho baseada em RdPs, utilizar transformações

entre modelos tornou-se uma alternativa a ser avaliada. Para isso, transformações entre modelos NML e duas RdPs específicas, i.e., QPN e SPN, foram definidas. Além disso, houve a avaliação experimental dos modelos gerados a partir destas transformações. A avaliação considerou o estudo de caso para uma aplicação de balanceamento de carga, comparando a acurácia das predições dos modelos QPN e SPN (gerados a partir do modelo NML para a aplicação de balanceamento de carga) com uma *baseline* obtida através da ferramenta Mininet.

Ficou constatado que apesar do modelo de simulação QPN predizer a taxa de transferência do estudo de caso com maior acurácia (i.e., taxa de erro de 3%) comparado ao modelo de simulação SPN (i.e., taxa de erro de 9%), o modelo SPN obteve tempo de simulação menor e ainda apresentou resultados mais próximos dos valores referência obtidos para a *baseline*, quando a distribuição de probabilidade do tráfego analisado foi observada, considerando o cenário avaliado (cf. Figura 33).

6 MODELOS EM TEMPO DE EXECUÇÃO PARA REDES DEFINIDAS POR SOFTWARE AUTO-ADAPTÁVEIS

Este capítulo evolui o conjunto de técnicas de modelagem apresentadas anteriormente e investiga a utilização do paradigma MART (BLAIR; BENCOMO; FRANCE, 2009) para compor um *framework* chamado MOSES, a qual aplica conceitos de MART no campo de SDN visando prover gerenciamento autonômico para estas redes. Esta aplicação visa analisar as hipóteses 2 e 3 desta tese, as quais sugerem que (i) a utilização de modelos em tempo de execução favorecem a realização do gerenciamento de SDN; e (ii) blocos funcionais adaptáveis, construídos a partir de modelos, potencializam o aproveitamento de recursos de rede. Além disso, as questões de pesquisa 2 e 4 também são respondidas, demonstrando os pontos positivos e negativos da realização do gerenciamento autonômico em SDN e mostrando as adequações de uma arquitetura que permite a implementação de aplicações e do gerenciamento autonômico em SDN.

Para verificar as hipóteses levantadas e oferecer respostas às questões de pesquisa, primeiro, foi analisado como conceito de ANM é realizado atualmente em SDN, e, então, foi feita a comparação do *framework* proposto neste trabalho, baseado em MART, sob a perspectiva de como este *framework* pode facilitar o desenvolvimento de um sistema ANM integrado para SDN. Destaca-se que a solução apresentada aqui é a primeira a aplicar conceitos de MART para ofertar modelagem de alto nível de um sistema ANM funcional para SDN, indo além de uma descrição arquitetural ou de uma funcionalidade autonômica específica.

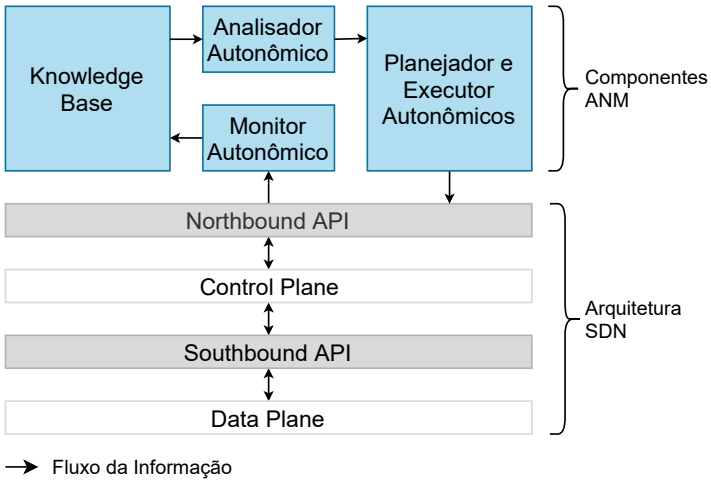
Desta forma, são duas as principais contribuições deste Capítulo. Primeiro, a apresentação da viabilidade do *framework* MOSES como uma solução para modelar ANM em SDN. Segundo, a introdução de uma nova maneira para prover propriedades *self-** nestas redes. Demonstra-se aqui que a solução proposta é adequada para: (i) predição de desempenho; e (ii) permitir ações auto-adaptativas para alcançar objetivos de rede definidos em alto nível.

O restante deste Capítulo está organizado assim: a Seção 6.1 detalha o problema do gerenciamento autonômico em SDN, anteriormente introduzido no Capítulo 1. A seção 6.2 apresenta o *framework* MOSES. Mais adiante, na seção 6.3, é descrito um caso de uso integrando as propriedades autonômicas do *framework* proposto nesta tese. A seção 6.4 apresenta os experimentos e resultados obtidos através da utilização do MOSES. Finalmente, a Seção 6.5 realiza as considerações finais deste Capítulo.

6.1 DESCRIÇÃO DO PROBLEMA

O interesse por computação autonômico iniciou há mais de uma década (HORN, 2001). Alguns anos depois, no contexto de redes, pesquisadores apresentaram suas visões sobre

Figura 34 – Arquitetura ANM baseada em SDN.



Fonte: O autor (2020).

como propriedades da computação autonômica poderiam ser aplicadas em redes para prover um sistema ANM (STRASSNER, 2004; SAMAAN; KARMOUCH, 2009; JENNINGS et al., 2007), mas, devido à rigidez e inflexibilidade presente nas redes naquele período, poucos estudos puderam ser realizados para avançar no campo de redes autonômicas. Felizmente, com a emergência do paradigma SDN, sua crescente adoção ao longo dos anos, e a necessidade por um gerenciamento autonômico por parte de operadores de rede, o cenário tem mudado e redes agora são capazes (i.e., ágeis, flexíveis) de aproveitar conceitos de autonomicidade em suas tarefas de gerenciamento.

De acordo com Samaan e Karmouch (2009), um sistema ANM deve possuir capacidades de antecipar, diagnosticar e prevenir falhar que possam ser causadas à rede subjacente. Além disso, tais capacidades devem ser alcançadas de uma maneira autonômica e independente, guiada por um conjunto de objetivos de alto nível. Essa visão é compartilhada também por Movahedi et al. (2012). Vale mencionar que trabalhos de pesquisa recentes talvez refiram-se à conceitos de ANM utilizando termos como *Cognitive Network Management* (AYOUBI et al., 2018), *Self-Driving Network* (KALMBACH et al., 2018), ou *Knowledge-Defined Networks* (MESTRES et al., 2017).

Para redes SDN, onde o controle tradicional do plano de dados foi movido para um controlador logicamente centralizado, é possível observar que tais controladores SDN desempenham um papel crucial na implementação para implementar um ANM baseado em SDN. A visão da composição de SDN com ANM (cf. Figura 34) defendida neste tese, segue a mesma linha de outros autores (STAMOU et al., 2019; NEVES et al., 2016). Nesta visão, um sistema ANM e seus componentes interagem com uma rede SDN por meio da *Northbound* API – presente na maioria dos controladores SDN (i.e., o plano de controle na Figura 34). Além disso, o sistema ANM executa ações de adaptação na rede SDN utilizando seu próprio conhecimento.

Considerando que a maioria dos estudos e métodos que visam implementar sistemas e propriedades ANM são baseadas no modelo "*Monitor, Analyze, Planejar, Execute, and Knowledge*"MAPE-K (KEPHART; CHESS, 2003), é possível definir que um ANM baseado em SDN deve consistir de: (i) uma base de conhecimento; (ii) monitoramento autonômico para verificar métricas de rede (e.g., QoS, utilização de largura de banda); (iii) análise autonômica, a qual aprende e provê diagnósticos a partir das informações colhidas no monitoramento da rede; e (iv) planejamento autonômico das ações a serem executadas e a própria execução destas ações para alcançar os objetivos de rede definidos pelos projetistas. O problema é que desenvolver e integrar estas capacidades não é uma tarefa simples. Neste contexto, soluções baseadas no paradigma MART prometem facilitar a implementação de um sistema de gerenciamento autonômico.

6.2 MODELOS EM TEMPO DE EXECUÇÃO PARA SDN

Nesta tese, visualiza-se o paradigma MART como um habilitador chave para implementar sistemas ANM para redes SDN e, assim, propõe-se o *framework MOdels at runtime for SElf-adaptive Sdn* (MOSES) – o qual aplica conceitos MART para permitir a implementação de redes SDN auto-adaptáveis. O *framework* a ser detalhado segue arquiteturas de referência baseadas no conceito de MART (ASSMANN et al., 2014; BENCOMO; BLAIR, 2009), onde sistemas baseados em MART devem sempre realizar uma interface com o sistema gerenciado.

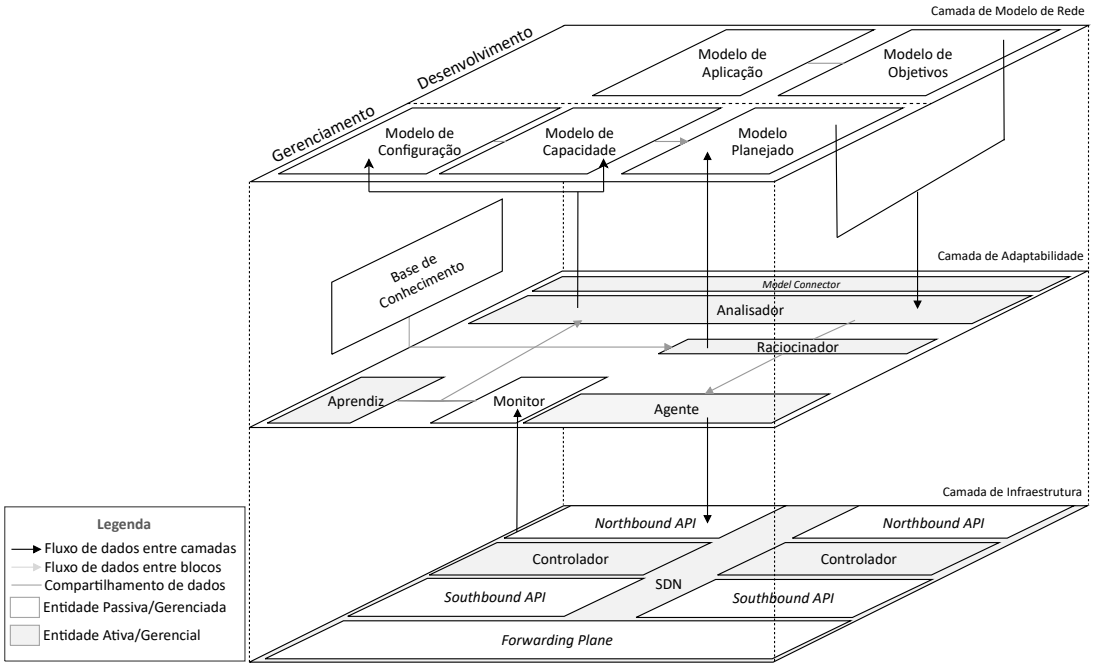
6.2.1 Visão Geral da Arquitetura

A arquitetura do MOSES (cf. Figura 35) foi projetada refletindo as três camadas propostas por (ASSMANN et al., 2014). Esta arquitetura pode ser categorizada como um hierárquica baseando-se na taxonomia de Movahedi et al. (2012), possuindo as seguintes camadas: modelo de rede, adaptabilidade e infraestrutura. Cada camada possui seus próprios componentes ou blocos funcionais, os quais devem interagir entre si e trocar informações com componentes ou blocos de outras camadas.

6.2.1.1 Camada de Modelo de Rede (CMR)

A Camada de Modelo de Rede (CMR) realiza a separação entre modelos de desenvolvimento (i.e., modelos de aplicação e objetivos) e modelos de gerenciamento (i.e., modelos de configuração, capacidade e planejamento). Desta forma, modelos de desenvolvimento podem ser diretamente definidos por um desenvolvedor ou projetista de rede, enquanto modelos de gerenciamento podem ser autonomicamente definidos.

Figura 35 – Arquitetura do MOSES.



Fonte: O autor (2020).

Os componentes da camada CMR são modelos que permitem: (i) a representação do estado da rede; e (ii) a especificação dos objetivos ou a modelagem de aplicações. O papel de cada bloco funcional desta camada é descrito a seguir:

- **Modelo de Aplicação:** permite que operadores de rede e desenvolvedores modelem aplicações de rede (e.g., firewall, balanceamento de carga, monitor de rede) resultando em código que controladores SDN executam. Mais detalhes sobre este modelo foram discutidos no Capítulo 3.

- **Modelo de Objetivos:** permite a modelagem de objetivos a serem alcançados pela rede de forma autonômica (e.g., minimizar atraso, melhorar taxa de transferência) para um cenário modelado.

- **Modelo de Configuração:** representa a configuração atual da rede (e.g., topologia, serviços em execução) e serve para verificar questões de compatibilidades (cf. Capítulo 4).

- **Modelo de Capacidades:** reflete o potencial da rede (ou capacidades) para satisfazer requisitos das aplicações e/ou objetivos modelados (e.g., QoS, largura de banda, atraso). Este modelo fornece informações para o Modelo de Objetivos.

- **Modelo Planejado:** descreve ações que devem ser realizadas pela rede. Um modelo planejado representa uma sequência de comandos ou ações descrevendo como

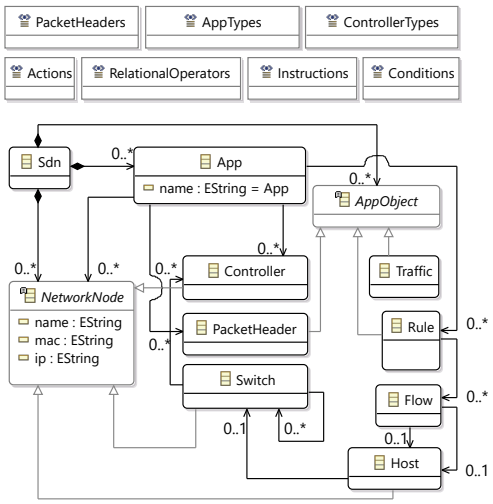
agentes das camadas mais baixas devem realizar mudanças na rede. Tais ações podem ser autonomicamente ou manualmente definidas. Na prática, os modelos desta camada são ligados entre si, embora sejam separados em termos arquiteturais. Além disso, as ações que pertencem a um modelo planejado são transformadas em código, da mesma forma descrita no Capítulo 3.

6.2.1.2 Camada de Adaptabilidade (CA)

A Camada de Adaptabilidade (CA) acomoda seis componentes, que são: Aprendiz, Analisador, Raciocinador, Agente, Monitor e Base de Conhecimento (i.e., *knowledge base*). Estes componentes realizam a interação com modelos NML (presentes na camada CMR) e contém implementações que representam conceitos MAPE-K. Neste nível, existem transformações (e.g., modelo-para-código, modelo-para-comando) partindo de modelos de alto nível para instruções de baixo nível e a partir de dados destes modelos de alto nível. Os seis componentes são detalhados a seguir:

- **Analisador:** possui dois papéis distintos na arquitetura do MOSES. Primeiro, ele categoriza a informação recebida da rede através dos modelos de configuração e capacidade. Depois, ele verifica se o estado atual da rede precisa ser alterado, considerando a base de conhecimento e os objetivos presentes no modelo planejado.
- **Raciocinador:** verifica se futuras configurações de rede devem ser definidas, considerando a base de conhecimento e o modelo de objetivos.
- **Aprendiz:** continuamente alimenta a base de conhecimento e verifica se a decisão do Raciocinador foi benéfica para a rede.
- **Agente:** desempenha mudanas na rede baseado nas instruções preparadas pelos componentes anteriores. O agente realiza mudanças através da API *Northbound* dos controladores e do código gerado a partir do modelo planejado.
- **Monitor:** consulta a rede utilizando parâmetros que sejam relevantes para atingir os objetivos modelados.
- **Base de Conhecimento (K-Base):** é o componente responsável por armazenar dados sobre o conhecimento do plano de controle, ao invés de conhecimento do domínio (já que esse está refletido nos modelos). O K-Base permite ao MOSES registrar descrições de aplicação, decisões benéficas e ações que possam ter causado erros.

Figura 36 – Versão resumida do metamodelo utilizado no modelo de aplicação.



Fonte: O autor (2020).

6.2.1.3 Camada de Infraestrutura (CI)

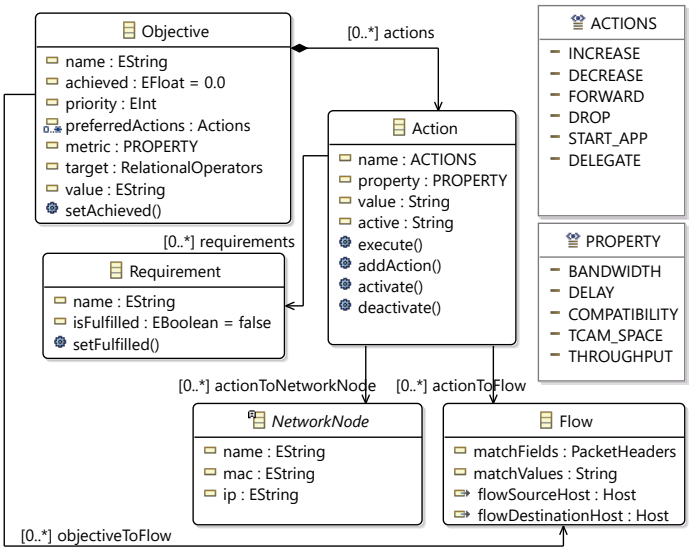
A Camada de Infraestrutura (CI) é o sistema gerenciado (de acordo com a definição do MART). No MOSES, a CI segue a arquitetura de referência definida pela ONF (2012). Esta camada é o alvo para os objetivos definidos na CMR. Além disso, a CI provê informação para a CA, de tal forma que os objetivos sejam atingidos.

6.2.2 Definindo Objetivos de Rede

Na disciplina de Engenharia de Requisitos (ER), modelos de objetivos bem-definidos são projetados em *design time* (QURESHI; PERINI, 2010). Engenheiros de software usam tais modelos para basear suas decisões de projeto. A modelagem de objetivos em soluções baseadas no MART, por outro lado, focam na avaliação de objetivos para tomada de decisão automatizada em tempo de execução. Assim, em um primeiro momento, é preciso permitir que um projetista de rede especifique os objetivos da rede que deseja. Para isso, no MOSES, foi realizada a criação de um metamodelo consistindo de classes e atributos para representar estes objetivos. O processo para descrever este metamodelo foi baseado em (LEHMANN et al., 2010).

A Figura 37 especifica entidades do metamodelo do MOSES para descrever objetivos de rede (baseada na linguagem ECore (GRONBACK, 2009)). Esta é a base para o bloco funcional chamado Modelo de Objetivos e consiste de cinco principais classes para representar um conjunto de objetivos, requisitos e ações. A classe Flow e a classe abstrata *NetworkNode* são classes reutilizadas do que foi apresentado no Capítulo 3 (com metamodelo brevemente ilustrado na Figura 36). A partir do metamodelo de objetivos

Figura 37 – Metamodelo para a camada CMR.



Fonte: O autor (2020).

(cf. Figura 37), um projetista de rede pode modelar objetivos, requisitos e as respectivas ações. Obviamente, em alguns casos, nem todos os objetivos serão alcançados pela rede subjacente. Assim, a classe Objective possui o atributo `active` para controlar quais os objetivos possíveis de serem atingidos – o valor desse atributo é atualizado pelo próprio ciclo de controle do sistema ANM gerado.

A partir dos objetivos modelados, aqueles que são possíveis de serem alcançados irão ser transformados em código de aplicação – da mesma maneira abordada n capítulo 3, porém com *templates* específicos – para interagir com o controlador SDN e a infraestrutura de rede subjacente.

6.2.3 Seleção de Ações Baseadas em Objetivos

Quando o MOSES precisa criar um Modelo Planejado (cf. Seção 6.2.1), é preciso escolher autonomicamente as ações que irão compor este modelo. Para não escolher ações apenas aleatoriamente, um modelo de seleção chamado *Goal-Action-Attribute Model* (GAAM) (SALEHIE; TAHVILDARI, 2012) foi adotado para definir: tanto quais objetivos serão buscados pela rede, quanto as ações que farão parte desta tarefa. O modelo GAAM é usado no processo de decisão em softwares auto-adaptativos, envolvendo a seleção de ações apropriadas para adaptação. No contexto em questão, para um melhor entendimento, o leitor deve estar ciente que a classe Requirement, presente no metamodelo do MOSES, refere-se ao conjunto de pré-condições definidas pelo GAAM. Dada a variabilidade de requisitos, o MOSES define um conjunto de palavras-chave e algoritmos de verificação correspondentes no processo de geração de código das ações. Tal decisão permite a simplificação do

metamodelo de objetivos.

Defende-se aqui que um modelo como GAAM (ou um modelo de comportamento similar) é necessário na modelagem de SDN auto-adaptáveis dada a capacidade desejada de um sistema ANM de descrever objetivos explicitamente que irão guiar a tomada de decisão em *runtime*. Além disso, na maior parte do tempo, um ANM baseado em SDN deve precisar lidar com diversos objetivos definidos pelos projetistas de rede. O processo de decisão deve levar em conta todos estes objetivos, coordenando-os e resolvendo conflitos.

Assim, inicialmente (sem conhecimento prévio ou sem a base de experiências passadas que farão parte da base de conhecimento), o MOSES seleciona as ações que irão fazer parte do Modelo Planejado considerando um conjunto G de objetivos modelados no Modelo de Objetivos, denotado aqui por $G = \{g_i, i = 1..m\}$, onde cada g_i possui seus requisitos relacionados continuamente avaliados por uma função de ativação $f_i(.)$ que determina se g_i está ativado ou não. Instâncias de Objective relacionadas à instâncias de Action com o atributo `active` igual a `true` participam do mecanismo de seleção de ação. Note que um elemento g_i pode ter todas as suas ações ativas, mas não ser atingido completamente. Para representar este cenário, o atributo `achieved` da classe Objective possui um valor de porcentagem para g_i. Como o GAAM não especifica nenhum processo de monitoramento para checar o status das propriedades, na especificação do MOSES ocorre a utilização dos blocos funcionais **Monitor** e **Analisador** (cf. Seção 6.2.1) para calcular a porcentagem de alcance de cada g_i. Elementos de G também possuem o atributo `priority` que irá impactar na influência destes elementos ao participarem da seleção de ações. As ações de adaptação, ilustradas na Figura 37 pela classe Action e denotadas aqui pelo conjunto $AC = \{ac_i, i = 1..n\}$, serão executadas pelo bloco funcional Agente e podem estar relacionadas à instâncias da classe `Requirement`, definida formalmente pelo conjunto $R = \{r_i, i = 1..n\}$. R refere-se ao conjunto de pré-condições necessárias para realizar uma ação ac_i que seja parte do Modelo Planejado. Além disso, obviamente, elementos de AC são relacionados aos elementos de G. Neste relacionamento existe uma ordem de importância, i.e., as primeiras ações na lista chamada `preferredActions` possuem precedência em relação às outras. Depois de formalizar estas entidades, é possível observar os relacionamentos possíveis entre elas na Figura 37.

Considerando todos os conjuntos e relacionamentos descritos até agora, o MOSES pode, então realizar seu processo de seleção de ações. Comparado às diretrizes do GAAM, a implementação do MOSES relaxou o aspecto de seleção de ação, já que possui o suporte de um algoritmo de aprendizagem de máquina. Ou seja, enquanto o GAAM define cinco passos para selecionar ações, o MOSES desempenha este processo em quatro passos. Assim, a sua seleção de ação ocorre na seguinte sequência:

1. *Análise de Requisitos:* neste passo, o mecanismo utilizado avalia os objetivos modelados, as ações e seus requisitos para determinar qual deles está satisfeito ou não.

2. *Análise de Capacidade:* aqui, é verificado se valores definidos para o atributo `property` em instâncias de Action podem ser alcançados pela rede subjacente. Tal verificação ocorre por meio da comparação entre os modelos de capacidade e os valores especificados para tais propriedades. Se alguma ação possui um valor de propriedade fora das capacidades da rede, esta ação não será levada em consideração no processo de seleção. Este passo é um outro filtro para evitar ações propensas à erros no Modelo Planejado.

3. *Elencando Votos:* Objetivos ativos possuem suas listas de ações preferidas. Este passo de elencar votos analisa a ordem da lista `preferredActions` ou utiliza o histórico das últimas ações realizadas. A saída é o conjunto $\hat{AC}$ contendo as ações em ordem de execução para cada objetivo.

4. *Agregação de Votos:* Para definir as preferências por ações, o MOSES aplica um esquema de votação baseado no mecanismo de contagem proposto por Straffin (1980) (i.e., cada objetivo possui sua ordem de preferência para executar ações). Neste mecanismo, candidatos (i.e., ações) recebem um número de pontos igual ao próprio número de candidatos participando da seleção. Por exemplo, se dois objetivos possuem um total de cinco ações, cada ação pode receber até cinco pontos – de acordo com as suas posições na ordem de preferência de cada objetivo. Este mecanismo é necessário para prover a adaptabilidade, considerando que a lista de ações preferidas por cada objetivo pode mudar com o passar do tempo.

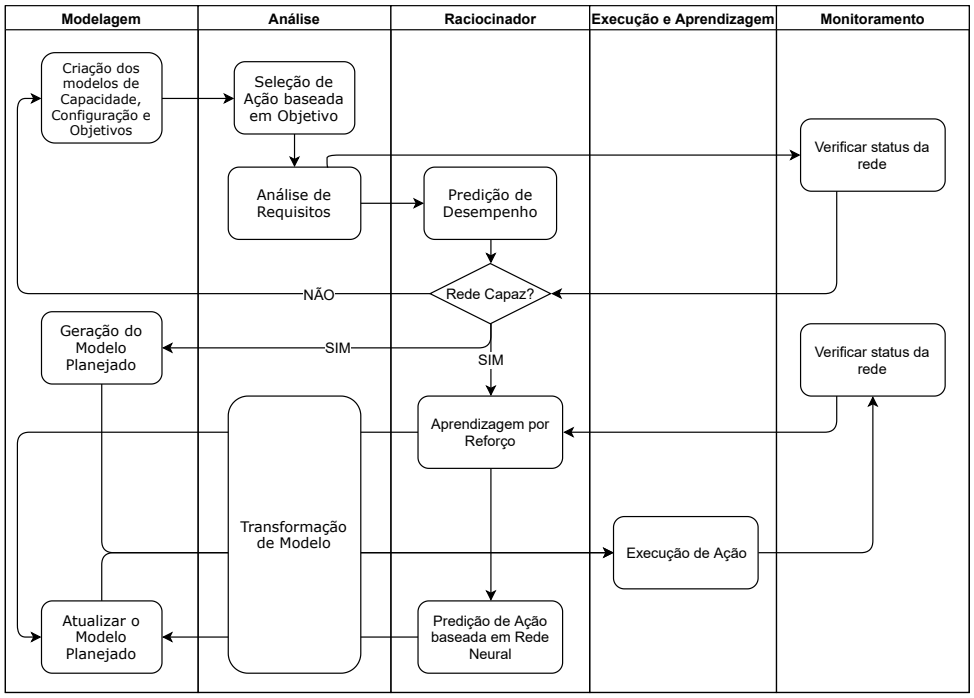
A implementação flexível do modelo GAAM torna o MOSES capaz de ativar ou desativar ações em *runtime*. Por exemplo, se um requisito não pode ser satisfeito em um dado instante, o MOSES irá desativar as ações relacionadas a este requisito. Além disso, o MOSES pode criar instâncias de Action baseado nas predições do algoritmo de aprendizagem de máquina que será apresentado neste Capítulo.

6.2.4 Funcionamento do Raciocinador

O bloco **Raciocinador** tem o papel de verificar as possibilidades de ações a serem tomadas para alcançar os objetivos da rede, principalmente quando o **Monitor** do MOSES identifica que mudanças ocorreram e o Modelo de Objetivos não está sendo alcançado. Ao tomar decisões, o **Raciocinador** reflete as ações no Modelo Planejado, que por sua vez transforma suas entidades em comandos de ações a serem realizadas na camada CI. Se o bloco **Raciocinador** decide realizar alguma ação na rede, o **Monitor** também irá receber tal ação e registrá-la na base de conhecimento.

No bloco **Raciocinador**, é preciso verificar *se*, *o que* e *como* objetivos e suas ações podem ser alcançados, considerando o estado da rede e seu desempenho. Para realizar

Figura 38 – Passo-a-passo da execução de uma instância do MOSES.



Fonte: O autor (2020).

este raciocínio, o MOSES tenta (i) predizer o desempenho da rede (e.g., largura de banda disponível, atraso) e (ii) analisa o impacto de uma ação executada na rede, aprendendo com ela. A próximas subseções definem as técnicas de simulação e aprendizagem adotadas no MOSES para permitir este raciocínio. Entretanto, vale mencionar que, inicialmente, quando a base de conhecimento da rede está vazia ou sem informação suficiente para basear uma decisão acurada, o operador de rede modela um número mínimo de ações semânticas, requisitos e fluxos alvo para alcançar o objetivo modelo (que, obviamente, também deve estar modelado). Além disso, neste ponto, MOSES pode apenas predizer as métricas de desempenho baseado nos modelos de Configuração e Capacidade. Mesmo embora um sistema ANM implementado com o MOSES utilize algoritmos de aprendizagem de máquina no processo de tomada de decisão, estes algoritmos precisam da entrada sobre requisitos e ações necessárias para alcançar o objetivo modelado e guiar o processo de aprendizagem.

A Figura 38 ilustra um diagrama de fluxo que ajuda a entender como o processo de raciocínio ocorre e quais blocos funcionais ele envolve, demonstrando os principais passos dados por uma instância do MOSES.

6.2.5 Predição de Desempenho

Na implementação do MOSES, o **Raciocinador** faz uso das transformações de modelos apresentadas no capítulo 5. Nestas transformações, as informações presentes nos modelos de Capacidade e Configuração foram utilizadas para gerar os modelos SPN e QPN que discutidos anteriormente.

O resultado das simulações baseadas nestes modelos é usada para obter uma ideia do desempenho esperado da rede de acordo com o que está modelado. Caso o resultado da simulação mostre que a rede subjacente não é capaz de alcançar os objetivos modelados, o ciclo do MOSES não deve ser executado.

6.2.6 Seleção de Ação baseada em Aprendizagem de Máquina

Embora os modelos baseados em RdPs possam predizer algumas métricas de desempenho, modelar e criar transformações de modelos para representar algumas ações SDN seria extremamente custoso. Por exemplo, caso o Modelo Planejado precisasse realizar uma ação de delegação de fluxo (cf. Capítulo 4) para evitar um gargalo na rede, se somente modelos SPN e QPN fossem utilizados, o MOSES precisaria: (i) descrever o formalismo de transformação do modelo de objetivo de alto nível para estes modelos (i.e., representação de uma delegação de fluxo – e outras ações não triviais – em modelos QPN e SPN); (ii) simular os modelos gerados; e (iii) verificar as predições de tais modelos. Considerando que seria preciso definir um formalismo para cada ação de alto nível, é possível alegar que: utilizar a infraestrutura do MOSES – inserindo um algoritmo de aprendizagem de máquina em seu raciocinador – torna o *framework* capaz de predizer capacidades da rede e o impacto de suas ações de forma mais simples.

Neste contexto, a análise de ações por meio de aprendizagem de máquina é importante para decidir: (i) quais ações devem estar no Modelo Planejado e na lista de `preferredActions`; e (ii) se a execução de uma ação será benéfica para alcançar o objetivo modelado, levando em consideração o estado da rede. Tal análise seria limitada ou mais complexa se **apenas** modelos baseados em RdPs fossem utilizados. Assim, enquanto o MOSES utiliza as transformações de modelos definidas anteriormente pra predizer o desempenho da rede (auxiliando, por exemplo, na verificação de requisitos), o algoritmo de aprendizagem de máquina deve predizer as ações que serão adicionadas no Modelo Planejado, e como estas ações farão parte do processo baseado no modelo GAAM.

Para implementar a seleção de ações no **Raciocinador**, foram utilizadas Redes Neurais (RNs) e *Reinforcement Learning* (RL). Estas técnicas utilizadas separadamente, não seriam úteis no cenário em questão devido às características de dados rotulados escassos e à dinâmica dos eventos de rede (e.g., estados não estacionários, espaço de estados desbalanceado e contínuo). entretanto, a combinação de RN e com RL – um conceito co-

nhecido como *Deep Reinforcement Learning* (DRL) – provê respostas rápidas e precisas, aproveitando o conhecimento prévio (GHAZIKHANI; MONSEFI; YAZDI, 2014) adquirido de acordo com a execução da rede. Assim, o algoritmo DRL seleciona as ações possivelmente benéficas em um certo estado da rede de acordo com a experiência prévia ao colocá-las na lista de ações a serem utilizadas pelo modelo GAAM. O algoritmo DRL realiza a predição primeiramente utilizando informações da K-Base.

A seleção da ação desempenhada no raciocinador complementa a seleção de ações do modelo GAAM. A necessidade destes dois modelos é para garantir que as ações selecionadas irão ocorrer não apenas considerando a otimização de alguma métrica de rede (obtida com o algoritmo DRL), mas também considerando os requisitos e a semântica presentes no Modelo Planejado e no Modelo de Objetivos.

O processo de seleção de ações discutido nesta seção e definido como p, pode ser modelado como um Processo de Decisão Markoviano, i.e., *Markov Decision Process* (MDP). Neste processo, define-se que p está em algum estado s (i.e., estado da rede) no instante de tempo t. Depois da execução de uma ação a, p vai para o instante de tempo t' e, de acordo com as mudanças de desempenho que ocorram, move-se para o estado s', fornecendo ao tomador de decisão uma recompensa que definida como $R_a(s, s')$. Assim, o MDP é uma 5-upla $< S, A, T, R, \gamma >$ – onde γ é um fator de desconto que varia entre 0 e 1 para as recompensas em cada instante t, representando a importância das recompensas futuras (e.g., um γ com valor 0 valoriza mais as recompensas recentes, enquanto um γ próximo ou igual a 1 valoriza mais as primeiras recompensas) (RUSSELL; NORVIG, 2016). Vale observar que, no presente escopo (i.e., modelos de objetivos para redes SDN), S é um conjunto que pode alcançar um grande número de elementos, e, juntamente com o conjunto de ações A, pode abrigar valores contínuos. Além disso, as probabilidades ou recompensas de R, inicialmente, são desconhecidas.

O cenário formalizado acima serve para enfatizar a escolha por um algoritmo de DRL que realize a seleção de ações a serem executadas na rede de forma autonômica e baseada nos modelos que compõem o *framework* MOSES. O algoritmo DRL, então, tem o papel de definir comportamento do bloco funcional **Raciocinador** durante o funcionamento da rede e dos modelos em tempo de execução.

Ao utilizar RL com NN (i.e., as técnicas de aprendizagem de máquina por trás de algoritmos de DRL), ao invés de buscar por uma solução ótima em todos os estados aprendidos, o algoritmo *Deep Reinforcement Learning* (DRL) realiza tentativas randômicas e, evolutivamente, ajusta os pesos das próximas tentativas. Este ajuste, na implementação, considera se a ação escolhida tornou o estado da rede mais próximo da implementação do Modelo Planejado.

Desta maneira, o algoritmo DRL utiliza um agente para escolher e, então, executar as ações na rede (continuamente monitoradas pelo bloco funcional **Monitor**). Assim, o algoritmo obtém o *feedback* para a ação escolhida, e utiliza o retorno para ajustar o

conjunto R.

Algoritmo 3: Deep Reinforcement Learning

```
   Input: Recupera os dados de K-Base; se houverem
   Input: Inicializa pares de ações-valor com pesos aleatórios
   Input: Inicializa o estado s_t
   Input: Inicializa o objetivo g
   Data: alcance = 0
 1 begin
 2     Function DRL():
 3         while alcance > g do
 4             for step = 0; step < taxa_de_aprendizagem; step++ do
 5                 Selecionar ação a_t em s_t
 6                 Recuperar parâmetros da rede em s_t
 7                 Selecionar um parâmetro param_t
 8                 ε = ε − (step/taxa_de_aprendizagem) ∗ ε
 9                 Executar a ação a_t com param_t na rede
10                 Observar o novo estado s_{t+1}
11                 Calcular recompensa r_t
12                 Armazenar a experiência na K-Base
13                 Atualizar o param_t no Modelo Planejado
14                 Amostragem de n transições aleatórias da K-Base
15                 Calcular fator de desconto da recompensa r_t
16                 Atualizar a transição t_t com o fator de desconto
17                 Treinar a Q-Network com o novo valor de desconto da transição t_t e
                   os novos valores de s_t + 1 e a_t
18                 alcance = comparar(melhor_t, pior_t)
19             end
20         end
21 end
```

O Algoritmo 3 mostra o pseudo-código do método DRL utilizado no MOSES, baseado no trabalho de Mnih et al. (2013). O algoritmo implementado no **Raciocinador** do MOSES começa recuperando valores (caso existam) da K-Base. Em todo caso, a experiência prévia da K-Base servirá apenas para melhorar a qualidade do treinamento da *Q-Network*. Ao entrar no instante t, o algoritmo seleciona uma ação a_t – utilizando uma política gulosa para escolher a ação com o valor de Q, ou seja, os casos em que a combinação entre s e a provocaram maior recompensa. Caso não tenha nenhuma recompensa, essa ação é escolhida aleatoriamente dentro das ações possíveis de serem executadas em s_t. Após a escolha, a ação a_t é executada na rede, observando o novo estado s_{t+1} e calculando a recompensa para o novo par (s_t, a_t). Os parâmetros envolvidos na ação e presentes no Modelo Planejado são atualizados e, então, realiza-se o ajuste dos pesos da *Q-Network* a partir das amostra, obtendo pesos balanceados para cada passo do treinamento.

Aqui, destaca-se que um desafio de pesquisa existente é avaliar diferentes parâmetros ou mesmo funções de ajuste visando alcançar o gerenciamento autonômico da rede com

a menor perda de desempenho possível, i.e., neste caso, reduzir a curva ou tempo de aprendizado do algoritmo presente no raciocinador.

6.2.7 Execução

Aliado ao processo de seleção de ações, o componente **Agente** torna-se responsável por realizar as ações que irão *de facto* modificar regras da rede e/ou políticas de encaminhamento de fluxos. Nesta execução, uma transformação de modelos de alto nível em instruções de baixo nível ocorre. Por exemplo, ao analisar o Algoritmo 3, a linha 9 refere-se à definição da ação que será executada na rede, neste passo, cria-se uma entidade no Modelo Planejado, e, então, a transformação deste modelo em código acontece. Considerando as ações definidas no Modelo Planejado, o **Analisador** irá transformar as ações modeladas em instruções de baixo nível (que interagem com o controlador SDN) – utilizando *templates* dinâmicos semelhantes aos foram introduzidos na seção 3.1.

6.2.8 Base de Conhecimento e o bloco Aprendiz

O MOSES estabelece uma base de conhecimento para armazenar informações sobre ações e resultados de mudanças na rede. Para oferecer essa base de conhecimento e prover informação útil aos processos de análise e seleção de ações, é preciso definir um formalismo para descrever a informação que é armazenada. Desta forma, primeiramente, verifica-se que três tipos de informações são relevantes na auto-adaptação aos eventos da rede na realização de seus objetivos: (i) identificação de eventos de rede (e.g., novos fluxos, sobrecarga de rotas); (ii) identificação das ações autonomicamente executadas; e (iii) métricas da rede. Em seguida, é necessário armazenar tal informação de forma que possa ser utilizada no treinamento de um algoritmo de aprendizagem de máquina e que ofereça suporte ao processo de tomada de decisão. No MOSES, não há uma definição explícita sobre onde estas informações devem ser armazenadas. Porém, para permitir um uso prático e a realização dos experimentos que serão demonstrados posteriormente, nesta tese, optou-se por um sistema de banco de dados não-relacional (ainda que um banco de dados relacional pudesse ser usado, com possível perda de desempenho) capaz de representar a base de conhecimento.

O bloco **Aprendiz**, neste contexto, age ao lado da base de conhecimento (cf. Figura 35) verificando se uma ação previamente executada foi positiva para o alcance dos objetivos de rede. Este bloco precisa verificar, a partir da comparação com a informação armazenada, se uma ação executada foi benéfica. Por exemplo, considere o objetivo de alcançar taxas de atraso limite, ilustrado na Figura 39. Se o processo de seleção define que uma ação de Delegação de Fluxo (cf. Seção 4.3) deve ser realizada, o bloco **Aprendiz** irá verificar se

Tabela 5 – Descrição dos atributos da base de conhecimento.

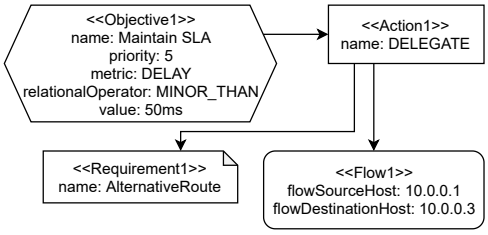
Nome do Atributo	Descrição	Tipo de Dado
ETH Origem	Identificador Ethernet do fluxo de origem.	Numérico
Porta Origem	Número da Porta utilizada no fluxo de origem.	Numérico
ETH Destino	Identificador Ethernet do fluxo de destino.	Numérico
ETH Porta	Número da Porta utilizada no fluxo de destino.	Numérico
Rota ID	Identificador para cada rota computada utilizada pelos fluxos de rede.	Numério
Versão OpenFlow	Versão do protocolo OpenFlow requisitada pelo fluxo.	Texto
Bandwidth Disponível	Largura de banda disponível na rota no momento anterior à seleção da ação - em Mbps.	Numérico
Bandwidth Máxima	Vazão agregada total computada para a rota em questão.	Numérico
RTT	Duração do *Round-trip time* (RTT) ao mensurar o atraso de rede para o fluxo em questão - em milissegundos.	Numérico
Perda de Pacote	Porcentagem de perda de pacote no momento anterior à seleção da ação.	Numérico
Action ID	Identificador para ação executada a partir da lista de ações preferidas (cf. Section 6.2.1).	Numérico
Objective ID	Identificador para os objetivos modelados.	Numérico
Taxa de Alcance	Porcentagem do quanto o objetivo em questão foi atingido ao executar determinada ação.	Numérico

Fonte: O autor (2020).

após a tal ação, o que pode ser visto a partir da taxa de alcance (i.e., coluna Alcance) aumentou ou diminuiu.

A Tabela 5 descreve o formato do conjunto de dados presente na K-Base. É este conjunto que é utilizado para treinar o algoritmo DRL. Vale ressaltar que, embora os atributos da K-Base atual tenham ajudado a realizar a solução autonômica aqui proposta, a definição de quais métricas coletar, ou seja, quais características o conjunto de dados deve ter para analisar o estado da rede – visando treinar um algoritmo de aprendizagem de máquina para desempenhar predição e tomada de decisão – ainda é um desafio em aberto (JAHROMI; HINES; DELANEV, 2018).

Figura 39 – Modelo de Objetivo de alto nível.



Fonte: O autor (2020).

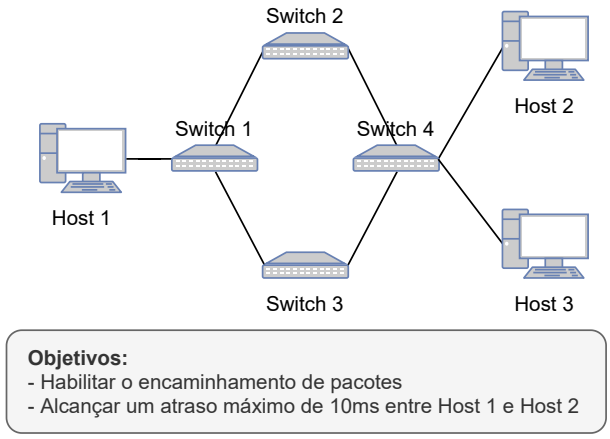
6.2.9 Monitoramento e Análise

Na camada CA, três principais blocos são responsáveis por: (i) monitorar eventos da rede; (ii) analisar os eventos da rede; e (iii) prover ações razoáveis para atingir os objetivos modelados na CMR. Para monitorar a rede e verificar se os objetivos estão sendo alcançados, o bloco **Monitor** continuamente consulta os recursos de rede utilizados pelo bloco **Agente**. Por exemplo, considere o Modelo de Objetivo ilustrado na Figura 39 – criada com a sintaxe da NML. Neste objetivo, há a intenção de alcançar taxas de atraso menores que 50ms. Assim, o bloco de monitoramento irá continuamente consultar a rede (por meio de testes de desempenho, por exemplo) para verificar se estes atingem o objetivo. Aqui, destaca-se um dos primeiros desafios para prover comportamento autonômico em SDN que é o *trade-off* entre acurácia e sobrecarga, porque se um projetista de rede precisa ter uma alta precisão na verificação do atraso, por exemplo, maior tráfego de monitoramento será gerado, o que irá aumentar a sobrecarga na rede.

Visando minimizar a sobrecarga, é possível incrementar o período de tempo no qual o **Monitor** irá verificar a rede. Obviamente, tal período irá depender das características do tráfego. Se a rede possui um comportamento de tráfego constante, ações de monitoramento podem utilizar menor granularidade. Este problema foi investigado por Su et al. (2015), Tahaei et al. (2018). Mesmo embora existam questões em aberto na área de monitoramento de fluxo, especialmente neste balanceamento entre sobrecarga e acurácia, a contribuição deste Capítulo está mais relacionada ao comportamento do monitoramento autonômico e como este monitoramento se altera de acordo com os objetivos definidos na CMR.

Por exemplo, considere que o Modelo de Objetivo precisa alcançar um atraso máximo de 50ms entre os *hosts* `h1` e `h2` em uma topologia arbitrária (cf. Figura 40). Neste caso, o MOSES irá coletar estatísticas de rede para verificar, especificamente, o atraso da rede. Isso é feito por meio da transformação a partir dos objetivos modelados em consultas de rede passando-as para a *Southbound* API. Então, o bloco **Monitor** recebe as respostas e as envia para os blocos **Analisador** e **Aprendiz** (caso seja preciso verificar o que ocorreu após alguma ação de adaptação).

Figura 40 – Esquema mínimo do caso de uso.



Fonte: O autor (2020).

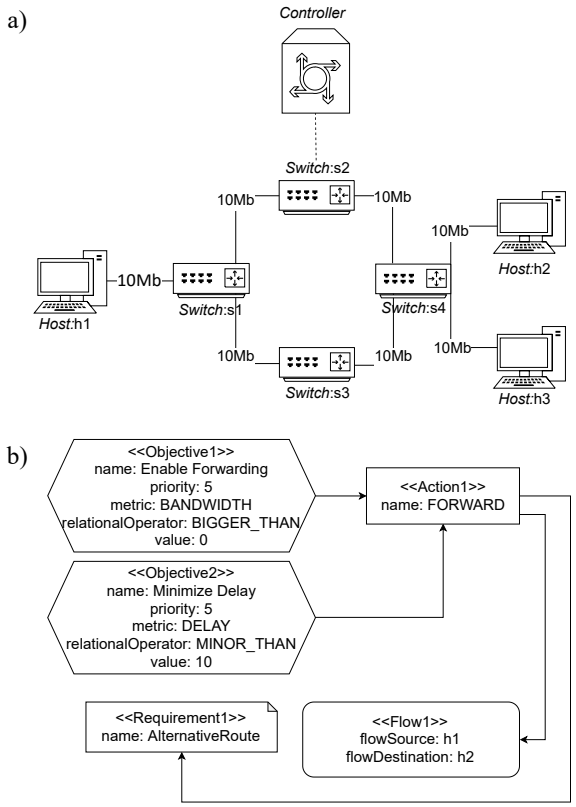
6.3 CASO DE USO

Para demonstrar o uso prático do MOSES, o seguinte cenário foi utilizado: considere uma simples topologia de rede na qual os objetivos são: (i) permitir o encaminhamento de pacotes; (ii) alcançar taxas máximas de atraso de 10ms na comunicação entre dois *hosts*. A topologia, ilustrada na Figura 40, consiste de três *hosts* (i.e., `h1`, `h2` e `h3`) conectados através de quatro *switches* (i.e., `s1`, `s2`, `s3`, e `s4`) que formam duas diferentes rotas. A comunicação entre estes *hosts* deve ocorrer através de um destes caminhos. Para simplificar o cenário, todos os *links* foram modelados com 10Mbps de largura de banda e atraso de 10 unidades de tempo.

Para implementar a camada CMR e permitir a criação de seus modelos, o GMF – *framework* para modelar e criar transformações de modelos (KOLOVOS et al., 2009) – foi utilizado. No GMF, foram implementados metamodelos e transformações de modelos (i.e., modelo-para-texto) descritos pela arquitetura do MOSES. O GMF suporta a geração de um editor gráfico, o qual foi utilizado em tempo de execução, i.e., *runtime*, para atualizar os modelos e gerenciar a rede subjacente. Este passo permite que operadores de rede utilizem um editor gráfico e a sintaxe concreta da linguagem NML para criar os modelos de Objetivos, Capacidades e Configuração do caso de uso em questão, como ilustrado na Figura 41.

Como discutido na Seção 6.2.1, a partir dos modelos criados na linguagem NML, foi utilizado o processo de transformação de modelos para instanciar os componentes da camada CA (i.e., analisador, raciocinador, aprendiz, agente, monitor e k-base). Para permitir os princípios de MART em na implementação em discussão, foi preciso verificar o formato dos modelos GMF para atualizar as instâncias destes modelos GMF após a execução de ações na rede. Neste caso de uso, quando o bloco Raciocinador decide por executar uma ação, é preciso atualizar arquivos XML que representam os modelos NML.

Figura 41 – Caso de uso implementado na NML.



Fonte: O autor (2020).

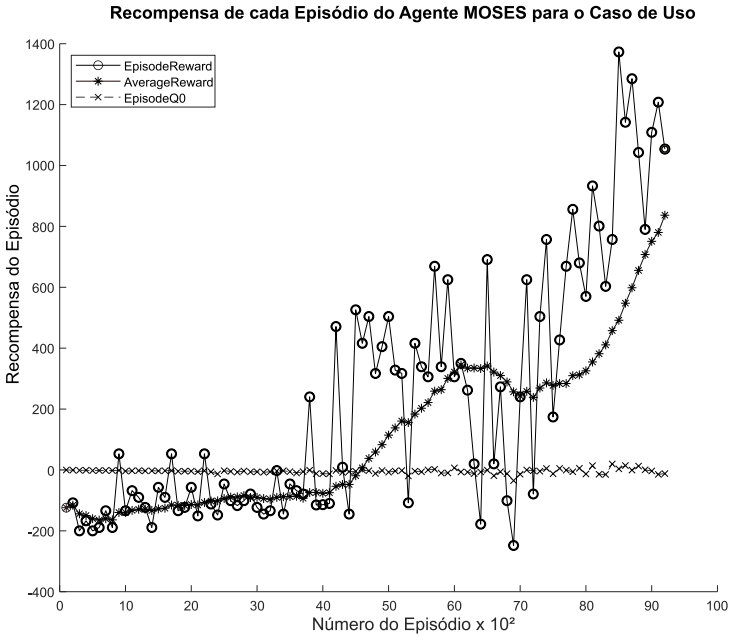
A atualização pode consistir da adição de novos elementos e ações à estes modelos.

6.4 AVALIAÇÃO E DISCUSSÃO

6.4.1 Impacto da Seleção de Ação

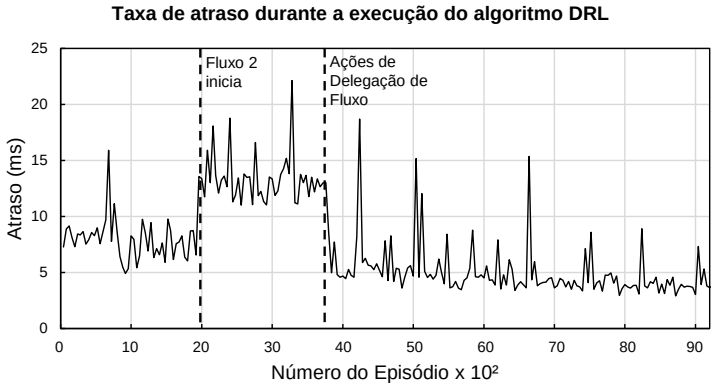
Para verificar a transformação de modelos de alto nível e o raciocínio realizado no processo de seleção de ação, i.e., execução do Algoritmo 3, foi introduzido tráfego de congestão do *host* `h3` para o *host* `h1` (cf. Fluxo 2 na Figura 43). O mecanismo de seleção de ação (iniciado com o modelo GAAM) escolhe a possível melhor ação para alcançar um estado planejado. A Figura 42 ilustra o processo de aprendizado para o caso de uso em questão. O algoritmo DRL foi instanciado com uma taxa inicial de aprendizagem de 0,00001 e função de ativação *Rectified Liner Unit* (ReLU) para todos os neurônios – a taxa inicial foi definida para evitar mínimos locais e a função ReLu foi escolhida baseando-se em evidências da literatura que mostram a sua adequação para dados esparsos (SCHMIDHUBER, 2015). O fator de desconto foi definido para $\gamma = 0.99$ (valorizando recompensas anteriores), e a rede alvo é atualizada a cada 20 passos. A função de recompensa mensurou o quanto o atraso aumentou ou diminuiu após cada decisão de seleção de ação, que representa

Figura 42 – Taxa de aprendizagem para o caso de uso.



Fonte: O autor (2020).

Figura 43 – Atraso durante o experimento.



Fonte: O autor (2020).

um episódio. Além disso, é preciso destacar que o espaço de ação envolvido na fase de treinamento abrange todas as ações possíveis do metamodelo do MOSES.

É possível perceber, na Figura 42, que a recompensa inicial do agente baseado em DRL é negativa, mas com tendência crescente conforme as ações possíveis são escolhidas e executadas na rede. De acordo com o resultado do experimento, os picos do gráfico são alcançados toda vez que o agente escolhe e executa uma ação de **delegação de fluxo**, o que descongestiona as rotas e, consequentemente, com a melhoria nas métricas da rede, uma maior recompensa é gerada para o conjunto R.

Este cenário fica evidente ao analisar a Figura 43, que mostra como o algoritmo começa a reagir com a sobrecarga na rede (por volta do episódio número 40). Neste mesmo

episódio, a rede começa a apresentar taxas de atraso maiores do que as que foram implementadas no Modelo de Objetivo. Assim, esperava-se que o algoritmo DRL começasse a dar maiores recompensas ao utilizar a ação de delegação de fluxo.

6.5 CONSIDERAÇÕES FINAIS

Neste capítulo, foi proposto o *framework* chamado MOSES para permitir um gerenciamento autonômico em redes SDN. Com a utilização de conceitos do paradigma MART em sua definição, o MOSES permite que um operador ou projetista de rede defina objetivos em alto nível de abstração e que serão alcançados por meio de ações autonômicas. A criação e seleção de tais ações é realizada a partir dos modelos de alto nível e de medições realizadas na rede. Com a transformação de modelos e o uso de um algoritmo de aprendizagem de máquina, instâncias do MOSES podem predizer o desempenho que será alcançado e quais ações irão impactar positivamente o comportamento da rede visando alcançar os objetivos de alto nível modelados.

A avaliação do MOSES, feita por meio de experimentos quantitativos, mostrou que a solução aqui proposta pôde predizer o desempenho do caso de uso analisado com uma taxa de erro de apenas 4%. Além disso, o sucesso em selecionar ações e em predizer o impacto destas no comportamento da rede melhorou o atraso da rede em até 25% no cenário avaliado. Assim, com base nestes resultados, foi possível demonstrar a adequabilidade do MOSES como um *framework* para implementar um sistema ANM para SDN.

Tanto a proposta arquitetural quanto os experimentos colocam o MOSES como um dos primeiros *frameworks* autonômicos para SDN. Este capítulo, então, responde não apenas às questões de pesquisa 2 e 4 (cf. Seção 1.2), mas também confirma as hipóteses 2 e 3, e aponta trabalhos futuros que podem ser desenvolvidos para superar barreiras existentes na generalização de uma solução auto-adaptativa para redes definidas por software.

7 CONCLUSÃO

Apesar de possuir mais de uma década desde o seu surgimento, o paradigma SDN McKeown et al. (2008) ainda não teve seu potencial totalmente alcançado. Por outro lado, novas tecnologias de rede, como 5G, demandam por soluções autonômicas que podem ser criadas utilizando redes SDN. A complexidade do desenvolvimento de aplicações ANM baseadas em SDN é que ainda considera-se um entrave para a sua adoção.

Neste contexto, a elaboração de propostas que integrem as facilidades do gerenciamento autonômico junto com a flexibilidade das redes SDN precisam ser realizadas. Consequentemente, essa tese teve como objetivo explorar a utilização do conceito de MART como solução para implementar um sistema ANM baseado em redes SDN. Em resumo, esta tese se propôs a resolver dois problemas: (i) o baixo nível de abstração e complexidade para implementar um sistema ANM; e (ii) a má utilização de recursos em redes SDN – principalmente quando considera-se cenários heterogêneos ou de sobrecarga de recursos isolados.

A elaboração do *framework* MOSES considerou métricas de desempenho na definição de objetivos de rede a serem alcançados e nas ações de auto-adaptação. Isso fica refletido na arquitetura do MOSES, a qual foi definida com três camadas, sendo uma delas a Camada de Adaptabilidade, composta por diversos blocos funcionais que interagem entre si para alcançar objetivos definidos em uma outra camada, chamada Camada de Modelo de Rede. A organização dos blocos funcionais e a forma que o fluxo de dados ocorre é que permitiu que o MOSES fosse aplicado em um estudo de caso prático para o gerenciamento autonômico de uma rede

Nas avaliações realizadas, o MOSES e as extensões parciais que o precederam, possibilitaram a resolução dos problemas destacados e demonstraram que a utilização de modelos, principalmente em tempo de execução, mostrou-se eficaz na garantia de benefícios não só de projeto, mas também de desempenho ao implementar um sistema ANM para SDN. Esta eficácia comprova as hipóteses levantada por esta tese, onde uma abordagem baseada em modelos de fato não só permite o gerenciamento de redes SDN mas também proporciona ganhos funcionais, como a realização de um gerenciamento autonômico e o aproveitamento de recursos de rede em ambientes heterogêneos.

7.1 CONTRIBUIÇÕES

Em decorrência do projeto de pesquisa que baseou o desenvolvimento desta tese, as seguintes contribuições científicas foram feitas:

7.1.1 Publicações

1. Lopes, Felipe A.; et al., Automatically Generated Simulations for Predicting Software-Defined Networking Performance. In: 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, Natal. 2018 IEEE Symposium on Computers and Communications (ISCC), 2018. (Qualis: A2)

2. Lopes, Felipe A.; et al., Model-based flow delegation for improving SDN infrastructure compatibility. In: NOMS 2018 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, Taipei. NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018. (Qualis: A2)

3. Lopes, Felipe A.; Bauer, Robert; Fernandes, Stenio. Capability-Aware SDN Application Models: Dealing with Network Heterogeneity. In: The International Conference on Networked Systems - NetSys'17 - PhD Forum, 2017, Göttingen.

4. Lopes, Felipe A.; Lima, Leonidas; Santos, Marcelo; Fidalgo, Robson; Fernandes, Stenio. High-level modeling and application validation for SDN. In: NOMS 2016 2016 IEEE/IFIP Network Operations and Management Symposium, 2016, Istanbul. (Qualis: A2)

5. Lopes, Felipe A.; Santos, Marcelo; Fidalgo, Robson; Fernandes, Stenio. A Software Engineering Perspective on SDN Programmability. IEEE Communications Surveys and Tutorials, v. PP, p. 1-1, 2015. (Qualis: A1)

7.1.2 Colaborações

- Cooperação entre Brasil e Alemanha no escopo do projeto SENDATE-PLANETS, por meio do programa ERASMUS Be Mundus.

7.2 AMEAÇAS À VALIDAÇÃO DO TRABALHO E LIMITAÇÕES

Investigar a validade da pesquisa refere-se a determinar se as conclusões obtidas (ou possíveis resultados) estão de acordo com a realidade. Ou seja, é preciso verificar a possibilidade das conclusões estarem erradas (MAXWELL, 2012). Cook, Campbell e Day (1979) definem quatro tipos de ameaças à validação:

1. Ameaças à validade da conclusão;

2. Ameaças à validade da construção;

3. Ameaças à validade interna; e

4. Ameaças à validade externa.

Nesta tese, foram analisados os quatro tipos de ameaças de acordo com o que foi proposto. A análise presente nas subseções a seguir categoriza estas ameaças, e deve auxiliar na elaboração de trabalhos futuros.

7.2.1 Ameaças à validade da conclusão

Ainda que a solução proposta baseada em MART tenha respondido as questões de pesquisa definidas no projeto dessa tese, é necessário comparar as conclusões identificadas (e.g., aumento do nível de abstração, melhoramento no aproveitamento de recursos) com outros cenários e soluções alternativas.

Além disso, nos Capítulos 5 e 6, as topologias avaliadas podem não refletir cenários reais, e uma investigação aprofundada do desempenho das técnicas apresentadas precisa ser realizada para afastar ameaças às conclusões obtidas a partir dos experimentos realizados.

7.2.2 Ameaças à validade da construção

Um aspecto importante ao se discutir a utilização de modelos para implementar aplicações SDN (autonômicas ou não), é a expressividade das sintaxes concreta e abstrata. Neste trabalho, estas sintaxes não foram exaustivamente avaliadas sob a perspectiva da expressividade. Portanto, há a necessidade de verificar se a notação da sintaxe concreta e se as entidades da sintaxe abstrata são suficientemente representativas para o contexto de SDN e para os projetistas de rede que farão uso do *framework* proposto.

O mesmo fator aplica-se à correção dos modelos gerados, ainda que a geração de código destes modelos tenha desempenhado o papel esperado nos experimentos, a construção destas transformações necessita de uma avaliação aprofundada.

Por outro lado, os experimentos em si foram elaborados de forma a afastar qualquer tipo de viés. Para isso, as métricas de desempenho foram colhidas sem nenhum tipo de favorecimento às técnicas aqui apresentadas, aplicando, por exemplo, aleatoriedade nas observações e cenários.

7.2.3 Ameaças à validade interna

A forma de implementação dos modelos (e.g., quantidade de regras, métricas a serem otimizadas, objetivos a serem atingidos) deve influenciar os resultados de desempenho dos sistemas ANM. Porém, até a finalização desta tese, não foi possível analisar de que forma e qual o impacto de outras formas de implementação destes modelos nas métricas avaliadas. Essa é uma ameaça interna para as conclusões obtidas até aqui.

7.2.4 Ameaças à validade externa

Dados os ambientes controlados em que os experimentos foram realizados, não é possível afirmar que os mesmos resultados de desempenho serão observados em ambientes reais ou com infraestrutura de simulação diferente. Futuramente, é preciso avaliar a implantação das técnicas apresentadas neste tese em cenários diversos e, possivelmente, com infraestruturas reais para validar os resultados obtidos ainda que fatores externos sejam modificados.

7.3 TRABALHOS FUTUROS

Como trabalhos futuros, além de buscar evitar as ameças à validação do trabalho, é preciso estender as possíveis ações e propriedades *self-** do *framework* MOSES, trazendo-o para o mais próximo possível do mundo real (e.g., implementando ações de adaptação em caso de falha dos dispositivos). Além disso, foi percebida a oportunidade de utilizar múltiplos algoritmos de aprendizagem profunda por reforço (e.g., *ensemble*, *boosting*) para maximizar a acurácia na seleção de ações e, possivelmente, diminuir o tempo de treinamento necessário.

Um outro ponto em aberto que merece investigação diz respeito à definição do conjunto de características que bases de conhecimento devem possuir. Não há ainda uma definição clara de quais atributos devem ser utilizados realizar o treinamento de algoritmos de aprendizagem de máquina ao serem utilizados para prover gerenciamento autonômico.

No contexto de predição de desempenho, um desafio é aumentar a acurácia da predição em cenários genéricos. Esta generalização para que o mesmo modelo de predição possa ser utilizado em diferentes situações é outra oportunidade de pesquisa que deve ser investigada futuramente. Além disso, é preciso considerar cenários diversificados para a avaliação de desempenho, ou pelo menos estudos de caso focados na prática (e.g., cenário de redes de grandes datacenters).

Por fim, o foco na implementação do **Raciocinador** revela que uma análise do estado da arte em DRL sob a perspectiva de SDN é de suma importância para que a comunidade de redes possa, *de facto*, alcançar a implementação de redes autonômicas.

REFERÊNCIAS

AFAQ, M.; REHMAN, S. U.; SONG, W.-C. Visualization of elephant flows and qos provisioning in sdn-based networks. In: IEEE. *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. [S.l.], 2015. p. 444–447.

AGARWAL, S.; KODIALAM, M.; LAKSHMAN, T. Traffic engineering in software defined networks. In: IEEE. *INFOCOM, 2013 Proceedings IEEE*. [S.l.], 2013. p. 2211–2219.

AHMAD, I.; NAMAL, S.; YLIANTTILA, M.; GURTOV, A. Towards software defined cognitive networking. In: *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*. [S.l.: s.n.], 2015. p. 1–5. ISSN 2157-4952.

AHMED, R.; BOUTABA, R. Design considerations for managing wide area software defined networks. *IEEE Communications Magazine*, IEEE, v. 52, n. 7, p. 116–123, 2014.

AKELLA, A. V.; XIONG, K. Quality of service (qos)-guaranteed network resource allocation via software defined networking (sdn). In: IEEE. *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*. [S.l.], 2014. p. 7–13.

ALENCAR, F.; SANTOS, M.; SANTANA, M.; FERNANDES, S. How Software Aging affects SDN: A view on the controllers. In: *2014 Global Information Infrastructure and Networking Symposium (GIIS)*. [S.l.]: IEEE, 2014. p. 1–6. ISBN 978-1-4799-5490-2.

ANWER, B.; BENSON, T.; FEAMSTER, N.; LEVIN, D.; REXFORD, J. A slick control plane for network middleboxes. In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.: s.n.], 2013. p. 147–148.

ARORA, N.; ZHANG, H.; LUMEZANU, C.; RHEE, J.; JIANG, G.; LU, H. *Hybrid network management*. [S.l.]: Google Patents, 2016. US Patent 9,450,823.

ASSMANN, U.; GÖTZ, S.; JÉZÉQUEL, J.-M.; MORIN, B.; TRAPP, M. A reference architecture and roadmap for models@ run. time systems. In: *Models@ run. time*. [S.l.]: Springer, 2014. p. 1–18.

AUTENRIETH, A.; ELBERS, J.-P.; KACZMAREK, P.; KOSTECKI, P. Cloud orchestration with sdn/openflow in carrier transport networks. In: IEEE. *Transparent Optical Networks (ICTON), 2013 15th International Conference on*. [S.l.], 2013. p. 1–4.

AVAYA. *SDN Expectations Report*. [S.l.], 2015. (Accessed on Sep. 11, 2015). Acesso em: 09/11/2015.

AYOUBI, S.; LIMAM, N.; SALAHUDDIN, M. A.; SHAHRIAR, N.; BOUTABA, R.; ESTRADA-SOLANO, F.; CAICEDO, O. M. Machine learning for cognitive network management. *IEEE Communications Magazine*, IEEE, v. 56, n. 1, p. 158–165, 2018.

BAINOMUGISHA, E.; CARRETON, A. L.; CUTSEM, T. v.; MOSTINCKX, S.; MEUTER, W. d. A survey on reactive programming. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 4, p. 52, 2013.

BARI, M. F.; CHOWDHURY, S. R.; AHMED, R.; BOUTABA, R. Policycop: An autonomic qos policy enforcement framework for software defined networks. In: IEEE. *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*. [S.l.], 2013. p. 1–7.

BARRON, J.; CROTTY, M.; ELAHI, E.; RIGGIO, R.; LOPEZ, D. R.; LEON, M. P. de. Towards self-adaptive network management for a recursive network architecture. In: IEEE. *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. [S.l.], 2016. p. 1143–1148.

BAUER, R.; ZITTERBART, M. Port based capacity extensions (pbces): Improving sdns flow table scalability. In: IEEE. *Teletraffic Congress (ITC 28), 2016 28th International*. [S.l.], 2016. v. 1, p. 225–233.

Bega, D.; Gramaglia, M.; Fiore, M.; Banchs, A.; Costa-Perez, X. Deepcog: Cognitive network management in sliced 5g networks with deep learning. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2019. p. 280–288. ISSN 0743-166X.

BEHRINGER, M. H.; CARPENTER, B. E.; ECKERT, T.; CIAVAGLIA, L.; PIERRE, P.; LIU, B.; NOBRE, J. C.; STRASSNER, J. *A Reference Model for Autonomic Networking*. [S.l.], 2017. Work in Progress. Disponível em: <https://datatracker.ietf.org/doc/html/draft-ietf-anima-reference-model-04>.

BENAYAS, F.; CARRERA, Á.; GARCÍA-AMADO, M.; IGLESIAS, C. A. A semantic data lake framework for autonomous fault management in sdn environments. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 30, n. 9, p. e3629, 2019.

BENCOMO, N.; BLAIR, G. Using architecture models to support the generation and operation of component-based adaptive systems. In: *Software engineering for self-adaptive systems*. [S.l.]: Springer, 2009. p. 183–200.

BJORKLUND, M. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. [S.l.], 2010. <http://www.rfc-editor.org/rfc/rfc6020.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc6020.txt>.

BLAIR, G.; BENCOMO, N.; FRANCE, R. B. Models@ run.time. *Computer*, v. 42, n. 10, p. 22–27, Oct 2009. ISSN 0018-9162.

BORDINI, R. H.; BRAUBACH, L.; DASTANI, M.; SEGHROUCHNI, A. E. F.; GOMEZ-SANZ, J. J.; LEITE, J.; O'HARE, G.; POKAHR, A.; RICCI, A. A survey of programming languages and platforms for multi-agent systems. *Informatica*, v. 30, n. 1, 2006.

BOTTA, A.; DAINOTTI, A.; PESCAPÈ, A. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, v. 56, n. 15, p. 3531–3547, 2012.

BRADEN, R.; CLARK, D.; SHENKER, S. *RFC1633: Integrated services in the internet architecture: an overview*. [S.l.]: RFC Editor, 1994.

BRADEN, R.; ZHANG, L.; BERSON, S.; HERZOG, S.; JAMIN, S. Resource reservation protocol:(rsvp); version 1 functional specification. University of Michigan, 1997.

CABA, C.; SOLER, J. Apis for qos configuration in software defined networks. In: IEEE. *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. [S.l.], 2015. p. 1–5.

CAMPBELL, A. T.; KATZELA, I.; MIKI, K.; VICENTE, J. Open signaling for atm, internet and mobile networks (opensig'98). *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 29, n. 1, p. 97–108, jan. 1999. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/505754.505762>.

CASE, A. F. Computer-aided software engineering (case): technology for improving software development productivity. *ACM SIGMIS Database*, ACM, v. 17, n. 1, p. 35–43, 1985.

CHAPARADZA, R.; MERIEM, T. B.; RADIER, B.; SZOTT, S.; WÓDCZAK, M.; PRAKASH, A.; DING, J.; SOULHI, S.; MIHAILOVIC, A. SDN enablers in the ETSI AFI GANA reference model for autonomic management & control (emerging standard), and virtualization impact. In: *Workshops Proceedings of the Global Communications Conference, GLOBECOM 2013, Atlanta, GA, USA, December 9-13, 2013*. [s.n.], 2013. p. 818–823. Disponível em: <https://doi.org/10.1109/GLOCOMW.2013.6825090>.

COOK, T. D.; CAMPBELL, D. T.; DAY, A. *Quasi-experimentation: Design & analysis issues for field settings*. [S.l.]: Houghton Mifflin Boston, 1979. v. 351.

CZARNECKI, K.; HELSEN, S. Classification of model transformation approaches. In: USA. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*. [S.l.], 2003. v. 45, n. 3, p. 1–17.

DARADKEH, Y. I.; ALDHAIFALLAH, M.; NAMIOT, D.; SNEPS-SNEPPE, M. On standards for application level interfaces in sdn. *International Journal of Advanced Computer Science and Applications, IJACSA*, v. 7, n. 10, 2016.

DIAO, Y.; HELLERSTEIN, J. L.; PAREKH, S.; GRIFFITH, R.; KAISER, G. E.; PHUNG, D. A control theory foundation for self-managing computing systems. *IEEE journal on selected areas in communications*, IEEE, v. 23, n. 12, p. 2213–2222, 2005.

DMTF. *Common Information Model (CIM) Specification Version 2.49*. [S.l.], 2012. Disponível em: <http://www.dmtf.org/sites/default/files/standards/documents/DSP0004_2.7.0.pdf>.

DORIA, A.; SALIM, J. H.; HAAS, R.; KHOSRAVI, H.; WANG, W.; DONG, L.; GOPAL, R.; HALPERN, J. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. [S.l.], 2010. <http://www.rfc-editor.org/rfc/rfc5810.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc5810.txt>.

DROSTE, H.; ROST, P.; DOLL, M.; BERBERANA, I.; MANNWEILER, C.; BREITBACH, M.; BANCHS, A.; PUENTE, M. A. An adaptive 5g multiservice and multitenant radio access network architecture. *Transactions on Emerging Telecommunications Technologies*, Wiley Online Library, v. 27, n. 9, p. 1262–1270, 2016.

EGILMEZ, H. E.; CIVANLAR, S.; TEKALP, A. M. An optimization framework for qos-enabled adaptive video streaming over openflow networks. *IEEE Transactions on Multimedia*, IEEE, v. 15, n. 3, p. 710–715, 2012.

EGILMEZ, H. E.; DANE, S. T.; BAGCI, K. T.; TEKALP, A. M. Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In: IEEE. *Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference*. [S.l.], 2012. p. 1–8.

ČEJKA, T.; KREJčí, R. Configuration of open vswitch using of-config. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2016. p. 883–888.

ENNS, R.; BJORKLUND, M.; SCHOENWAELDER, J.; BIERMAN, A. *Network Configuration Protocol (NETCONF)*. [S.l.], 2011. <http://www.rfc-editor.org/rfc/rfc6241.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc6241.txt>.

ERICKSON, D. The beacon openflow controller. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 13–18.

ERICKSON, D. The beacon openflow controller. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 13–18. ISBN 978-1-4503-2178-5.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn. *Queue*, ACM, New York, NY, USA, v. 11, n. 12, p. 20:20–20:40, dez. 2013. ISSN 1542-7730. Disponível em: <http://doi.acm.org/10.1145/2559899.2560327>.

FERNANDES, S. Principles of performance evaluation of computer networks. In: _____. *Performance Evaluation for Network Services, Systems and Protocols*. Cham: Springer International Publishing, 2017. p. 1–43. ISBN 978-3-319-54521-9. Disponível em: <https://doi.org/10.1007/978-3-319-54521-9_1>.

FOFACK, N. C.; NAIN, P.; NEGLIA, G.; TOWSLEY, D. Performance evaluation of hierarchical ttl-based cache networks. *Computer Networks*, Elsevier, v. 65, p. 212–231, 2014.

FONTES, R. R.; OLIVEIRA, A. L.; SAMPAIO, P. N.; PINHEIRO, T. R.; FIGUEIRA, R. A. Authoring of openflow networks with visual network description (sdn version)(wip). In: SOCIETY FOR COMPUTER SIMULATION INTERNATIONAL. *Proceedings of the 2014 Summer Simulation Multiconference*. [S.l.], 2014. p. 22.

FONTES, R. R.; OLIVEIRA, A. L. C.; SAMPAIO, P. N. M.; PINHEIRO, T. R.; FIGUEIRA, R. A. R. B. Authoring of openflow networks with visual network description (sdn version) (wip). In: *Proceedings of the 2014 Summer Simulation Multiconference*. San Diego, CA, USA: Society for Computer Simulation International, 2014. (SummerSim '14), p. 22:1–22:6.

FOSTER, N.; HARRISON, R.; FREEDMAN, M. J.; MONSANTO, C.; REXFORD, J.; STORY, A.; WALKER, D. Frenetic: A network programming language. In: ACM. *ACM Sigplan Notices*. [S.l.], 2011. v. 46, n. 9, p. 279–291.

FOSTER, N.; HARRISON, R.; FREEDMAN, M. J.; MONSANTO, C.; REXFORD, J.; STORY, A.; WALKER, D. Frenetic: A network programming language. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 46, n. 9, p. 279–291, set. 2011. ISSN 0362-1340.

GELENBE, E. A software defined self-aware network: The cognitive packet network. In: *2013 Ninth International Conference on Semantics, Knowledge and Grids*. [S.l.: s.n.], 2013. p. 1–5.

GHAZIKHANI, A.; MONSEFI, R.; YAZDI, H. S. Online neural network model for non-stationary and imbalanced data stream classification. *International Journal of Machine Learning and Cybernetics*, Springer, v. 5, n. 1, p. 51–62, 2014.

GHORBANZADEH, M.; ABDELHADI, A.; CLANCY, C. Quality of service in communication systems. In: *Cellular Communications Systems in Congested Environments*. [S.l.]: Springer, 2017. p. 1–20.

GONZÁLEZ, C. A.; CABOT, J. Formal verification of static software models in mde: A systematic review. *Information and Software Technology*, Elsevier, v. 56, n. 8, p. 821–838, 2014.

GRONBACK, R. C. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. [S.l.]: Pearson Education, 2009.

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833.

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1384609.1384625>.

HAREL, D.; RUMPE, B. Meaningful modeling: what's the semantics of"semantics"? *Computer*, IEEE, v. 37, n. 10, p. 64–72, 2004.

HARKAL, V. B.; DESHMUKH, A. Software defined networking with floodlight controller. *International Journal of Computer Applications*, v. 975, p. 23–27, 2016.

HARRISON, P. G.; PATEL, N. M. *Performance Modelling of Communication Networks and Computer Architectures (International Computer S*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992. ISBN 0201544199.

HELLER, B.; SHERWOOD, R.; MCKEOWN, N. The controller placement problem. *ACM SIGCOMM Computer Communication Review*, v. 42, n. 4, p. 473, 2012. ISSN 01464833. Disponível em: <http://dl.acm.org/citation.cfm?doid=2377677.2377767>.

HENDERSON, T. R.; LACAGE, M.; RILEY, G. F.; DOWELL, C.; KOPENA, J. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, v. 14, n. 14, p. 527, 2008.

HILL, N.; STONE, W. C. *Success through a positive mental attitude*. [S.l.]: Simon and Schuster, 1991.

HINRICHS, T. L.; GUDE, N. S.; CASADO, M.; MITCHELL, J. C.; SHENKER, S. Practical declarative network management. In: ACM. *Proceedings of the 1st ACM workshop on Research on enterprise networking*. [S.l.], 2009. p. 1–10.

HORN, P. *Autonomic Computing: IBM's Perspective on the State of Information Technology*. [S.l.], 2001.

ISOLANI, P. H.; WICKBOLDT, J. A.; BOTH, C. B.; ROCHOL, J.; GRANVILLE, L. Z. Interactive monitoring, visualization, and configuration of openflow-based sdn. In: IEEE. *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. [S.l.], 2015. p. 207–215.

JACKSON, E.; SZTIPANOVITS, J. Formalizing the structural semantics of domain-specific modeling languages. *Software & Systems Modeling*, Springer, v. 8, n. 4, p. 451–478, 2009.

JAHROMI, H. Z.; HINES, A.; DELANEV, D. T. Towards application-aware networking: Ml-based end-to-end application kpi/qoe metrics characterization in sdn. In: IEEE. *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. [S.l.], 2018. p. 126–131.

JARSCHEL, M.; ZINNER, T.; HOSSFELD, T.; TRAN-GIA, P.; KELLERER, W. Interfaces, attributes, and use cases: A compass for sdn. *IEEE Communications Magazine*, IEEE, v. 52, n. 6, p. 210–217, 2014.

JENNINGS, B.; MEER, S. V. D.; BALASUBRAMANIAM, S.; BOTVICH, D.; FOGHLU, M. O.; DONNELLY, W.; STRASSNER, J. Towards autonomic management of communications networks. *IEEE Communications Magazine*, v. 45, n. 10, p. 112–121, October 2007. ISSN 0163-6804.

KALMBACH, P.; ZERWAS, J.; BABARCZI, P.; BLENK, A.; KELLERER, W.; SCHMID, S. Empowering self-driving networks. In: ACM. *Proceedings of the Afternoon Workshop on Self-Driving Networks*. [S.l.], 2018. p. 8–14.

KARAKUS, M.; DURRESI, A. Quality of service (qos) in software defined networking (sdn): A survey. *Journal of Network and Computer Applications*, Elsevier, v. 80, p. 200–218, 2017.

KAZEMIAN, P.; CHANG, M.; ZENG, H.; VARGHESE, G.; MCKEOWN, N.; WHYTE, S. Real time network policy checking using header space analysis. In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2013. (nsdi'13), p. 99–112. Disponível em: <http://dl.acm.org/citation.cfm?id=2482626.2482638>.

KELLY, F. P. Stochastic models of computer communication systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, JSTOR, p. 379–395, 1985.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE, v. 36, n. 1, p. 41–50, 2003.

KHURSHID, A.; ZHOU, W.; CAESAR, M.; GODFREY, P. B. Veriflow: Verifying network-wide invariants in real time. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: ACM, 2012. (HotSDN '12), p. 49–54. ISBN 978-1-4503-1477-0. Disponível em: <http://doi.acm.org/10.1145/2342441.2342452>.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 114–119, February 2013. ISSN 0163-6804.

KIM, H.; REICH, J.; GUPTA, A.; SHAHBAZ, M.; FEAMSTER, N.; CLARK, R. Kinetic: Verifiable dynamic network control. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015. p. 59–72. ISBN 978-1-931971-218.

KIM, H.; REICH, J.; GUPTA, A.; SHAHBAZ, M.; FEAMSTER, N.; CLARK, R. Kinetic: Verifiable dynamic network control. In: *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2015. (NSDI'15), p. 59–72. ISBN 978-1-931971-218. Disponível em: <http://dl.acm.org/citation.cfm?id=2789770.2789775>.

KIM, H.; REICH, J.; GUPTA, A.; SHAHBAZ, M.; FEAMSTER, N.; CLARK, R. J. Kinetic: Verifiable dynamic network control. In: *NSDI*. [S.l.: s.n.], 2015. p. 59–72.

KNIGHT, S.; NGUYEN, H.; FALKNER, N.; BOWDEN, R.; ROUGHAN, M. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, v. 29, n. 9, p. 1765 –1775, october 2011. ISSN 0733-8716.

KOLOVOS, D.; ROSE, L.; PAIGE, R. *The Epsilon Book*. [S.l.]: Eclipse, 2013. <http://www.eclipse.org/epsilon/doc/book/>. (Accessed on Sep. 9, 2015).

KOLOVOS, D. S.; ROSE, L. M.; PAIGE, R. F.; POLACK, F. A. Raising the level of abstraction in the development of gmf-based graphical model editors. In: IEEE. *2009 ICSE Workshop on Modeling in Software Engineering*. [S.l.], 2009. p. 13–19.

KOPONEN, T.; CASADO, M.; GUDE, N.; STRIBLING, J.; POUTIEVSKI, L.; ZHU, M.; RAMANATHAN, R.; IWATA, Y.; INOUE, H.; HAMA, T. et al. Onix: A distributed control platform for large-scale production networks. In: *OSDI*. [S.l.: s.n.], 2010. v. 10, p. 1–6.

KOUNEV, S.; SPINNER, S.; MEIER, P. Introduction to queueing petri nets: Modeling formalism, tool support and case studies. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: ACM, 2012. (ICPE '12), p. 9–18. ISBN 978-1-4503-1202-8. Disponível em: <http://doi.acm.org/10.1145/2188286.2188290>.

KOUTSOURIS, N.; TSAGKARIS, K.; DEMESTICHAS, P.; MAMATAS, L.; CLAYMAN, S.; GALIS, A. Managing software-driven networks with a unified management framework. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. [S.l.: s.n.], 2013. p. 1084–1085. ISSN 1573-0077.

KREUTZ, D.; RAMOS, F.; VERISSIMO, P. Towards secure and dependable software-defined networks. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 55–60.

KREUTZ, D.; RAMOS, F. M.; VERISSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14–76, 2015.

KUKLIŃSKI, S.; WYTRĘBOWICZ, J.; DINH, K. T.; TANTAR, E. Application of cognitive techniques to network management and control. In: *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. [S.l.]: Springer, 2014. p. 79–93.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: Rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: <http://doi.acm.org/10.1145/1868447.1868466>.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, IEEE, v. 16, n. 1, p. 493–512, 2014.

LEHMANN, G.; BLUMENDORF, M.; TROLLMANN, F.; ALBAYRAK, S. Meta-modeling runtime models. In: SPRINGER. *International Conference on Model Driven Engineering Languages and Systems*. [S.l.], 2010. p. 209–223.

LEHOCINE, M. B.; BATOUCHE, M. Achieving efficiency in autonomic network management of ip networks based on sdn management logic. *International Journal of Communication Networks and Distributed Systems*, Inderscience Publishers (IEL), v. 20, n. 4, p. 484–518, 2018.

LEMOS, R. D.; GIESE, H.; MÜLLER, H. A.; SHAW, M.; ANDERSSON, J.; LITOIU, M.; SCHMERL, B.; TAMURA, G.; VILLEGAS, N. M.; VOGEL, T. et al. Software engineering for self-adaptive systems: A second research roadmap. In: *Software Engineering for Self-Adaptive Systems II*. [S.l.]: Springer, 2013. p. 1–32.

LI, C.; ALTAMIMI, T.; ZARGARI, M. H.; CASALE, G.; PETRIU, D. Tulsa: A tool for transforming uml to layered queueing networks for performance analysis of data intensive applications. In: BERTRAND, N.; BORTOLUSSI, L. (Ed.). *Quantitative Evaluation of Systems*. Cham: Springer International Publishing, 2017. p. 295–299. ISBN 978-3-319-66335-7.

LI, D.; RUAN, L.; XIAO, L.; ZHU, M.; DUAN, W.; ZHOU, Y.; CHEN, M.; XIA, Y.; ZHU, M. High availability for non-stop network controller. In: IEEE. *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. [S.l.], 2014. p. 1–5.

LI, H.; QUE, X.; HU, Y.; XIANGYANG, G.; WENDONG, W. An autonomic management architecture for sdn-based multi-service network. In: *2013 IEEE Globecom Workshops (GC Wkshps)*. [S.l.: s.n.], 2013. p. 830–835. ISSN 2166-0077.

LIN, T.-N.; HSU, Y.-M.; KAO, S.-Y.; CHI, P.-W. Opene2eqos: Meter-based method for end-to-end qos of multimedia services over sdn. In: IEEE. *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. [S.l.], 2016. p. 1–6.

LOPES, F. A.; LIMA, L.; SANTOS, M.; FIDALGO, R.; FERNANDES, S. High-level modeling and application validation for sdn. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2016. p. 197–205.

LOPES, F. A.; SANTOS, M.; FIDALGO, R.; FERNANDES, S. Model-driven networking: A novel approach for sdn applications development. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. [S.l.: s.n.], 2015. p. 770–773. ISSN 1573-0077.

LOPES, F. A.; SANTOS, M.; FIDALGO, R.; FERNANDES, S. A software engineering perspective on sdn programmability. *IEEE Communications Surveys Tutorials*, v. 18, n. 2, p. 1255–1272, Secondquarter 2016. ISSN 1553-877X.

LOPES, F. A.; SOUZA, R. R.; FERNANDES, S. Automatically generated simulations for predicting software-defined networking performance. In: IEEE. *2018 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2018. p. 915–918.

LOPES, F. A.; TIBURCIO, P.; BAUER, R.; FERNANDES, S.; ZITTERBART, M. Model-based flow delegation for improving sdn infrastructure compatibility. In: IEEE. *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. [S.l.], 2018. p. 1–9.

MAHENDRAN, V.; GUNASEKARAN, R.; MURTHY, C. S. R. Performance modeling of delay-tolerant network routing via queueing petri nets. *IEEE Transactions on Mobile Computing*, IEEE, v. 13, n. 8, p. 1816–1828, 2014.

MATIAS, J.; GARAY, J.; TOLEDO, N.; UNZILLA, J.; JACOB, E. Toward an sdn-enabled nfv architecture. *IEEE Communications Magazine*, IEEE, v. 53, n. 4, p. 187–193, 2015.

MAXWELL, J. A. *Qualitative research design: An interactive approach*. [S.l.]: Sage publications, 2012. v. 41.

MCCAULEY, J.; SCOTT, C.; WUNDSAM, A.; GUEDES, D. *POX OpenFlow Controller - Version Fall 2013*. [S.l.], 2015. (Accessed on Aug. 29, 2015). Acesso em: 08/29/2015.

MCCLOGHRIE, K.; FINE, M.; SELIGSON, J.; CHAN, K.; HAHN, S.; SAHITA, R.; SMITH, A.; REICHMEYER, F. *Structure of Policy Provisioning Information (SPPI)*. [S.l.], 2001.

MCCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. *Structure of Management Information Version 2 (SMIv2)*. [S.l.], 1999. <http://www.rfc-editor.org/rfc/rfc2578.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc2578.txt>.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008.

MEDVED, J.; TKACIK, A.; VARGA, R.; GRAY, K. Opendaylight: Towards a model-driven sdn controller architecture. In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*. [S.l.: s.n.], 2014. p. 1–6.

MENDONCA, M.; OBRACZKA, K.; TURLETTI, T. The case for software-defined networking in heterogeneous networked environments. In: *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*. New York, NY, USA: ACM,

2012. (CoNEXT Student '12), p. 59–60. ISBN 978-1-4503-1779-5. Disponível em: <http://doi.acm.org/10.1145/2413247.2413283>.

MESTRES, A.; RODRIGUEZ-NATAL, A.; CARNER, J.; BARLET-ROS, P.; ALARCÓN, E.; SOLÉ, M.; MUNTÉS-MULERO, V.; MEYER, D.; BARKAI, S.; HIBBETT, M. J. et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 47, n. 3, p. 2–10, 2017.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

MOHAGHEGHI, P.; GILANI, W.; STEFANESCU, A.; FERNANDEZ, M. A.; NORDMOEN, B. r.; FRITZSCHE, M. Where does model-driven engineering help? Experiences from three industrial cases. *Software & Systems Modeling*, v. 12, n. 3, p. 619–639, out. 2011. ISSN 1619-1366.

MOKDAD, L.; BEN-OTHMAN, J.; YAHYA, B.; NIAGNE, S. Performance evaluation tools for qos mac protocol for wireless sensor networks. *Ad Hoc Networks*, Elsevier, v. 12, p. 86–99, 2014.

MONSANTO, C.; FOSTER, N.; HARRISON, R.; WALKER, D. A compiler and run-time system for network programming languages. In: ACM. *ACM SIGPLAN Notices*. [S.l.], 2012. v. 47, n. 1, p. 217–230.

MORENO-VOZMEDIANO, R.; MONTERO, R. S.; LLORENTE, I. M. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, IEEE, v. 45, n. 12, p. 65–72, 2012.

MOVAHEDI, Z.; AYARI, M.; LANGAR, R.; PUJOLLE, G. A survey of autonomic network architectures and evaluation criteria. *IEEE Communications Surveys & Tutorials*, IEEE, v. 14, n. 2, p. 464–490, 2012.

MURCH, R. *Autonomic computing*. [S.l.]: Ibm Press, 2004.

NELSON, B. J. Remote procedure call. Carnegie-Mellon University Pittsburgh, PA, 1981.

NELSON, T.; FERGUSON, A. D.; SCHEER, M. J. G.; KRISHNAMURTHI, S. Tierless programming and reasoning for software-defined networks. In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2014. (NSDI'14), p. 519–531. ISBN 978-1-931971-09-6.

NEVES, P.; CALÉ, R.; COSTA, M. R.; PARADA, C.; PARREIRA, B.; ALCARAZ-CALERO, J.; WANG, Q.; NIGHTINGALE, J.; CHIRIVELLA-PEREZ, E.; JIANG, W. et al. The selfnet approach for autonomic management in an nfv/sdn networking paradigm. *International Journal of Distributed Sensor Networks*, SAGE Publications Sage UK: London, England, v. 12, n. 2, p. 2897479, 2016.

NICHOLS, K.; JACOBSON, V.; ZHANG, L. et al. *A two-bit differentiated services architecture for the Internet*. [S.l.], 1999.

NILSSON, H.; COURTNEY, A.; PETERSON, J. Functional reactive programming, continued. In: ACM. *Proceedings of the 2002 ACM SIGPLAN workshop on Haskell*. [S.l.], 2002. p. 51–64.

OCHOA-ADAY, L.; CERVELLó-PASTOR, C.; FERNáNDEZ-FERNáNDEZ, A. Self-healing and sdn: bridging the gap. *Digital Communications and Networks*, 2019. ISSN 2352-8648. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2352864818302827>.

OLIVEIRA, R. L. S. D.; SCHWEITZER, C. M.; SHINODA, A. A.; PRETE, L. R. Using mininet for emulation and prototyping software-defined networks. In: IEEE. *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*. [S.l.], 2014. p. 1–6.

ONF. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/whitepapers/wp-sdn-newnorm.pdf>.

ONF. *OpenFlow Switch Specification v1.3.2*. [S.l.], 2013. v. 0. (Accessed on Sep. 12, 2015).

OREIZY, P.; GORLICK, M. M.; TAYLOR, R. N.; HEIMHIGNER, D.; JOHNSON, G.; MEDVIDOVIC, N.; QUILICI, A.; ROSENBLUM, D. S.; WOLF, A. L. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, IEEE, v. 14, n. 3, p. 54–62, 1999.

ÖZGÜR, T. *Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling in the context of Model-Driven Development*. 2007.

PILZ, A.; SWOBODA, J. Network management information models. *Aeu-International Journal of Electronics and Communications*, Elsevier, v. 58, n. 3, p. 165–171, 2004.

PINHEIRO, B.; CHAVES, R.; CERQUEIRA, E.; ABELEM, A. Cim-sdn: A common information model extension for software-defined networking. In: IEEE. *Globecom Workshops (GC Wkshps), 2013 IEEE*. [S.l.], 2013. p. 836–841.

POULIOS, G.; TSAGKARIS, K.; DEMESTICHAS, P.; TALL, A.; ALTMAN, Z.; DESTRé, C. Autonomics and sdn for self-organizing networks. In: *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*. [S.l.: s.n.], 2014. p. 830–835. ISSN 2154-0217.

PRAS, A.; SCHOENWAELDER, J. *On the Difference Between Information Models and Data Models*. United States, 2003.

QI, Q.; WANG, W.; GONG, X.; QUE, X. A sdn-based network virtualization architecture with autonomie management. In: *2014 IEEE Globecom Workshops (GC Wkshps)*. [S.l.: s.n.], 2014. p. 178–182. ISSN 2166-0077.

QURESHI, N. A.; PERINI, A. Requirements engineering for adaptive service based applications. In: IEEE. *2010 18th IEEE International Requirements Engineering Conference*. [S.l.], 2010. p. 108–111.

REICH, J.; MONSANTO, C.; FOSTER, N.; REXFORD, J.; WALKER, D. Modular sdn programming with pyretic. *Technical Reprot of USENIX*, 2013.

REICH, J.; MONSANTO, C.; FOSTER, N.; REXFORD, J.; WALKER, D. Modular sdn programming with pyretic. *Technical Reprot of USENIX*, 2013.

REITBLATT, M.; CANINI, M.; GUHA, A.; FOSTER, N. Fattire: Declarative fault tolerance for software-defined networks. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 109–114.

RINALDI, S.; FERRARI, P.; BRANDãO, D.; SULIS, S. Software defined networking applied to the heterogeneous infrastructure of smart grid. In: *2015 IEEE World Conference on Factory Communication Systems (WFCS)*. [S.l.: s.n.], 2015. p. 1–4.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

RYGIELSKI, P.; KOUNEV, S.; TRAN-GIA, P. Flexible performance prediction of data center networks using automatically generated simulation models. In: *SimuTools*. [S.l.: s.n.], 2015. p. 119–128.

RYGIELSKI, P.; SELIUCHENKO, M.; KOUNEV, S. Modeling and prediction of software-defined networks performance using queueing petri nets. In: ICST (INSTITUTE FOR COMPUTER SCIENCES, SOCIAL-INFORMATICS AND TELECOMMUNICATIONS ENGINEERING). *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*. [S.l.], 2016. p. 66–75.

RYGIELSKI, P.; SELIUCHENKO, M.; KOUNEV, S. Modeling and prediction of software-defined networks performance using queueing petri nets. In: *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016. (SIMUTOOLS'16), p. 66–75. ISBN 978-1-63190-120-1. Disponível em: <http://dl.acm.org/citation.cfm?id=3021426.3021437>.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*, ACM New York, NY, USA, v. 4, n. 2, p. 1–42, 2009.

SALEHIE, M.; TAHVILDARI, L. Towards a goal-driven approach to action selection in self-adaptive software. *Software: Practice and Experience*, Wiley Online Library, v. 42, n. 2, p. 211–233, 2012.

SAMAAN, N.; KARMOUCH, A. Towards autonomic network management: an analysis of current and future research directions. *IEEE Communications Surveys Tutorials*, v. 11, n. 3, p. 22–36, rd 2009. ISSN 1553-877X.

SCHMID, S.; SIFALAKIS, M.; HUTCHISON, D. Towards autonomic networks. *Autonomic Networking*, Springer, v. 4195, p. 1–11, 2006.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural networks*, Elsevier, v. 61, p. 85–117, 2015.

SCHMIDT, D. Guest editor's introduction: Model-driven engineering. *Computer*, v. 39, n. 2, p. 25–31, Feb 2006. ISSN 0018-9162.

SCHMIDT, D. C. Guest editor's introduction: Model-driven engineering. *Computer*, v. 39, n. 2, p. 25–31, Feb 2006. ISSN 0018-9162.

SCHONWALDER, J.; BJORKLUND, M.; SHAFER, P. Network configuration management using netconf and yang. *IEEE communications magazine*, IEEE, v. 48, n. 9, 2010.

SEDDIKI, M. S.; SHAHBAZ, M.; DONOVAN, S.; GROVER, S.; PARK, M.; FEAMSTER, N.; SONG, Y.-Q. Flowqos: Qos for the rest of us. In: ACM. *Proceedings of the third workshop on Hot topics in software defined networking*. [S.l.], 2014. p. 207–208.

SHAFER, P. *An Architecture for Network Management Using NETCONF and YANG*. [S.l.], 2011.

SHARMA, S.; STAESSENS, D.; COLLE, D.; PALMA, D.; GONCALVES, J.; FIGUEIREDO, R.; MORRIS, D.; PICKAVET, M.; DEMEESTER, P. Implementing quality of service for the software defined networking enabled future internet. In: IEEE. *2014 third European workshop on software defined networks*. [S.l.], 2014. p. 49–54.

SHEN, S.-H.; HUANG, L.-H.; YANG, D.-N.; CHEN, W.-T. Reliable multicast routing for software-defined networks. In: IEEE. *2015 IEEE Conference on Computer Communications (INFOCOM)*. [S.l.], 2015. p. 181–189.

SHU, Y. Heterogeneous networking architecture based on sdn. *Chinese Journal of Electronics*, v. 26, n. 1, p. 166–171, 2017. ISSN 1022-4653.

SMITH, M.; DVORKIN, M.; LARIBI, Y.; PANDEY, V.; GARG, P.; WEIDENBACHER, N. *OpFlex Control Protocol*. [S.l.], 2014. <http://www.ietf.org/internet-drafts/draft-smith-opflex-00.txt>. Disponível em: <http://www.ietf.org/internet-drafts/draft-smith-opflex-00.txt>.

SONG, H. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 127–132.

SONG, H. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. (HotSDN '13), p. 127–132. ISBN 978-1-4503-2178-5.

SOULÉ, R.; BASU, S.; MARANDI, P. J.; PEDONE, F.; KLEINBERG, R.; SIRER, E. G.; FOSTER, N. Merlin: A language for provisioning network resources. In: ACM. *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. [S.l.], 2014. p. 213–226.

STAMOU, A.; DIMITRIOU, N.; KONTOVASILIS, K.; PAPAVASSILIOU, S. Autonomic handover management for heterogeneous networks in a future internet context: A survey. *IEEE Communications Surveys & Tutorials*, IEEE, 2019.

STOJANOVIC, L.; SCHNEIDER, J.; MAEDCHE, A.; LIBISCHER, S.; STUDER, R.; LUMPP, T.; ABECKER, A.; BREITER, G.; DINGER, J. The role of ontologies in autonomic computing systems. *IBM Systems Journal*, IBM, v. 43, n. 3, p. 598–616, 2004.

STRAFFIN, P. *Topics in the Theory of Voting.* [S.l.]: ERIC, 1980.

STRASSNER, J. Autonomic networking-theory and practice. In: IEEE. *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP.* [S.l.], 2004. v. 1, p. 927–Vol.

SU, Z.; WANG, T.; XIA, Y.; HAMDI, M. Cemon: A cost-effective flow monitoring system in software defined networks. *Computer Networks*, v. 92, p. 101 – 115, 2015. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128615003291>.

SUN, S.; GONG, L.; RONG, B.; LU, K. An intelligent sdn framework for 5g heterogeneous networks. *IEEE Communications Magazine*, IEEE, v. 53, n. 11, p. 142–147, 2015.

TAHAEI, H.; SALLEH, R. B.; RAZAK, M. F. A.; KO, K.; ANUAR, N. B. Cost effective network flow measurement for software defined networks: A distributed controller scenario. *IEEE Access*, IEEE, v. 6, p. 5182–5198, 2018.

TANGARI, G.; TUNCER, D.; CHARALAMBIDES, M.; QI, Y.; PAVLOU, G. Self-adaptive decentralized monitoring in software-defined networks. *IEEE Transactions on Network and Service Management*, p. 1–1, 2018. ISSN 1932-4537.

TENNENHOUSE, D. L.; SMITH, J. M.; SINCOSKIE, W. D.; WETHERALL, D. J.; MINDEN, G. J. A survey of active network research. *Comm. Mag.*, IEEE Press, Piscataway, NJ, USA, v. 35, n. 1, p. 80–86, jan. 1997. ISSN 0163-6804. Disponível em: <http://dx.doi.org/10.1109/35.568214>.

TOMOVIC, S.; PRASAD, N.; RADUSINOVIC, I. Sdn control framework for qos provisioning. In: IEEE. *2014 22nd Telecommunications Forum Telfor (TELFOR).* [S.l.], 2014. p. 111–114.

TOMOVIC, S.; RADUSINOVIC, I.; PRASAD, N. Performance comparison of qos routing algorithms applicable to large-scale sdn networks. In: IEEE. *IEEE EUROCON 2015-International Conference on Computer as a Tool (EUROCON).* [S.l.], 2015. p. 1–6.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: A distributed control plane for openflow. In: *Proceedings of the 2010 internet network management conference on Research on enterprise networking.* [S.l.: s.n.], 2010. p. 3–3.

TSAGKARIS, K.; GALANI, A.; DEMESTICHAS, P.; NGUENGANG, G.; BOUET, M.; YAHIA, I. G. B.; DESTRE, C.; GHAMRI-DOUDANE, S.; CIAVAGLIA, L. Designing the core components of an operator-driven, framework for unifying autonomic network and service management. In: *2012 Future Network Mobile Summit (FutureNetw).* [S.l.: s.n.], 2012. p. 1–8.

TSAGKARIS, K.; KOUTSOURIS, N.; DEMESTICHAS, P.; COMBES, R.; ALTMAN, Z. Son coordination in a unified management framework. In: IEEE. *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th.* [S.l.], 2013. p. 1–5.

TSAGKARIS, K.; LOGOTHETIS, M.; FOTEINOS, V.; POULIOS, G.; MICHALOLIAKOS, M.; DEMESTICHAS, P. Customizable autonomic network management: Integrating autonomic network management and software-defined networking. *IEEE Vehicular Technology Magazine*, v. 10, n. 1, p. 61–68, March 2015. ISSN 1556-6072.

TUNCER, D.; CHARALAMBIDES, M.; CLAYMAN, S.; PAVLOU, G. Adaptive resource management and control in software defined networks. *IEEE Transactions on Network and Service Management*, v. 12, n. 1, p. 18–33, March 2015. ISSN 1932-4537.

TURNER, J. S.; TAYLOR, D. E. Diversifying the internet. In: IEEE. *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*. [S.l.], 2005. v. 2, p. 6–pp.

VARGA, A.; HORNIG, R. An overview of the omnet++ simulation environment. In: ICST (INSTITUTE FOR COMPUTER SCIENCES, SOCIAL-INFORMATICS AND TELECOMMUNICATIONS ENGINEERING). *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. [S.l.], 2008. p. 60.

VOELLMY, A.; HUDAK, P. Nettle: Taking the sting out of programming network routers. *Practical Aspects of Declarative Languages*, Springer, p. 235–249, 2011.

VOELLMY, A.; KIM, H.; FEAMSTER, N. Procera: a language for high-level reactive network control. In: ACM. *Proceedings of the first workshop on Hot topics in software defined networks*. [S.l.], 2012. p. 43–48.

VOLPATO, F.; SILVA, M. P. D.; GONçALVES, A. L.; DANTAS, M. A. R. An autonomic qoe-aware management architecture for software-defined networking. In: *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. [S.l.: s.n.], 2017. p. 220–225.

WANG, J. M.; WANG, Y.; DAI, X.; BENSAOU, B. Sdn-based multi-class qos guarantee in inter-data center communications. *IEEE Transactions on Cloud Computing*, IEEE, v. 7, n. 1, p. 116–128, 2015.

WANG, Z.; CROWCROFT, J. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on selected areas in communications*, IEEE, v. 14, n. 7, p. 1228–1234, 1996.

WHITE, S. R.; HANSON, J. E.; WHALLEY, I.; CHESS, D. M.; KEPHART, J. O. An architectural approach to autonomic computing. In: IEEE. *Autonomic Computing, 2004. Proceedings. International Conference on*. [S.l.], 2004. p. 2–9.

WICKBOLDT, J. A.; JESUS, W. P. D.; ISOLANI, P. H.; BOTH, C. B.; ROCHOL, J.; GRANVILLE, L. Z. Software-defined networking: management requirements and challenges. *IEEE Communications Magazine*, v. 53, n. 1, p. 278–285, January 2015. ISSN 0163-6804.

WODCZAK, M.; MERIEM, T. B.; RADIER, B.; CHAPARADZA, R.; QUINN, K.; KIELTHY, J.; LEE, B.; CIAVAGLIA, L.; TSAGKARIS, K.; SZOTT, S. et al. Standardizing a reference model and autonomic network architectures for the self-managing future internet. *IEEE Network*, IEEE, v. 25, n. 6, 2011.

WONG, J. Queueing network modeling of computer communication networks. *ACM Computing Surveys (CSUR)*, ACM, v. 10, n. 3, p. 343–351, 1978.

XIONG, B.; YANG, K.; ZHAO, J.; LI, W.; LI, K. Performance evaluation of openflow-based software-defined networks based on queueing model. *Computer Networks*, Elsevier, v. 102, p. 172–185, 2016.

YAHIA, I. G. B.; BENDRISS, J.; SAMBA, A.; DOOZE, P. Cognitive 5g networks: Comprehensive operator use cases with machine learning for management operations. In: *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. [S.l.: s.n.], 2017. p. 252–259.

YAO, H.; MAI, T.; XU, X.; ZHANG, P.; LI, M.; LIU, Y. Networkai: An intelligent network architecture for self-learning control strategies in software defined networks. *IEEE Internet of Things Journal*, p. 1–1, 2018. ISSN 2327-4662.

YAO, L.; DONG, P.; ZHENG, T.; ZHANG, H.; DU, X.; GUIZANI, M. Network security analyzing and modeling based on petri net and attack tree for sdn. In: IEEE. *Computing, Networking and Communications (ICNC), 2016 International Conference on*. [S.l.], 2016. p. 1–5.

YEGANEH, S. H.; TOOTOONCHIAN, A.; GANJALI, Y. On scalability of software-defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 136–141, February 2013. ISSN 0163-6804.

YEGANEH, S. H.; TOOTOONCHIAN, A.; GANJALI, Y. On scalability of software-defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 136–141, 2013. ISSN 01636804.

YU, C.; LUMEZANU, C.; ZHANG, Y.; SINGH, V.; JIANG, G.; MADHYASTHA, H. V. Flowsense: Monitoring network utilization with zero measurement cost. In: SPRINGER. *International Conference on Passive and Active Network Measurement*. [S.l.], 2013. p. 31–41.

ZHOU, K.-Q.; ZAIN, A. M. Fuzzy petri nets and industrial applications: a review. *Artificial Intelligence Review*, v. 45, n. 4, p. 405–446, Apr 2016. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-015-9451-9>.

ZHOU, T.; XIANGYANG, G.; HU, Y.; QUE, X.; WENDONG, W. Pindswitch: A sdn-based protocol-independent autonomic flow processing platform. In: *2013 IEEE Globecom Workshops (GC Wkshps)*. [S.l.: s.n.], 2013. p. 842–847. ISSN 2166-0077.

ZHOU, W.; LI, L.; LUO, M.; CHOU, W. Rest api design patterns for sdn northbound api. In: IEEE. *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*. [S.l.], 2014. p. 358–365.

APÊNDICE A – REGRAS DE VALIDAÇÃO

Como forma de exemplificar a maneira que ocorre a validação dos modelos, exibimos o exemplo abaixo. As demais regras de validação utilizadas em nossa abordagem podem ser obtidas em <https://git.io/vQ0bj>.

Algoritmo 4: Exemplo de restrição OCL usada para validar os modelos.

```
context NetworkNode {
    constraint hasName { // Rule #1
        check : self.name.isDefined()
        message : 'Unnamed ' + self.eClass().name.
                    toUpperCase() + ' not allowed'
    }
    fix {
        title : 'Define the node name'
        do {
            var type := UserInput.prompt('What is
                        the name?');
            if (type.isDefined()) self.type := type;
        }
    }
}
```

APÊNDICE B – TEMPLATES DE GERAÇÃO DE CÓDIGO

A geração de código dos modelos de aplicação e de objetivos ocorre através da utilização de *templates*. Um exemplo destes *templates* está descrito abaixo, no método que verifica a compatibilidade entre uma aplicação de monitoramento e a infraestrutura de rede, além de iniciar o processo de delegação de fluxo, caso necessário. Os demais templates podem ser obtidos em: <https://git.io/vQ0bj>.

Algoritmo 5: Template usado para gerar a delegação de fluxo em cenários de switches incompatíveis.

```
      @set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
      def handler_datapath(self, ev):
  [% /*Verifica aplicações, protocolos
       Verifica se o switch alvo possui suporte
          Se possuir: DP.ID -> aprendizado de rota normal
          Se não: DP.ID -> inundação ou traffic shapping*/ %]
  [%
  var supportedProcotols : String;
  var supportedProcotols2 : String;
  supportedProcotols2 = "";
  %]
  [% for(monitor in AppMonitor.all) {
      for (s in OpenFlowSwitch.All) {
          if (monitor.targetSwitch.supportedProtocols.concat().
          toLowerCase().contains(monitor.type.getName().toLowerCase())
              or monitor.targetSwitch.supportedProtocols.concat().
              toLowerCase() == monitor.type.getName().toLowerCase()){ %]
          if ev.enter and ev.dp.id == [%=s.id%]:
              print('mirroring flows from dpid', ev.dp.id)
              self.switch_to_monitor.append(ev.dp.id)
              self._define_flow_rule_to_mirror_and_monitor(ev.dp)
  [%      }
          else {%]
          if ev.enter and ev.dp.id == [%=s.id%]:
              print('handle mirrored flow in dpid', ev.dp.id)
              self._handle_mirrored_flow(ev.dp)
        [%}
      }
  } %]
1
```

APÊNDICE C – ALGORITMOS UTILIZADOS

C.1 SOURCE-ROUTING ALGORITHM

Este apêndice exibe o algoritmo utilizado para realizar a obtenção de rotas baseadas em restrições QoS definidas nos modelos de objetivos ou aplicações.

Algoritmo 6: Obtendo caminho baseado em QoS

```
1  Function getQoSPath(dp_id, dst_id, qos, [d, bw]optional) : path is
2      i, j ← getSwitchIds()
3      delay_list ← getDelayForAll(d)
4      bandwidth_list ← getBandwidthForAll(bw)
5      for d_{i,j} in delay_list and b_{i,j} in bandwidth_list do
6          if b_{i,j} < qos_bandwidth then
7              if releaseBandwidth(b_{i,j}) = false then  d_{i,j} = MAX
8          end
9      end
10     P ← {dp_id}
11     for i ≠ dp_id do
12         D_i = b_{dpid,i}
13     end
14     k ← i
15     for k not in P do
16         D_k = min_{i∉P} D_i if D_k > qos_delay then
17             if balanceTraffic(k, qos_delay) = false then  return false
18         end
19         else if dst_id in P then  return P
20         else
21             P ← P ∪ {k}
22         end
23         for i not in P do
24             D_i ← min[D_i, D_k + d_{k,i}]
25         end
26     end
27 end
```