MICHAEL PRIETO HERNANDEZ

**FRAMEWORK PARA ENGENHARIA DE TRÁFEGO
EM REDES DEFINIDAS POR SOFTWARE
UTILIZANDO NETWORK CALCULUS**

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para obtenção
do Título de Mestre em Ciências.

São Paulo
2017

MICHAEL PRIETO HERNANDEZ

**FRAMEWORK PARA ENGENHARIA DE TRÁFEGO
EM REDES DEFINIDAS POR SOFTWARE
UTILIZANDO NETWORK CALCULUS**

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para obtenção
do Título de Mestre em Ciências.

Área de Concentração:

Engenharia de Computação

Orientador:

Profa. Dra. Regina Melo Silveira

São Paulo
2017

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 31 de agosto de 2017.

Assinatura do autor

Assinatura do orientador

FICHA CATALOGRÁFICA

Hernandez, Michael Prieto

Framework para Engenharia de Tráfego em Redes Definidas por Software utilizando Network Calculus/ M. Prieto Hernandez. – versão corr. – São Paulo, 2017.

103 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

1. SDN #1. 2. Engenharia de Tráfego #2. 3. Network Calculus #3. 4. QoS #4. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais (PCS). II. t.

AGRADECIMENTOS

Gostaria de agradecer a minha esposa Ariagna, por ser minha fiel companheira durante todos esses anos. Por ter me dado forças nos momentos difíceis, e especialmente nesta minha etapa de estudante. Te amo "mami"! Aos meus pais, quero agradecer pela vida e por permitir-me ser livre. Ao meu irmão Ulises, por ser uma parte fundamental da minha existência. Aos meus amigos latino-americanos por fazer a vida de um imigrante um pouco mais prazerosa. Aos meus amigos, pelos quais prezo pela sua amizade mesmo com a distância. A minha orientadora, professora Regina, por ter me dado a oportunidade de crescer profissionalmente, e pela tolerância com todos os problemas que acompanham um Cubano pelo mundo. Desejo agradecer os meus colegas de trabalho, que alguns são mais que isso, por todo o apoio incondicional e sincero. Quero agradecer a "mi cosita de papá", Galilea, que chegou a este mundo para me fazer um pai muito feliz! Gostaria de agradecer também a professora Cíntia Borges Margi pelo apoio na primeira etapa do curso de mestrado e a Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP). Um agradecimento especial para a equipe da empresa TV on Demand Consultoria e Soluções, que foi e continua sendo de grande ajuda para nossa família.

AGRADECIMIENTOS

Quiero agradecer a mi esposa Ariagna, por ser mi fiel compañera durante todos estos años. Que me ha dado fuerzas en los momentos más difíciles, y especialmente en esta etapa de estudiante. Te amo mami! A mis padres quiero agradecerles por la vida y por permitirme ser libre. A mi hermano Ulises, porque forma parte fundamental de mi existencia. A mis amigos latinoamericanos por hacer la vida de emigrante um poco más placentera. A mis amigos, que me cuidan y los cuido en la distancia. A mi orientadora, la professora Regina, por darme la oportunidad de crecer profesionalmente, y por la tolerancia con los problemas que acompañan a un cubano por el mundo. Deseo agradecer a mis colegas de trabajo, que algunos ya son más que eso, por el apoyo desinteresado y sincero. Quiero agradecer a mi cosita de papá, Galilea, que llegó a este mundo para hacerme un hombre feliz! Me gustaría agradecer también a la profesora Cintia Borges Margi por el apoyo en la etapa inicial del curso de maestría y a la Fundación de Amparo a la Pesquisa del estado de São Paulo (FAPESP). Quiero dedicar um agradecimiento especial al equipo de la empresa TV on Demand consultaría y Soluciones, la cual fue y continua siendo de gran ayuda para nuestra familia.

RESUMO

A demanda de eficiência no gerenciamento das redes aumenta atualmente devido ao rápido desenvolvimento da computação em nuvem e a implantação em grande escala de centros de dados. O controle da infraestrutura deve ser capaz de classificar os diversos tipos de tráfego para aplicações diferentes, e prover o atendimento adequado no menor tempo possível. Porém, devido ao dinamismo da rede, nem sempre podem ser garantidos os requisitos mínimos necessários na Internet. Os mecanismos propostos para a engenharia de tráfego até hoje, são baseados em arquiteturas inflexíveis com a camada de controle e dados fortemente integrados. Isto dificulta um atendimento diferenciado adaptável aos diversos padrões de tráfego das aplicações modernas. Para enfrentar os problemas de flexibilidade das redes tradicionais, as Redes Definidas por Software, apresentam um paradigma inovador que separa as camadas de controle e encaminhamento de pacotes de forma a garantir a utilização eficiente dos recursos disponíveis, e ao mesmo tempo maior flexibilidade de implementação. Este paradigma permite conhecer o estado da rede e a sua topologia em tempo real, o que faz possível a reconfiguração de rotas e alocação de recursos de forma dinâmica. Esta dissertação apresenta um *framework* para engenharia de tráfego em SDN que utiliza a teoria matemática *Network Calculus* como ferramenta para subsidiar a caracterização e o policiamento de fluxos de pacotes. Através desta teoria, as decisões de encaminhamento e a distribuição do tráfego são baseadas não só por valores obtidos monitorando a rede, como também por *projeções determinísticas* que descrevem o comportamento do tráfego. Os resultados obtidos nos experimentos, comprovaram a eficiência no balanceamento de carga da rede em termos de atraso, demonstrando ganhos em termos de vazão do sistema e possibilitando a diminuição da porcentagem de perda de pacotes dos fluxos trafegados. O *framework* proposto visa contribuir na solução dos desafios relacionados a se estabelecer Engenharia de Tráfego para SDN com a especificação de mecanismos de gerenciamento adaptáveis as mudanças topológicas da rede, as diferentes características dos fluxos e que sejam capazes de distribuir de forma equilibrada o tráfego na rede.

ABSTRACT

The demand for network management efficiency is currently boosted by the rapid development of cloud computing and the large-scale deployment of data centers. Infrastructure control should be able to classify the various types of traffic for different applications, and provide adequate service in the shortest time possible. However, due to the dynamism of the network, the necessary minimum requirements on the Internet can not always be guaranteed. The proposed mechanisms for traffic engineering to date are based on inflexible architectures with the control and data layer strongly integrated. This hinders a differentiated service adaptable to the different traffic patterns of modern applications. To address the flexibility problems of traditional networks, Software Defined Networks presents an innovative paradigm that separates packet control and datapath layers to ensure the efficient use of available resources while providing greater deployment flexibility. This paradigm allows to know the state of the network and its topology in real time, which makes it possible to reconfigure routes and allocate resources dynamically. This work presents a framework for SDN traffic engineering that uses the mathematical theory Network Calculus as a tool to subsidize the characterization and policing of packet flows. Through this theory, routing decisions and traffic distribution are based not only on values obtained by monitoring the network, but also on deterministic projections of traffic behavior. The results obtained in the experiments, proved the efficiency in the load balancing of the network based on flow delay, demonstrating gains in terms of total system throughput and enabling the reduction of the percentage of packet loss of the traffic flows. The framework proposed aims to contribute to the solution of the challenges related to establishing Traffic Engineering for SDN with specification of management mechanisms, adaptable to the topological changes of the network, the different characteristics of the flows and capable of distribute in a balanced way the traffic in the network.

SUMÁRIO

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

Lista de Símbolos

1	Introdução	16
1.1	Motivação	19
1.2	Objetivos	20
1.2.1	Objetivo geral	20
1.2.2	Objetivos específicos	20
1.3	Justificativa	21
1.4	Organização	22
2	Estado da arte	23
2.1	Qualidade de Serviço (QoS)	26
2.2	Engenharia de Tráfego (TE)	29
2.2.1	Resiliência	31
2.2.2	Balanceamento de carga	32
2.3	Considerações finais	34

3	Definições Teóricas	36
3.1	Redes Definidas por Software	36
3.2	Qualidade de Serviço	40
3.2.1	OpenFlow e QoS	42
3.3	Engenharia de tráfego	43
3.4	Network Calculus	44
3.4.1	Representação e análise do modelo da rede	48
4	Proposta	53
4.1	Componente para atendimento diferenciado	54
4.2	Componente para admissão de fluxos	56
4.3	Componente para o balanceamento de carga	59
4.3.1	Exemplo ilustrativo	62
4.4	Componente para a resiliência	66
4.5	Modelagem e implementação	67
4.5.1	Requisitos	68
4.5.2	Estruturas	69
4.5.2.1	Modelo de classes principais que sustentam o <i>framework</i> proposto nesta dissertação.	70
4.5.2.2	Componente para o atendimento diferenciado: . . .	71
4.5.2.3	Componente para admissão de fluxos:	71
4.5.2.4	Componente para o balanceamento de carga:	72

4.5.2.5	Componente para resiliência:	72
5	Resultados	75
5.1	Considerações iniciais	75
5.2	Experimentos	77
5.2.1	Balanceamento de carga	80
5.2.1.1	Ambiente virtual e cenário de teste	81
5.2.1.2	Discussão	85
5.2.2	Resiliência	87
5.2.2.1	Discussão	90
5.2.3	Aspectos de desempenho	91
5.3	Conclusão	93
6	Conclusões	95
	Referências	98

LISTA DE ILUSTRAÇÕES

1	Engenharia de tráfego desde o passado até o futuro. (AKYILDIZ et al., 2014)	29
2	Escopo das linhas de pesquisa sobre engenharia de tráfego em SDN. (AKYILDIZ et al., 2014)	30
3	Arquitetura de uma rede definida por software	37
4	Sequência de operações no comutador <i>OpenFlow</i>	38
5	Descoberta da topologia com LLDP em SDN	38
6	Controlador SDN ONOS	39
7	Interfaces Open vSwitch, (PFAFF; DAVIE, 2013)	43
8	Network Calculus, representação gráfica	45
9	Representação tradicional de redes e o modelo utilizado por NetCalc .	48
10	Modelo de rede através de NetCalc	50
11	Controlador <i>Leaked-Bucket</i>	51
12	<i>Framework</i> para a engenharia de tráfego em redes definidas por software utilizando <i>Network Calculus</i>	53
13	Atendimento diferenciado nos canais de saída dos nós da rede	55
14	Modelo de rede com atendimento diferenciado	56
15	Particionamento da rede para o atendimento de diversas classes de fluxos	57
16	Sequência de atividades realizadas pelo componente de admissão . . .	58
17	Características do algoritmo de Dijkstra utilizando NetCalc	60

18	Balanceamento de carga	62
19	Cenário hipotético de uma rede com fluxos de vídeo e VoIP	63
20	Reserva de recursos por classe de fluxos	63
21	Funcionamento por iterações do Algoritmo 3	65
22	Mecanismo de restauração	66
23	Eventos que ativam o componente de resiliencia	67
24	Diagrama de classes	70
25	Diagrama de classes do componente para o atendimento diferenciado	71
26	Diagrama de classes do componente para a admissão de fluxos	72
27	Diagrama de classes do componente para o balanceamento de carga	73
28	Diagrama de classes do componente para a resiliência	73
29	Diagrama de componentes	74
30	Ambiente para a automatização de testes em SDN	78
31	Aplicações instaladas no controlador ONOS	79
32	Rede SDN em termos de curvas de serviço	79
33	Perfis de tráfego cadastrados no controlador	80
34	Topologia Fat-Tree k=4	82
35	Algoritmo proposto em (FIZI; ASKAR, 2016)	83
36	DSDN-TE	84
37	Algoritmo proposto em (FIZI; ASKAR, 2016)	84
38	DSDN-TE	85
39	Vazão da carga inicial	86

40	Curva construída a partir de h_2 e h_3	87
41	Topologia fat-tree $k=4$ com enlaces adicionais	88
42	Resiliência do sistema com DSDN-TE	89
43	Resiliência do sistema com DSDN-TE	89
44	Comportamento do sistema com diferentes topologias	91
45	Topologia fat-tree $k=6$	92

LISTA DE TABELAS

1	Configuração dos enlaces e os serviços oferecidos	63
2	Modelagem da rede para fluxos de vídeo aplicando NetCalc	65
3	Modelagem dos fluxos de vídeo aplicando NetCalc	65
4	Características dos equipamentos utilizados no experimento	77
5	Versionamento dos softwares e protocolos utilizados no experimento .	77
6	Nós responsáveis pela geração da carga inicial na rede	82
7	Aumento da vazão com a utilização de DSDN-TE	84
8	Perdas experimentadas pelos fluxos com a utilização de DSDN-TE . .	84
9	Aumento da vazão com a utilização de DSDN-TE	89
10	Aumento das perdas com a utilização de DSDN-TE	89

LISTA DE ABREVIATURAS E SIGLAS

ATM	Asynchronous Transfer Mode
DiffServ	Serviços Diferenciados
DSCP	Ponto de código de serviços diferenciados
FML	Flow-based management language
IBN	Intent Based Network
IETF	Internet Engineering Task Force
IntServ	Serviços Integrados
IoT	Internet das coisas
IP	Protocolo de Internet
IS-IS	Intermediate System to Intermediate System
ISP	Provedores de Serviços de Internet
MINA	Multinetwork INformation Architecture
MPLS	Multi Protocol Label Switching
NetCalc	Network Calculus
Netconf	Protocolo de Configuração de Rede
NOS	Sistema operacional de rede
OSPF	Open Shortest Path First
OVSDB	Open vSwitch Database
QoS	Qualidade de Serviço

RSVP	Protocolo de Reserva de Recursos
SDN	Redes Definidas por Software
TCP	Protocolo de controle de transmissão
USP	Universidade de São Paulo
TE	Engenharia de Tráfego

LISTA DE SÍMBOLOS

- $\otimes$ Operador que indica a convolução de duas funções
- $\oslash$ Operador que indica a deconvolução de duas funções
- $\wedge$ Operador que indica o ínfimo de um conjunto de valores

1 INTRODUÇÃO

As aplicações na Internet de hoje requisitam que o comportamento da infraestrutura de rede seja em tempo real e que escale proporcionalmente ao grande volume de tráfego. O controle da rede deve ser capaz de classificar os diversos tipos de tráfego para aplicações diferentes, e prover o atendimento adequado no menor tempo possível. A demanda de eficiência no gerenciamento das redes aumenta devido ao rápido desenvolvimento da computação em nuvem e a implantação em grande escala de centros de dados (ou *datacenters*).

A distribuição de vídeo dentro do contexto atual tem se tornado cada vez mais frequente, e algumas previsões indicam que até o final desta década oitenta e dois por cento do tráfego da Internet será constituído por fluxos de vídeo (CISCO, 2016). Considerando as previsões de expansão de consumo desta mídia é necessário desenvolver mecanismos nas redes que permitam viabilizar soluções que apresentem bom desempenho, assim como robustez e escalabilidade.

Visando garantir QoS fim-a-fim, o Internet Engineering Task Force (IETF) propôs dois modelos, Serviços Integrados (IntServ) (BRADEN; CLARK; SHENKER, 1994) e Serviços Diferenciados (DifServ) (BLAKE; MICROSYSTEMS; WANG, 1998) os quais não foram satisfatoriamente implementados globalmente até hoje devido à sua complexidade e falta de escalabilidade (NI, 1999). Embora o IETF tenha criado o mecanismo de transporte MPLS (Multi Protocol Label Switching) (ROSEN; VISWANATHAN; CALLON, 2001) que possibilita a associação de requisitos de QoS, baseado nos rótulos carre-

gados pelos pacotes, este carece de reconfiguração e adaptabilidade em tempo real (EGILMEZ; DANE, 2012).

O roteamento baseado nos pesos dos enlaces de rede com OSPF/IS-IS (J.MOY, 1998), (SMIT; LI, 2004), não possui a flexibilidade suficiente para dividir de forma eficiente o tráfego entre múltiplos caminhos de igual custo. *Equal-Cost Multi-Path* (ECMP) visa a distribuição uniforme na associação de um fluxo de dados em caminhos de igual custo, utilizando para isso uma função *hash*. Porém, ECMP não considera a disponibilidade dos recursos nem as características dos fluxos, resultando em colisões e sobrecarga dos equipamentos os quais levam a uma degradação da utilização dos recursos de infraestrutura de rede (AKYILDIZ et al., 2014).

Devido às limitações da Internet atual em prover conexão com QoS e, consequentemente, em prover qualidade de experiência (QoE) compatível com as expectativas do usuário, os mecanismos de QoS continuam com grande foco, principalmente em se tratando de mecanismos que atendam as necessidades de sistemas autônomos como as redes dos Provedores de Serviços de Internet (ISP) ou datacenters.

Todos os mecanismos para a engenharia de tráfego (TE) até hoje propostos, são baseados em arquiteturas de redes inflexíveis com a camada de controle e dados fortemente integrados. Isto dificulta um atendimento diferenciado adaptável aos diversos padrões de tráfego das aplicações modernas (AKYILDIZ et al., 2016).

Com o surgimento das Redes Definidas por Software (SDN) (ONF, 2012b), é possível especificar a partir da implementação por linguagem de programação o comportamento de um comutador de pacotes ou *switch*, o que diminui custos e ao mesmo tempo aumenta a flexibilidade do processo de encaminhamento de pacotes IP. Esta abordagem permite criar mecanismos que, através do monitoramento da rede, forneçam um controle ativo da mesma para superar problemas de limitações de recursos e congestionamentos.

Uma das implementações de maior repercussão de SDN é o protocolo *OpenFlow* (MCKEOWN et al., 2008), mantido atualmente pela Fundação Open Networking. Este protocolo é uma implementação de código aberto que tem sido utilizado pela comunidade acadêmica e por corporações para resolver diversos problemas em suas redes. SDN através de *OpenFlow* centraliza o controle e padroniza a comunicação entre os comutadores e o controlador da rede, diminuindo a complexidade tanto na borda como no núcleo da rede.

As características únicas do paradigma SDN como a programabilidade e o fato de ser uma especificação aberta, oferece oportunidades para o desenvolvimento de novas técnicas de engenharia de tráfego flexíveis, adaptáveis e customizáveis.

No entanto, avaliar e analisar o comportamento e desempenho de uma rede não é uma tarefa simples. A teoria clássica de filas trata geralmente de valores médios dentro de sistemas em estado de equilíbrio. Esta teoria introduzida com a publicação do trabalho “*The theory of probabilities and telephone conversations*” no ano 1909 pelo matemático A.K. Erlang, tem sido fundamental na modelagem, análise e dimensionamento das redes de comunicação. Com o avanço das novas tecnologias de redes e a evolução da Internet, a teoria de filas tem sido aplicada também nas redes de computadores (JIANG, 2009). Porém, as características únicas e os diversos requisitos dos serviços e consumidores na Internet dificultam a aplicação desta teoria. Devido a isto surgiu no início dos anos 90 uma nova teoria que descreve o comportamento das redes de computadores chamada *Network Calculus* (NetCalc) (CRUZ, 1991b; CRUZ, 1991a).

Através de NetCalc a rede é abstraída em termos de *curvas de chegadas*, as quais caracterizam os fluxos de pacotes e as *curvas de serviço*, que descrevem o comportamento das portas de saída dos equipamentos de rede. Com o mapeamento de uma rede em termos de curvas de chegada e curvas de serviço, o NetCalc, possibilita a obtenção de métricas de QoS como o atraso que experimenta um fluxo de pacotes na rede, assim como o dimensionamento da utilização das filas de saída nas portas dos equipamentos

da infraestrutura.

1.1 Motivação

Na literatura, a maioria dos trabalhos científicos têm como objetivo o desenvolvimento da arquitetura SDN e dedicam menos esforços na especificação de mecanismos de TE. Os métodos atuais já foram intensamente estudados em redes ATM assim como em redes IP baseadas em MPLS, porém não está documentado o desempenho destes métodos considerando diferentes padrões de tráfego e como obter o grande volume de informações dos fluxos e recursos disponíveis na rede de forma eficiente na implantação de uma SDN (AKYILDIZ et al., 2016).

Do ponto de vista de TE e QoS, SDN elimina limitações chaves que não permitiram que os mecanismos tradicionais fossem implantados até hoje. Algumas das características de SDN que facilitam a implementação de mecanismos de QoS e TE são (ONF, 2012b):

- Conhecimento da topologia de rede em tempo real;
- Centralização do algoritmo de roteamento que trabalha sobre a topologia;
- Implantação de forma padronizada das políticas de roteamento sobre a topologia de rede;
- Padronização dos atendimentos para diferentes classes de fluxos no percurso entre ilhas SDNs.

Considerando uma rede de pacotes implementada de forma tradicional a teoria clássica de filas, assim como NetCalc, podem ser utilizadas como ferramentas para a simulação e o planejamento estático da infraestrutura de rede. Porém estas teorias

não poderiam ser utilizadas nas decisões de roteamento em tempo real ou na distribuição equilibrada do tráfego, devido á falta de programabilidade e visão global que caracterizam este tipo de redes.

Com a utilização de SDN surgem oportunidades da geração de uma matriz de tráfego construída através de definições analíticas (e.g., NetCalc) que leve em consideração as características de cada fluxo com independência da capacidade de monitoração da infraestrutura. Isto contribuirá no planejamento e recuperação de forma dinâmica da rede, assim como na utilização eficiente dos recursos disponíveis.

1.2 Objetivos

A seguir são definidos os objetivos geral e específicos deste trabalho.

1.2.1 Objetivo geral

Esta dissertação tem como objetivo analisar a eficiência da utilização de uma abordagem analítica baseada na teoria matemática de Network Calculus para engenharia de tráfego em redes definidas por software.

1.2.2 Objetivos específicos

Para alcançar o objetivo proposto será especificado e implementado um *framework* composto por quatro componentes. Estes componentes contemplarão três aspectos chaves da qualidade de serviço em redes IP, que são o controle de tráfego, o escalonamento (*scheduling*) e o controle de admissão. Tais aspectos darão subsídio a técnicas de TE como o balanceamento de carga e a resiliência. Os componentes a serem especificados são os seguintes:

- Componente para o atendimento diferenciado de fluxos;

- Componente para o controle da admissão de fluxos;
- Componente para o balanceamento de carga;
- Componente para a resiliência.

1.3 Justificativa

A utilização de uma mesma infraestrutura de rede para atender vários tipos de serviços (Tv, Telefonia, VoIP, VoD, Web, email, vídeo conferência, mensageiro instantâneo, etc.) e o crescente número de dispositivos conectados são desafios que devem ser enfrentados na Internet moderna.

O grau de sensibilidade dos fluxos de dados trafegados às métricas de QoS como o atraso, a variação de atraso, a largura de banda e a perda de pacotes, varia de um tipo de fluxo para outro. A falta de mecanismos que consigam lidar com a heterogeneidade da Internet de hoje demanda a criação e implantação de novos paradigmas.

As análises probabilísticas através da teoria de filas determinam comportamentos a longo prazo, por este motivo elas não parecem adequadas para fornecer garantias por fluxos de dados em termos de métricas de QoS. Assumindo esta hipótese como verdadeira exploramos aqui uma teoria determinística (NetCalc), visando garantir níveis máximos de eficiência para que seja possível provisionar aos fluxos de dados os recursos de rede necessários.

Com o intuito de alcançar maior flexibilidade, melhor controle e menor custo de implantação será utilizado neste trabalho a rede definida por software. SDN, pelo fato de ser um novo paradigma, carece de pesquisas que explorem os desafios relacionados aos mecanismos de controle de QoS e técnicas de TE. A utilização do SDN nos permitirá obter um gerenciamento mais ativo da rede e, conseqüentemente, a utilização mais eficiente da mesma.

Persistem na literatura desafios relacionados a se estabelecer Engenharia de Tráfego para SDN com a especificação de mecanismos de balanceamento de carga dinâmicos para a camada de dados. Tais mecanismos devem ser adaptáveis as mudanças topológicas da rede e considerar características de tráfego precisas como as rajadas e a taxa de transmissão. Outros desafios atuais são encontrados na recuperação eficiente também na camada de dados. Esta recuperação as falhas deve ser implementada de forma a minimizar/eliminar a comunicação com o controlador SDN e diminuir o requisito de inteligência nos comutadores da rede (AKYILDIZ et al., 2016).

1.4 Organização

Esta dissertação apresenta no Capítulo 2 vários trabalhos relacionados com esta pesquisa, ressaltando a nossa contribuição. No Capítulo 3 são descritos os principais conceitos relacionados com as SDNs, além do seu funcionamento e arquitetura. Posteriormente são mencionados os mecanismos fundamentais de QoS existentes até hoje e como reutilizamos algumas características destes métodos. Ainda neste capítulo, detalha-se o conjunto de definições da teoria de NetCalc utilizadas na elaboração do *framework* proposto, e é feita a modelagem de uma rede fazendo uso desta teoria. O Capítulo 4, detalha a proposta do *framework*. Os experimentos realizados, assim como os resultados obtidos, são apresentados no Capítulo 5. As conclusões são apresentadas no Capítulo 6.

2 ESTADO DA ARTE

A empresa Google publicou em 2012 um trabalho no qual expõem experiências na aplicação do conceito e arquitetura de SDN. A empresa mostrou no evento Open Network Summit (GOOGLE, 2012) uma implementação em grande escala de uma rede baseada em SDN para conectar seus *datacenters*). Em (JAIN et al., 2013) são apresentados os detalhes de implementação desta iniciativa chamada B4. A motivação foi a necessidade de customizar o roteamento e a engenharia de tráfego, e o fato de que o nível de escalabilidade, a tolerância à falhas, a eficiência e o controle necessários, não podem ser atendidos através da arquitetura tradicional de uma rede de longa distância. SDN é considerada uma tecnologia que trará estes benefícios para a implantação e operação de redes, principalmente em alguns cenários específicos como em *datacenters*.

Por outro lado o paradigma de redes definidas por software propõe o gerenciamento centralizado da rede impactando significativamente na escalabilidade da infraestrutura. O trabalho de (YEGANEH; TOOTOONCHIAN; GANJALI, 2013) apresenta diversas deficiências neste sentido e discute alternativas do modelo arquitetônico das SDNs.

O modelo centralizado é uma abordagem onde toda a inteligência da rede está em um único controlador (i.e., Maestro (NG, 2010), Beacon (ERICKSON, 2013)) , permitindo que as informações, estatísticas e comandos de controle sejam gerenciados por um único elemento de rede que fornece granularidade e confiabilidade. Porém, este modelo não leva em consideração outros controladores de rede que possam servir de apoio no caso do controlador principal não conseguir atender a carga da rede, podendo

deixar toda a infraestrutura inutilizável.

No entanto, o modelo distribuído permite um gerenciamento logicamente centralizado, onde as informações de rede (i.e., topologia, estatísticas) são compartilhadas entre controladores distintos (i.e., ONOS (BERDE et al., 2014), OpenDayLight (OPENDAYLIGHT, 2013)), eliminando o risco da presença de um único ponto de falha na rede. Porém, problemas de ambiguidade e informações de rede duplicadas ou desatualizadas podem surgir (NUNES et al., 2014).

A implementação de controladores de rede que abstraíam as características específicas dos equipamentos, e forneçam de maneira transparente funcionalidades em comum, é até hoje quase ausente nas redes. Atualmente programadores de protocolos de roteamento precisam lidar com a complexidade de algoritmos distribuídos, tentando resolver os problemas da infraestrutura (KREUTZ; RAMOS, 2014).

Existem ao menos três camadas bem definidas em quase todas as plataformas de controle (e.g., controlador SDN): (i) camada de aplicações e serviços (*northbound*), (ii) camada de funções básicas nativas e (iii) camada de interação com dispositivos físicos (*southbound*).

A especificação da camada superior é baseada em interfaces através de *APIs RESTful* (VINOSKI, 2008) e linguagens de programação como Flow-based Management Language (FML) (HINRICHS et al., 2009), Frenetic (FOSTER et al., 2011) e NetCore (MONSANTO et al., 2012). A camada inferior dos controladores SDN está definida por APIs e implementações de protocolos em forma de *plugins* capazes de interagir com diversos equipamentos de rede (i.e., comutadores *OpenFlow*). O núcleo do controlador pode ser descrito como uma combinação de serviços de rede e interfaces disponíveis para as camadas superiores (*northbound*) e inferiores (*southbound*).

As abstrações que podem ser obtidas com SDN provêm informações relevantes que as aplicações poderiam utilizar para aperfeiçoar funcionalidades dentro de me-

canismos de QoS ou TE. Estas abstrações fornecem os meios para que as aplicações possam especificar o comportamento desejado da rede sem necessidade de conhecer os detalhes de configuração da mesma. O desafio está no desenvolvimento correto das abstrações para que possam ser úteis a um grande número de aplicações de distintas classes.

Intent Based Network (IBN) é um modelo arquitetural para SDN que define uma forma de relacionamento entre as aplicações e a infraestrutura da rede, porém mantendo a abstração dos mecanismos de rede para as aplicações. Atualmente em vias de padronização pela IETF, a ideia é prover ao *usuário* uma interface simples de acesso à rede para configurar a mesma de acordo com suas necessidades, ou melhor, de suas aplicações. Isso significa um passo além da configuração de rede via *Netconf* (ENNS et al., 2011) ou *OpenFlow*, que atualmente não é de maneira nenhuma simples ao usuário comum.

IBN deve ser entendido como descritivo, isto é, faça o que for possível para entregar este pacote de A para B, em vez de prescritivo, isto é, faça esta ação especificamente, roteie de A para B, então de B para C, etc, para este pacote que entra na porta X. Segundo (GUTIÉRREZ; LÓPEZ, 2016) do IEEE, IBN deve ser “human-understandable” a fim de simplificar e facilitar a interação do usuário final com a rede. Na atualidade vários projetos estão focados na especificação e implementação do IBM, com o intuito de fornecer melhores abstrações para as aplicações de rede.

ONOS Intents (ON.LAB, 2016) é um exemplo de framework que expressa um “desejo” sobre o comportamento da rede na forma de política ao invés de descrever esta política explicitamente através de mecanismos que vão implementá-la. Por outra parte no projeto OpenSourceSDN Boulder (OPENSOURCESDN,) as interfaces NBIs são definidas independentemente do controlador SDN e seu foco está na semântica e em modelos de dados. Neste sentido a implementação OpenStack (SEFRAOUI; AISSAOUI; ELEULDJ, 2012) permite a definição de um grupo de políticas, denominado Group Ba-

sed Policy (GBP), como uma coleção de endpoints que possuem as mesmas políticas, e desta forma podem compartilhar e reutilizar regras de controle. O grupo OpenDay-Light possui dois projetos, o Intent-based Network Modelling (IBNEMO) (ZHANG et al., 2016), que define uma linguagem estruturada para descrição de cenários de rede mais simples e voltada ao usuário, e o Network Intent Composition(NIC) (OPENDAY-LIGHT, 2015) que tem a visão do Intent como um serviço de protocolo de roteamento.

2.1 Qualidade de Serviço (QoS)

Vários trabalhos apresentam o uso de SDN para a otimização da transmissão de fluxos, onde o melhor esforço da Internet não atende os requisitos mínimos em termos de qualidade de serviço.

Uma destas propostas implementa um módulo que é denominado OpenQoS integrado ao controlador da SDN, que utiliza um mecanismo de diferenciação de tráfego. Este módulo permite tratar dinamicamente, de forma diferenciada, os fluxos de vídeo escolhendo a rota mais adequada aos parâmetros de QoS requisitados (EGILMEZ; DANE, 2012).

Em outro trabalho Egilmez propõe a utilização de codificação de vídeo escalável, criando com isso diferentes camadas de qualidade de serviço e priorização, que serão usadas de acordo com as condições da rede (EGILMEZ et al., 2011). A priorização é feita estaticamente no controlador no instante de associar uma rota a um fluxo, e não existe uma alocação de recursos no *switch*.

Outro trabalho (OWENS; DURRESI, 2013) propõe ainda a implantação de um módulo no controlador que manipula métricas de QoS. O protocolo proposto visa garantir o entendimento mútuo entre o transmissor e o receptor no instante da solicitação de um vídeo e a procura de uma rota com qualidade de serviço. O autor propõe modificações ao protocolo *OpenFlow* para implementar a sua solução, acrescentando métricas de

Qualidade de Serviço nas descrições das filas dos comutadores *OpenFlow*.

Qin (QIN et al., 2014) propõe um controlador SDN multicamadas para cenários de Internet das Coisas (IoT) em que são consideradas redes sem fio heterogêneas. A principal contribuição do trabalho é a especificação da arquitetura do controlador, a qual visa possibilitar o gerenciamento eficiente das tarefas requisitadas pelos clientes da rede. Estas tarefas são traduzidas para requisitos de conectividade com diferentes padrões de tráfego levando em consideração os recursos disponíveis (capacidade dos enlaces, routers, comutadores, etc). Qin ainda modifica a teoria de Network Calculus para obter o *jitter* dos fluxos, sendo conhecido o tempo de início de cada fluxo. Os caminhos fim-a-fim de menor custo são calculados utilizando um algoritmo genético, o qual usa NetCalc na definição da função de avaliação (*fitness*) e na fase de *mutação*. O trabalho valida a precisão do NetCalc na obtenção do atraso fim-a-fim, assim como do *jitter*, aplicando a solução proposta e realizando uma comparação dos resultados obtidos através de um simulador de redes (Qualnet) num cenário hipotético com capturas de tráfego *Skype* de diferentes universidades. Finalmente são demonstrados ganhos em termos de métricas de QoS (i.e., vazão, atraso e jitter) na escolha dos caminhos fim-a-fim quando comparados com diferentes algoritmos de agendamento (*Bin Packing* e balanceamento de carga).

Fazendo uma análise crítica mas aprofundada com relação ao trabalho anteriormente explicado, levando em consideração a relevância do mesmo, podemos dizer que com a utilização de um algoritmo genético Qin (QIN et al., 2014) introduz uma estratégia probabilística que depende do número de iterações para convergir em um resultado quase ótimo. Na fase de cruzamento (*crossover*) cada cromossoma gerado necessita ser um caminho válido o qual precisa de processamentos adicionais. Já na fase de mutação o nó gargalo da rota é identificado quando nele é maior o atraso para um fluxo conhecido. Para isso cada nó deve ser analisado de forma individual, aplicando Net-Cal em cada um deles. Os autores não realizam um detalhamento de qual é o impacto

desta análise na performance do algoritmo. Outro aspecto a destacar é que a forma de avaliar os resultados em (QIN et al., 2014) não reflete as consequências das análises anteriormente mencionadas em termos de tempo de execução, pois o agendamento das rotas foi realizado de forma proativa e não em tempo real.

Diferentemente de Qin (QIN et al., 2014), a presente dissertação explora um mecanismo determinístico visando prover QoS de forma garantida (*Guaranteed Service*). Cada rota gerada é um caminho válido usando o algoritmo de Dijkstra sem precisar de processamentos adicionais, e cada um dos nós escolhidos na rota são os mais eficientes em termos de atraso através de uma análise global (SFA) (CRUZ, 1991b) e não individual (TFA) (CRUZ, 1991b), obtendo assim valores mais ajustados (BONDORF; SCHMITT, 2015). Por outra parte o agendamento de fluxos nesta dissertação é realizado em tempo real e de forma reativa. Além disso é provido um mecanismo de admissão o qual garante que os recursos alocados para um fluxo sejam mantidos durante toda a transmissão. Estas duas características são apenas mencionadas como trabalhos futuros em (QIN et al., 2014).

Yan (YAN et al., 2015) propõe a solução HiQoS. Neste trabalho os autores classificam os tráfegos de dados em termos de tipo de aplicação no controlador SDN e criam uma fila de atendimento para cada aplicação nos comutadores *OpenFlow*. Um algoritmo de Dijkstra modificado determina as rotas de encaminhamento baseado em três parâmetros: preço do enlace, distância e largura de banda disponível obtida através do monitoramento da rede. Os autores demonstram o incremento da vazão do sistema com a utilização de múltiplas rotas de encaminhamento. Este elemento faz com que o atraso da transmissão seja otimizado. A solução prevê ainda a recuperação da transmissão em caso de falhas nos enlaces aplicando rotas alternativas determinadas pelo algoritmo Dijkstra. HiQoS não faz caracterização dos fluxos de dados (i.e., taxa de transmissão, rajada, atraso máximo tolerável) nem controle de admissão. As garantias de largura de banda são feitas baseadas em monitoramento periódico com um *threshold*

inserido manualmente.

2.2 Engenharia de Tráfego (TE)

No contexto do SDN, Engenharia de Tráfego passa a ser um tema de grande importância para o bom gerenciamento dos recursos disponíveis na rede. Na literatura alguns poucos trabalhos tratam de questões relacionadas à este tema. (AKYILDIZ et al., 2014) faz um aprofundado estudo no qual expõe a evolução da engenharia de tráfego nas redes de computadores. A Figura 1 mostra os diversos mecanismos para TE propostos até hoje com o novo paradigma SDN. As principais linhas de pesquisa SDN-TE identificadas nesse trabalho são apresentadas na Figura 2.

Figura 1: Engenharia de tráfego desde o passado até o futuro. (AKYILDIZ et al., 2014)

No trabalho (AGARWAL; KODIALAM; LAKSHMAN, 2013) foi desenvolvido um esquema de implantação SDN que pode-se adaptar dinamicamente a diferentes padrões de tráfego. Na pesquisa é considerada a engenharia de tráfego no caso em que o controlador da SDN gerencia só um subconjunto dos equipamentos da rede. O resto da infraestrutura utiliza roteamento tradicional passo-a-passo através de protocolos como OSPF. Os autores desenham um algoritmo que visa rotear o tráfego que passa pelos

comutadores *Openflow* tal que o atraso e a perda de pacotes nos enlaces sejam minimizados. A pesquisa mostra a viabilidade de implantação gradual de SDN em infraestruturas tradicionais, assim como os ganhos que esta poderia oferecer em termos de vazão total do sistema. Os experimentos demonstram melhoras no desempenho do atraso e das perdas de pacotes na rede.

Figura 2: Escopo das linhas de pesquisa sobre engenharia de tráfego em SDN. (AKYILDIZ et al., 2014)

Em (DAVOLI et al., 2015) é apresentada uma solução para a engenharia de tráfego em um ambiente híbrido SDN-IP com roteamento por segmentos (*Segment Routing* (SR)) (FILSFILS et al., 2016), baseado em encaminhamento com MPLS. Conforme a arquitetura exposta em (SALSANO et al., 2016), os autores desenham e implementam um algoritmo que transforma uma rota ótima previamente computada (rota TE) em rotas SR. Foi considerado um cenário hipotético com fluxos caracterizados com chegadas *Poisson* e os links modelados como filas *M/M/1*. O algoritmo em um primeiro passo visa minimizar o tempo de trânsito no sistema dos fluxos considerando a capacidade nominal dos enlaces e a taxa esperada de transmissão dos fluxos. Uma vez encontrada a rota(s) ótima o algoritmo procura mapear esta rota(s) em caminhos SR considerando ainda ECMP. Resultados iniciais de performance mostram o tempo de execução do algoritmo que consegue alocar até 900 fluxos aproximadamente em 8 segundos o qual

consideram adequado para procedimentos de realocação, que visam redistribuir a carga dos enlaces da infraestrutura.

2.2.1 Resiliência

Assumindo que as falhas em uma infraestrutura de rede são fenômenos frequentes (JAIN et al., 2013), a definição de mecanismos que minimizem o impacto de possíveis falhas continuam com grande foco (AKYILDIZ et al., 2014), (AKYILDIZ et al., 2016). Em um contexto SDN os erros de funcionamento podem afetar tanto a camada de controle como a de dados e ambos podem ter uma grande influência na performance da rede.

Devido ao caráter de controle centralizado de uma SDN, as pesquisas realizadas foram direcionadas para a avaliação de desempenho da camada de controle. Estas pesquisas contribuíram ao desenvolvimento de controladores SDN robustos de modelos arquiteturais distribuídos que utilizam tecnologias de clusterização e réplicas de respaldo, tudo isso visando garantir maior escalabilidade e consistência da camada de controle. Enquanto as falhas que possam acontecer na camada de dados, existem dois tipos de mecanismos para recuperação de falhas descritas na literatura: restauração e proteção (SHARMA et al., 2011; AKYILDIZ et al., 2014).

- **Restauração:** Compreende a interação entre o comutador que possui a falha e o controlador de rede de forma reativa. Uma vez acontecida a falha uma mensagem de notificação *OpenFlow* é enviada para o controlador. Rotas alternativas deverão ser consideradas e finalmente instaladas as novas regras de encaminhamento na tabela de fluxos dos comutadores da rota afetada.
- **Proteção:** Compreende a instalação de forma proativa de regras de encaminhamento de respaldo nos comutadores. Neste caso não é enviada uma mensagem *OpenFlow* para o controlador da rede. Uma vez detectada a falha, o comutador passa a utilizar a regra pré-computada de respaldo.

Devido ao fato do mecanismo de proteção não envolver a comunicação com o controlador da rede, este parece ser mais adequado em redes de grande porte (AKYILDIZ et al., 2014). O tempo máximo de recuperação atribuído a este mecanismo é de aproximadamente 60 ms enquanto o mecanismo de restauração leva entre 200-300 ms (AKYILDIZ et al., 2014). No entanto, uma desvantagem do mecanismo de proteção é que não há garantias de que uma vez acontecida a falha a rota alternativa cumpra com a qualidade de serviço requerida pelo fluxo. Devido a este fato um pós-processamento deve ser aplicado.

Em (PADMA; YOGESH, 2015) explora-se um mecanismo de proteção no qual se faz modificações ao protocolo *OpenFlow* versão 1.0, a fim de enviar mensagens anunciando a recuperação do enlace previamente afetado e permitir ao controlador SDN recalcular a rota principal. O mecanismo proposto possui como objetivo reduzir o tempo de recuperação durante a falha de um enlace da rede. O controlador SDN calcula a rota de respaldo e a instala de forma proativa nos comutadores *OpenFlow*, que por sua vez, são habilitados para usar operações de recuperação locais. Isto evita a intervenção do controlador e reduz o processamento nele, poupando o tempo necessário para transmitir as mensagens de falha e definição de novas regras nas tabelas de encaminhamento. Porém, para que isso funcione é necessário modificar a implementação do software *Openvswitch* (PFAFF et al., 2009) nos comutadores. Os resultados demonstram que foi necessário um tempo de 50 ms para recuperar a falha de um único enlace. O método utilizado na pesquisa aumenta o número de regras instaladas nas tabelas dos comutadores por cada rota definida, o que poderia degradar o desempenho em redes de maior porte e reduzir a eficiência da largura de banda.

2.2.2 Balanceamento de carga

Em SDN existem dois tipos de tráfego de rede, o tráfego da camada de dados e o tráfego da camada de controle. Nesta dissertação é tratado só o tráfego da camada de

dados. Uma das principais vantagens do balanceamento de carga em SDN é o caráter centralizado do cálculo para a tomada de decisões de encaminhamento. Isto possibilita considerar os níveis de utilização de múltiplos enlaces alternativos, assim como as características dos fluxos com uma visão mais abrangente. Desta forma a estratégia do balanceamento de carga pode ser planejada com maior precisão se a comparamos com uma rede tradicional.

Equal Cost Multi Path (ECMP) é uma solução efetiva para o balanceamento de carga aplicado em SDN (JAIN et al., 2013). Porém esta tecnologia possui uma deficiência, a qual pode ser apreciada quando múltiplos fluxos de larga duração chamados de *fluxos elefantes*, são encaminhados pela mesma rota, resultando em um balanceamento desequilibrado e numa degradação da largura de banda (DIXIT et al., 2013).

Visando solucionar o problema do ECMP vários trabalhos foram publicados (AL-FARES et al., 2010), (CURTIS; KIM; YALAGANDULA, 2011), (BENSON et al., 2011). A ideia principal é primeiramente identificar os *fluxos elefantes* para depois escolher a melhor rota através do controlador SDN.

Hedera (AL-FARES et al., 2010) é um sistema para gerenciamento de tráfego escalável e dinâmico capaz de distribuir os fluxos de forma adaptativa para utilizar com efetividade os recursos disponíveis na rede. Hedera faz uso de múltiplas rotas entre os servidores do *datacenter*, porém detectando e gerenciando a priori os *fluxos elefantes*. Com isso o mecanismo para o escalonamento dos fluxos contém três etapas: (1) Detecção dos *fluxos elefantes* nos comutadores de borda e encaminhamento utilizando ECMP padrão; (2) Coleta periódica de informação dos comutadores e se um fluxo alcançar uma taxa previamente definida, então é marcado como *fluxo elefante*; (3) Estimativa da largura de banda requerida pelos *fluxos elefantes* detectados para depois, levando em consideração a situação atual da rede, calcular as rotas apropriadas.

Devido ao grande *overhead* gerado pela coleta de estatísticas nos comutadores de borda na detecção dos *fluxos elefantes* por Hedera, a proposta denominada Mahout

(CURTIS; KIM; YALAGANDULA, 2011) faz uso dos *hosts* destino no *datacenter* para a monitoração e detecção dos *fluxos elefantes*, utilizando para isso os *sockets* do sistema operacional. Uma vez detectados os *fluxos elefantes* estes são marcados e os comutadores os encaminham para o controlador da rede, que determina as rotas adequadas para este tipo de fluxo.

A proposta MicroTE (BENSON et al., 2011) por sua vez, utiliza as estatísticas fornecidas pelo protocolo *OpenFlow* de forma passiva. Através das variações nas taxas médias da transmissão e as taxas instantâneas dos fluxos, MicroTE é capaz de identificar o *fluxos elefantes* e otimizar o encaminhamento.

Hailong (HAILONG et al., 2014) faz uma comparação entre OSPF, ECMP e ECMP modificado. Este trabalho demonstra que através de modificações no ECMP, que consideram a largura de banda disponível nos enlaces da rede, a vazão total da infraestrutura pode ser aprimorada.

O algoritmo proposto em (FIZI; ASKAR, 2016) leva em consideração a utilização dos enlaces da rede através de estatísticas oferecidas pelo protocolo *OpenFlow* para melhorar a eficiência do ECMP. É demonstrado além de ganhos na vazão total do sistema em 21.9%, uma redução das perdas em 88.2% quando comparado com ECMP.

2.3 Considerações finais

Depois de uma análise aprofundada da literatura relacionada com a engenharia de tráfego em SDN, podemos identificar algumas deficiências que oferecem novas oportunidades de pesquisa. A presente dissertação explora várias destas oportunidades que são classificadas como desafios por trabalhos recentes como em (AKYILDIZ et al., 2016) e (SHU et al., 2016).

Podemos observar que existe uma tendência de uso do protocolo *OpenFlow* para monitorar a rede visando aprimorar as decisões de encaminhamento. Este fato cria uma

forte dependência entre os mecanismos de TE e o desenvolvimento do *OpenFlow*, o qual não parece adequado devido as limitações do protocolo neste sentido.

Outro aspecto que consideramos desafiador, dependendo do cenário, nas soluções fortemente dependentes da monitoração, é que o intervalo de tempo utilizado para atualizar as métricas monitoradas é geralmente de vários segundos. Em caso de que uma falha ocorra neste intervalo, o sistema terá que tomar as decisões baseado em informações possivelmente desatualizadas.

Esta dissertação provê, através de NetCalc, o ferramental analítico necessário para que a distribuição do tráfego possa ser guiada não só por valores obtidos na monitoração da rede, como também por *projeções determinísticas* do comportamento do tráfego. A taxa de transmissão dos fluxos é a métrica que frequentemente observamos como determinante para as decisões de encaminhamento, porém a rajada máxima dos fluxos não é levada em consideração. Este trabalho incorpora a rajada dentro dos cálculos analíticos, a fim de estimar limiares que representem os piores casos de métricas como o atraso e o tamanho das filas de saída nas portas dos comutadores.

Os trabalhos relacionados com TE descritos neste capítulo, utilizam balanceamento de carga dinâmico. Isto é, a distribuição do tráfego inicialmente utilizando ECMP padrão, uma vez que seja observada uma degradação da rede devido a um balanceamento desigual, são aplicadas heurísticas levando em consideração o estado atual da rede para rebalancear as cargas. Esta dissertação difere neste sentido de pesquisas anteriores. O peso dos enlaces, tradicionalmente fixos, são modificados com a entrada de novos fluxos na rede. Isto faz com que caminhos de igual custo só estejam presentes quando as características dos fluxos sejam semelhantes. Neste caso é escolhida qualquer uma das rotas aleatoriamente, no caso contrário o encaminhamento é baseado num grafo no qual o custo de transmissão entre a origem e o destino é único. Esta abordagem permite que a distribuição de novos fluxos que entram na rede seja, a priori, equilibrada e os recursos disponíveis sejam utilizados de forma eficiente.

3 DEFINIÇÕES TEÓRICAS

Este capítulo resume os principais tópicos necessários para a compreensão do *framework* proposto. São destacados aqui os conceitos relacionados com as redes definidas por software e especificamente o protocolo *OpenFlow*. Na sequência é abordado o tema da qualidade de serviço nas redes e algumas funcionalidades adicionadas no protocolo *OpenFlow* visando possibilitar a implementação de mecanismos básicos de QoS. Mais adiante é feita uma introdução na teoria algébrica de *Network Calculus* apresentando as definições desta teoria que serão utilizadas para a caracterização, controle, classificação e distribuição eficiente do tráfego.

3.1 Redes Definidas por Software

Rede definida por software (SDN) (ONF, 2012b) é um novo paradigma que visa criar de uma forma simples redes dinâmicas, gerenciáveis e adaptáveis. A arquitetura SDN observada na Figura 3 define a rede como uma entidade programável e abstrai os serviços e aplicações de rede das camadas inferiores através de padrões abertos como o protocolo *OpenFlow*.

Na arquitetura *OpenFlow*, o controlador da SDN é responsável pelo reconhecimento dos fluxos de pacotes, determinar uma ou várias ações sobre eles e atualizar as tabelas de fluxos nos comutadores. *OpenFlow* fornece um conjunto de estatísticas que permitem que o controlador conheça a topologia e o estado dos nós da rede (comutador *OpenFlow*) em tempo real, possibilitando o tratamento dinâmico de fluxos.

Figura 3: Arquitetura de uma rede definida por software

A figura 4 apresenta a sequência de operações que realiza um comutador *OpenFlow* quando recebe um fluxo de dados. Primeiramente, se a tabela de fluxos não possui uma regra específica para um fluxo, os pacotes são encapsulados em mensagens *OFPT_PACKET_IN* e enviados para o controlador SDN. Uma vez que é recebida a mensagem, o controlador deve ser capaz de decidir as ações a serem feitas com o fluxo e enviar estas decisões em mensagens *OFPT_FLOW_MOD*. O comutador cria novas entradas na tabela de fluxos e são aplicadas as regras. Enquanto se instalam as novas entradas na tabela de fluxos, o controlador SDN pode enviar os pacotes recebidos em mensagens *OFPT_PACKET_IN* para as portas de saída do comutador através de mensagens *OFPT_PACKET_OUT*.

Os parâmetros da topologia de rede são relacionados com o número de comutadores, a capacidade dos enlaces, a estrutura das conexões e estado das portas. Na especificação *OpenFlow* o controlador SDN deve detectar a topologia da rede utilizando *Link Layer Discovery Protocol* (LLDP) (HUANG et al., 2014) e persistir a informação

Figura 4: Sequência de operações no comutador *OpenFlow*

global da rede.

No processo de descoberta dos enlaces, o controlador envia mensagens LLDP encapsulados em *OFPT_PACKET_OUT* para todos os comutadores da rede. Uma vez que um comutador receba pacotes LLDP do controlador, este envia os pacotes para todos os comutadores conectados a ele. Quando os comutadores recebem pacotes LLDP de outros comutadores, estes enviam os pacotes em forma de mensagens *OFPT_PACKET_IN* para o controlador como pode ser observado na Figura 5.

Figura 5: Descoberta da topologia com LLDP em SDN

Através destas mensagens *OFPT_PACKET_IN* contendo pacotes LLDP, o controlador é capaz de processar as informações e construir a topologia global da infraes-

trutura (PAKZAD et al., 2014). As capacidades e estado dos enlaces, assim como informações mais detalhadas das portas dos comutadores, são descritas nas mensagens assíncronas de tipo *OFPT_PORT_STATUS* e *OFPMPP_PORT_DESC*.

Dentre as plataformas de controle, Open Network Operating Sytem (ONOS) (BERDE et al., 2014) é o primeiro controlador SDN de código aberto direcionado especialmente para os fornecedores de serviços. O desenvolvimento deste controlador visa garantir altos níveis de disponibilidade, desempenho e escalabilidade. ONOS utiliza várias tecnologias baseadas na plataforma Java (e.g., Maven, Karaf , JMS, Netty) e implementa diferentes protocolos de gerenciamento/controle em forma de plugins que viabilizam a programação de dispositivos da camada de dados. Uma destas implementações é o protocolo OpenFlow, que permite estabelecer um canal de comunicação com os comutadores que compõem a topologia de rede, por onde são transmitidos comandos de controle e gerenciamento de forma segura. A arquitetura deste controlador pode ser observada na figura 6.

Figura 6: Controlador SDN ONOS

3.2 Qualidade de Serviço

A qualidade de serviço nas redes de computadores pode ser definida de várias formas e inclui um conjunto diverso de requisitos como são o desempenho, a disponibilidade, a confiabilidade e a segurança. As métricas de desempenho usualmente usadas em QoS são a largura de banda, o atraso, a variação de atraso e a porcentagem de perda de pacotes. Baseados nestas métricas as garantias de desempenho da rede pode ser expressa utilizando seguinte definição (FIROIU; Le Boudec; TOWSLEY, 2002):

- *determinística*, e.g., a conexão de rede estabelecida tem garantias de reserva de recursos (e.g., 10Mbps) em todo momento;
- *probabilística*, e.g., é garantido que o atraso da rede não pode ser maior que 100 ms para o 95% dos pacotes transmitidos;
- *média ao longo do tempo*, e.g., a taxa de perda de pacote mensurada por mês é menor que 10^{-5} .

Um aspecto chave dos mecanismos de controle de QoS é a escala de tempo onde eles operam. Novamente utilizando as definições propostas por Firoiu (FIROIU; Le Boudec; TOWSLEY, 2002), podem-se dividir estas escalas de tempo em vários níveis. A escala de tempo no nível de *pacote* de rede, na ordem dos micros segundos, é a mais rápida pois o pacote é a menor unidade sobre a qual a rede pode exercer controle. Mecanismos de controle de QoS que operam nesta escala são utilizados nos dispositivos que processam o tráfego, os quais podem classificar, marcar e policiar. Também dentro desta escala podemos encontrar as operações de escalonamento de pacotes ou gerenciamento de filas. Na ordem dos milissegundos estão os mecanismos de controle de congestionamento e controle de fluxo encontrados em protocolos como o *Transmission Control Protocol* (TCP). A escala de tempo de *sessão* é a seguinte na ordem dos segundos e minutos. Usualmente é nesta escala de tempo onde operam os mecanismos

de controle de admissão e de roteamento. No próximo nível podem ser identificados os mecanismos que levam desde minutos ou horas, até meses. Dentro desta escala estão as operações de planejamento de capacidade da rede, o provisionamento de recursos e a engenharia de tráfego.

Dois dos modelos amplamente conhecidos que visam garantir QoS fim-a-fim são os Serviços Integrados (IntServ) (BRADEN; CLARK; SHENKER, 1994) e Serviços Diferenciados (DifServ) (BLAKE; MICROSYSTEMS; WANG, 1998). Ambos especificados pela Internet Engineering Task Force (IETF). A arquitetura dos **Serviços Integrados** é caracterizada pela reserva de recursos. Antes que a transmissão seja efetuada, uma rota é estabelecida e são reservados recursos ao longo do percurso, através do protocolo *Resource Reservation Protocol* RSVP (BRADEN; CLARK; SHENKER, 1994). Essa reserva de recursos exige que mudanças sejam realizadas tanto no núcleo quanto na borda da rede. Os **Serviços Integrados** propõem duas classes de serviço além do melhor esforço:

- Serviço Garantido (*Guaranteed Service*) (SHENKER; PARTRIDGE; GUERIN, 1997) para aplicações que requerem limites fixos de atraso;
- Serviço de carga controlada (*Controlled-load Service*) (WROCLAWSKI, 1997) para aplicações altamente sensíveis a redes congestionadas.

Vários problemas de escalabilidade, complexidade e custo de implementação do IntServ são conhecidos e detalhados em (NI, 1999).

Nos **Serviços Diferenciados**, os pacotes são marcados diferentemente visando criar várias classes de fluxos. Os pacotes em classes diferentes recebem serviços diferenciados. Nesta arquitetura a qualidade de serviço é garantida por meio de filas com prioridade (NI, 1999).

Ao contrário dos Serviços Integrados, os Serviços Diferenciados só precisam de

mudanças nas bordas da rede. Os roteadores de borda devem classificar os pacotes e os roteadores do núcleo da rede encaminham estes pacotes segundo as prioridades.

3.2.1 OpenFlow e QoS

Devido á importância da qualidade de serviço o protocolo *OpenFlow* na versão 1.0 (ONF, 2009), inclui algumas funcionalidades básicas para QoS. Nesta versão os pacotes podem ser encaminhados para as filas nas portas de saída dos comutadores através da ação *enqueue*, transformada em *set-queue* na versão 1.2 (ONF, 2011).

OpenFlow na versão 1.3 (ONF, 2012a) introduziu os medidores (*meters*). Estes medidores complementam as operações já existentes sobre as filas em versões anteriores, permitindo a monitoração da taxa dos fluxos antes do encaminhamento para as portas de saída. Utilizando esta nova funcionalidade, um ou vários fluxos podem ser associados a um medidor através da instrução *go-to-meter* e o medidor pode depois executar operações (e.g., descartar, remarcar o campo *differentiated services code point* (DSCP) do pacote IP) baseadas na taxa com que recebe os pacotes.

Até a data de elaboração deste trabalho os controladores da SDN podem requisitar as estatísticas e configurações das filas utilizando *Openflow*, mas não podem solicitar a criação de novas filas ou a modificação do comportamento de uma fila já existente. Devido a isto a criação e configuração de filas deve ser realizado fora do domínio *OpenFlow* utilizando por exemplo o protocolo de gerenciamento *Open vSwitch Database Management Protocol* (PFAFF; DAVIE, 2013), como pode ser observado na figura 7. Ao contrario das filas os medidores podem ser instalados, atualizados e removidos dinamicamente através de *OpenFlow*.

Figura 7: Interfaces Open vSwitch, (PFAFF; DAVIE, 2013)

3.3 Engenharia de tráfego

A Engenharia de Tráfego (TE) é o processo de organizar o tráfego de fluxos através da rede visando evitar o congestionamento que poderia causar a utilização desigual da rede. Conforme a RFC 2702, a “Engenharia de Tráfego engloba a aplicação de tecnologia e princípios científicos na medição, a caracterização, a modelagem e o controle do tráfego da Internet”

O Roteamento Baseado em Restrições(Constraint-based routing)(CRAWLEY et al., 1998) é uma importante ferramenta para implementar de forma automática a Engenharia de Tráfego. Evitar o congestionamento e a diminuição progressiva do desempenho em caso de congestionamento, são operações que se complementam. Dentro do contexto de TE o Roteamento Baseado em Restrições visa encontrar rotas que estão restritas a várias métricas de QoS como por exemplo, atraso e largura de banda. Os objetivos deste tipo de roteamento são:

- Selecionar rotas que possam atender diversos requisitos de QoS;

- Incrementar a utilização da rede.

Dois conceitos importantes da engenharia de tráfego são o balanceamento de carga e a resiliência. Pode-se dizer que o balanceamento de carga é a solução aos problemas de congestionamento enquanto a resiliência é a solução as falhas que podem ocorrer na infraestrutura de rede.

De forma geral, o balanceamento de carga com múltiplas rotas/caminhos (*multipath load balancing*) refere-se á distribuição mais equilibrada do tráfego na rede adicionando rotas na tabela de encaminhamento dos roteadores e utilizando algoritmos de balanceamento de carga capazes de identificar os fluxos e distribuir eles utilizando diferentes rotas. O conceito de balanceamento de carga com múltiplas rotas pode ser utilizado em diferentes camadas da pilha IP. Tecnologias como *Equal Cost Multi Path* (ECMP) e *Multiprotocol Label Switching Traffic Engineering* (MPLS-TE) utilizam distribuição de tráfego.

Os protocolos de roteamento que usam o estado dos enlaces como OSPF e IS-IS são baseados no algoritmo de caminho mais curto primeiro (SPF). Este algoritmo calcula somente um caminho entre o nó origem e um nó destino. ECMP é uma técnica que possibilita a utilização de múltiplos caminhos de igual custo no roteamento IP, permitindo distribuir o tráfego de forma mais equilibrada e ao mesmo tempo prover resiliência. Em ECMP, caminhos de igual custo são instalados na tabela de balanceamento de carga, na camada de encaminhamento dos roteadores. Quando uma falha em um enlace é detectada, o tráfego pode ser distribuído de forma rápida utilizando os caminhos restantes.

3.4 Network Calculus

Network Calculus (NetCalc) é uma teoria algébrica que fornece um conjunto de resultados visando descrever problemas de fluxos encontrados em redes (CRUZ, 1991b;

CRUZ, 1991a). NetCalc viabiliza o entendimento de propriedades fundamentais dos serviços integrados, agendamento, alocação de recursos e dimensionamento de atraso, onde, com o fim de fornecer garantias para os fluxos de dados, algumas caracterizações dos tráfegos e serviços da rede são necessários.

A teoria clássica de filas se preocupa geralmente com os valores médios de métricas de desempenho de um sistema em equilíbrio e o objetivo é a precisão. A obtenção dos resultados utilizando esta teoria, na maioria dos casos, está restrito a sistemas Markovianos (tráfego), os quais não são necessariamente adequados as redes TCP/IP.

Ao contrário da teoria de filas, *Network Calculus* descreve o comportamento de sistemas por meio de piores casos (*upper bounds*) e não por meio das médias (em equilíbrio), então esta teoria não trata somente de chegadas e partidas propriamente dito, mas sim de processos com limites restritivos chamados de curvas de chegadas e curvas de serviços especificadas matematicamente a seguir e mostradas na Figura 8.

Figura 8: Network Calculus, representação gráfica

O subconjunto de definições da teoria do NetCalc utilizadas neste trabalho são as

seguintes:

- **Notação**

$R(t)$: função acumulativa de entrada no sistema, estritamente crescente, que determina a quantidade de unidades (bits) em um tempo t ;

$R'(t)$: função acumulativa de saída do sistema, estritamente crescente, que determina a quantidade de unidades (bits) atendidos em um tempo t ;

- **Curva de chegada:** Dada uma função crescente α definida para $t \geq 0$ podemos dizer que um fluxo R é restrito por α se e somente se para todo $t \leq s$:

$$R(t) - R(s) \leq \alpha(t - s), \forall s, t \text{ onde } 0 \leq s \leq t \quad (3.1)$$

Pode-se dizer que R tem α como uma representação abstrata.

- **Curva de serviço:** Considere um sistema S e um fluxo que passa através de S com funções de entrada e saída R e R' . Pode-se dizer que S oferece uma curva de serviço β para o fluxo se e somente se β é estritamente crescente, $\beta(0) = 0$ e:

$$R'(s) - R'(t) \geq \beta(s - t), \forall s, t \text{ onde } 0 \leq s \leq t \quad (3.2)$$

ou

$$R'(t) \geq (R \otimes \beta)(t) \quad (3.3)$$

- **Curva de serviço remanescente:** Considere um servidor que oferece uma curva de serviço β_s . Se dois fluxos, F_0 e F_1 , atravessam o servidor com curvas de chegada α^{F_0} e α^{F_1} respectivamente, pode-se dizer que a curva de serviço remanescente oferecida para F_0 em s é dada por:

$$\beta^{l.o.F_1} = \beta_s \oslash \alpha^{F_0} \quad (3.4)$$

- **Curva de chegada remanescente:** Considere um fluxo delimitado por uma curva de chegada α , que atravessa um sistema que oferece uma curva de serviço β , o fluxo de saída é delimitado pela curva de chegada através da operação de deconvolução dada por:

$$\alpha^* = \alpha \oslash \beta \quad (3.5)$$

- **Buffer:** Para um sistema sem perdas o *buffer* no instante t é $R(t) - R'(t)$.
- **Limiar do buffer:** Considere um fluxo delimitado por uma curva de chegada α , que atravessa um sistema que oferece uma curva de serviço β . O *buffer* $R(t) - R'(t)$ para todo t satisfaz:

$$R(t) - R'(t) \leq \sup\{\alpha(s) - \beta(s)\}, \forall s, t \text{ onde } 0 \leq s \leq t \quad (3.6)$$

- **Atraso:** Menor intervalo de tempo $A(t)$ tal que todas as tarefas iniciadas no instante t sejam completadas para o instante $t + A(t)$ e pode ser especificada da seguinte forma:

$$A(t) = \min\{T : T \geq 0 \wedge R(t) \leq R'(t + T)\} \quad (3.7)$$

O atraso no instante t é o atraso que seria experimentado por um *bit* que chega no tempo t se todos os *bits* recebidos antes dele tiverem sido atendidos. Para calcular o limite do atraso o conceito de distância horizontal deve ser utilizado.

- **Distância horizontal:** Para duas funções α e β estritamente crescentes o desvio horizontal $h(\alpha, \beta)$ é dado por:

$$h(\alpha, \beta) = \max_{s \geq 0} \{\min\{T : T \geq 0 \wedge \alpha(s) \leq \beta(s + T)\}\} \quad (3.8)$$

A distância horizontal é o valor utilizado para determinar o limite do **atraso** que

pode ser descrito como:

- **Limite de atraso:** Para um fluxo restrito por uma curva de chegada α com instantes diferentes e consecutivos de início, sendo servido por uma política de *first-in-first-out* e um sistema S que garante uma curva de serviço β , o atraso para todo t satisfaz:

$$A(t) \leq h(\alpha, \beta), \forall t \geq 0 \quad (3.9)$$

- **Concatenação:** Considere um fluxo que passa através dos sistemas S_1 e S_2 em sequência. Assuma que S_i oferece uma curva de serviço $\beta_i, i = 1, 2$ ao fluxo. Então a concatenação dos dois sistemas oferece uma curva de serviço $\beta_1 \otimes \beta_2$ ao fluxo.

3.4.1 Representação e análise do modelo da rede

Geralmente as redes são modeladas como grafos, nos quais os nós representam os dispositivos de rede como os comutadores ou roteadores. Estes dispositivos podem possuir várias entradas e saídas para se conectar com outros dispositivos, como pode ser observado na Figura 9. Este modelo é diferente do modelo de servidores, representado na Figura 9(b), utilizado em NetCalc para a análise das filas.

Figura 9: Representação tradicional de redes e o modelo utilizado por NetCalc

A análise da rede com NetCalc permite calcular o limite do atraso fim-a-fim para

um fluxo de interesse (foi). Existem dois métodos principais documentados na literatura para determinar o atraso que experimenta um fluxo quando ele passa através de uma sequência de servidores.

A análise total de fluxos (TFA) (CRUZ, 1991b), aplica de forma direta os resultados das expressões (3.5)(3.6)(3.8). Dada a curva de chegada representada pela agregação dos fluxos presentes em um servidor e a curva de serviço do mesmo, a TFA permite calcular os limites dos valores de atraso experimentado pelo fluxo (agregado) ao passar pelo servidor analisado, assim como o *buffer* total demandado no servidor.

TFA é considerada uma análise local de cada servidor, pois não leva em consideração os fluxos individualmente e sim uma agregação de todo o tráfego presente nele. Quando a TFA é utilizada para analisar uma sequência de servidores, o atraso experimentado pelo fluxo de interesse é representado pela soma de todos os atrasos individuais em cada servidor no percurso.

A análise separada de fluxos (SFA), melhora os valores de atraso da TFA separando o fluxo de interesse do resto dos fluxos presentes no servidor. Primeiro aplica-se a expressão (3.4) e a curva de serviço β_s , em cada um dos servidores passa a ser representada pela curva de serviço remanescente (*left-over service curve*) $\beta^{l.o.foi}$. Depois, através da concatenação destas curvas de serviço remanescentes, todo o sistema (sequência de servidores) é representado por uma curva de serviço remanescente. Calculando o atraso com a curva de serviço remanescente fim-a-fim, a rajada do fluxo de interesse é considerada só uma vez. Este efeito na literatura é chamado de *Pay Burst Only Once* (PBOO).

Neste trabalho a topologia da rede é tratada como um grafo conectado e ponderado onde cada um dos nós (servidores) oferece uma curva de serviço β . O peso das arestas é dado pelo atraso que experimenta um fluxo caracterizado por uma curva de chegada ao longo das possíveis rotas desde a fonte até o destino, como pode ser observado na Figura 10.

Formalizando matematicamente a rede observada na Figura 10, para um fluxo que passa através dos nós 1,2,3 temos que:

Seja $G = (V, E)$ uma rede onde V é o subconjunto de nós que compõem a rede, representados pelas suas respectivas curvas de serviço (β),

- $V: \{\beta_1, \beta_2, \beta_3\}$

E é o subconjunto de segmentos de caminhos, formados por enlaces pelos quais o fluxo irá passar da origem X ao destino Y , as quais tem associado um atraso (h), considerando o fluxo de entrada (α) e seu atendimento pelo nó através do serviço (β).

- $E: \{h_{x \rightarrow y}(\alpha, \beta_1 \otimes \beta_2 \otimes \beta_3)\}$

Figura 10: Modelo de rede através de NetCalc

Sendo assim, ao escolher um caminho P qualquer, desde uma fonte transmissora até o destino que requisita a transmissão, o atraso máximo A_{max} , experimentado pelo fluxo α através de P será:

$$A_{max} \leq h(\alpha, (\beta_i \otimes \beta_{i+1} \dots, \otimes \beta_m)) \quad (3.10)$$

Observe que no grafo não é considerado nem o enlace inicial de X para o primeiro nó, nem o atraso associado. A explicação desta consideração é que o controlador SDN somente, por padrão, possui informação sobre os comutadores SDN e não dos equipamentos (e.g., *hosts*) conectados em cada um deles.

A forma adotada nesta dissertação para a formalização matemática dos fluxos responde ao funcionamento do algoritmo *leaked-bucket*. Logo a expressão que caracteriza

um fluxo de rede está dada pela curva de chegada $\gamma_{b,r}$, onde b representa a rajada máxima do fluxo em qualquer instante de tempo e r , a taxa de transmissão ao longo do tempo como pode ser observado na Figura 8.

Um mecanismo simples para garantir que os fluxos de rede sejam restritos pela curva de chegada $\gamma_{b,r}$, é a instalação de controladores *leaked-bucket* nos dispositivos emissores (e.g., servidores de vídeo). Um controlador *leaked-bucket* é um dispositivo que analisa os dados em um fluxo $R(t)$ da forma seguinte. Existe um depósito de fluído (*bucket*) com capacidade b como pode-se observar na Figura 11. O depósito está originalmente vazio e possui um furo que vaza a uma taxa de r unidades de fluído por segundo quando não está vazio. É colocado no depósito tanto fluído quanto dado do fluxo $R(t)$ se deseja transmitir. A quantidade de dados que cause uma sobrecarga do depósito será declarada não conforme, caso contrário os dados serão declarados conformes.

Figura 11: Controlador *Leaked-Bucket*

A formalização matemática das curvas de serviço, que representará os nós da rede (servidores) estará definida pela expressão $\beta_{R,T}$, como pode ser observada na Figura 8. Esta função chamada de Taxa-Latência é uma forma simples de representar um

servidor que possui uma taxa constante de atendimento R e não demora mais do que T unidades de tempo em processar uma unidade de dado.

4 PROPOSTA

O *framework* proposto nesta dissertação, observado na Figura 12, está baseado na teoria de NetCalc o qual permitirá caracterizar os comutadores da rede em termos de *curvas de serviço* e os fluxos em termos de *curvas de chegada*. Com isto é possível construir uma matriz de tráfego que represente o estado atual da infraestrutura, considerando a disponibilidade de recursos e os custos de transmissão.

Figura 12: *Framework* para a engenharia de tráfego em redes definidas por software utilizando *Network Calculus*

Para implementar o *framework* será utilizado um controlador SDN que fornecerá capacidade de armazenamento assim como a disponibilidade e a consistência das informações persistidas na matriz de tráfego. Através de SDN e especificamente dos protocolos *OpenFlow* e *Open vSwitch Database Management Protocol*, vistos na seção 3.1 e subseção 3.2.1, será possível manter um controle ativo dos dispositivos que fazem parte da topologia (i.e., comutadores, *hosts*) e das filas de atendimento.

No *framework* é reutilizado o conceito de serviço garantido através de NetCalc, o qual permitirá determinar os fluxos que serão aceitos na rede e fornecer rotas com o menor atraso possível. Além disso, este trabalho propõe alocação de recursos não por fluxos individuais, e sim por fluxos agregados de uma mesma classificação. Nesta pesquisa a identificação, classificação e a decisão de enfileiramento de fluxos é feita no controlador da rede.

Cada fluxo é cadastrado inicialmente no controlador SDN pela fonte transmissora (e.g., servidor de vídeos) utilizando a interface de gerenciamento e deve conter as informações necessárias para a identificação do fluxo bem como os parâmetros de QoS que caracterizam o perfil de tráfego a ser tratado.

Como visto na Seção 3.4.1 as funções que representarão os fluxos de dados e os nós da rede respectivamente serão:

$$\alpha = \gamma_{r,b} = b + r(t) \quad \text{para } t > 0 \quad (4.1)$$

$$\beta = \beta_{R,T} = R[t - T]^+ = \begin{cases} R(t - T) & \text{se } t > T \\ 0 & \text{senão} \end{cases} \quad (4.2)$$

Neste capítulo são detalhados os diversos componentes que fazem parte do *framework*. Cada um deles, segundo a nossa visão, responde as necessidades de controle, caracterização, classificação e distribuição eficiente do tráfego que as redes atuais demandam.

4.1 Componente para atendimento diferenciado

O grau de sensibilidade dos fluxos de pacotes às métricas de QoS como o atraso, a variação de atraso, a largura de banda e a perda de pacotes vai depender do tipo de fluxo. Para vídeos interativos (videoconferência) (SZIGETI; HATTINGH, 2004) recomenda-se uma perda de pacotes menor que 1%, atraso menor que 150 ms e uma

variação de atraso não maior que 30 ms. Por sua vez, para *streaming* de vídeo, o mesmo autor sugere uma perda menor que 5% e um atraso não maior que 5 segundos. Para este caso não há valores significativos para a variação de atraso.

Visando fornecer atendimento diferenciado para distintas classes de fluxos, os nós da rede (servidores) deverão oferecer mais de uma curva de serviço. Cada classe tem associada a sua respectiva fila de atendimento as quais, neste trabalho, são do tipo *first-in-first-out* (FIFO), como é observado na Figura 13.

Figura 13: Atendimento diferenciado nos canais de saída dos nós da rede

Levando em consideração a representação da rede na Subseção 3.4.1 uma nova rede pode ser definida como:

Seja $G = (V, E)$ uma rede onde V é o subconjunto de nós que compõem a rede, representados pelas suas respectivas curvas de serviço específicas para cada classe de fluxo $(\beta_{c1, c2, c3, \dots, cx})$. E é o subconjunto de segmentos de caminhos, formados por enlaces pelos quais o fluxo irá passar da origem X ao destino Y , os quais tem associado um atraso (h) , considerando o fluxo de entrada (α) e seu atendimento **diferenciado** pelo nó através do serviço específico requisitado $(\beta_{c1|c2|c3|\dots|cx})$, representado na Figura 14.

Novamente escolhendo um caminho P , desde uma fonte transmissora até o destino que requisita a transmissão, o atraso máximo $A_{\max}$, experimentado pelo fluxo α com

Figura 14: Modelo de rede com atendimento diferenciado

atendimento diferenciado C_m através de P será:

$$A_{\max} \leq h(\alpha_{cm}, (\beta_{icm} \otimes \beta_{(i+1)cm} \dots \otimes \beta_{jcm})), \text{ onde } i, j \in \mathbb{N}, j = |P| \quad (4.3)$$

Quando aplicado este método a rede deve estar particionada conforme as classes de fluxos criadas. Como pode ser observado na Figura 15, é possível a criação de várias topologias lógicas dentro de uma mesma infraestrutura física.

4.2 Componente para admissão de fluxos

O componente para admissão de fluxos é responsável por determinar quando um fluxo, levando em consideração as características dele e a identificação do mesmo, deve ser admitido na rede ou não. Para o cálculo do atraso que experimenta um fluxo na rede, NetCal, define que para sistemas sem perdas a taxa de transmissão de um fluxo não deve exceder a taxa de atendimento dos servidores pelos quais este passa. Isto é,

Figura 15: Particionamento da rede para o atendimento de diversas classes de fluxos

interpretando a Figura 8, com $\alpha = rt + b$ e $\beta = R(t - T)$:

$$d = h(\alpha, \beta) = \frac{b}{R} + T \quad \text{para } r \leq R \quad (4.4)$$

Uma vez garantida a disponibilidade de recursos na rede, o componente para admissão deverá calcular o atraso máximo do fluxo e verificar que não seja maior do que o valor especificado no cadastro do fluxo ou um valor predefinido pelo administrador da rede. A Figura 16 mostra a sequência de atividades que realiza o componente de admissão.

Recalcular os atrasos experimentados para todos os fluxos da rede, cada vez que um novo fluxo solicita a entrada, seria inviável. O *framework* proposto nesta dissertação realiza esta tarefa após um novo fluxo ser aceito. Ou seja, o componente verifica a influência do fluxo aceito sobre os recursos já disponibilizados para os fluxos presentes na rede. Uma vez que os atrasos sejam atualizados, como pode ser observado no Algoritmo 1 e 2, o componente de admissão disponibiliza estes valores para outros

Figura 16: Sequência de atividades realizadas pelo componente de admissão

componentes (e.g., componente para o balanceamento de carga).

Algoritmo 1: Influência do fluxo solicitante sobre os recursos disponíveis

Data: *rota_do_fluxo_solicitante, fluxo_solicitante*

begin

servidor_borda := *rota_do_fluxo_solicitante*[0] ;

for *fluxo* ≠ *fluxo_solicitante* ∈ *servidor_borda* **do**

fluxo.atualizarAtraso() ;

// Fluxos afetados

end

end

No processo de atualização dos atrasos dos fluxos, detalhado no Algoritmo 2, é utilizada uma análise SFA, levando em consideração as curvas de serviço remanescentes nas rotas dos fluxos afetados. Logo basta aplicar a expressão (3.10) vista no Capítulo

3, Seção 3.4;

Algoritmo 2: Atualização dos atrasos dos fluxos afetados pela entrada de um novo fluxo

```

begin
    betas_da_rota = [] ;
    for servidor  $\in$  fluxo.rota do
        | betas_da_rota.adiciona(servidor.beta_lo) ;
    end
    /* curva de serviço remanescente resultante na rota */
    beta_lo_total =: convolução(betas_da_rota) ;
    /* atualização do atraso experimentado pelo fluxo */
    fluxo.atraso = h(fluxo.alpha, beta_lo_total);
end

```

4.3 Componente para o balanceamento de carga

Múltiplos algoritmos para encontrar o caminho mais curto dado um valor de custo entre os vértices de um grafo foram propostos ao longo do tempo. Um dos mais utilizados nas redes (e.g., IS-IS (SMIT; LI, 2004), OSPF (J.MOY, 1998)) é o algoritmo de Dijkstra (DIJKSTRA, 1959).

Qualquer algoritmo, que para encontrar o caminho mais curto leve em consideração o valor associado a cada aresta, pode utilizar o modelo analítico proposto aqui. Neste trabalho utilizaremos Dijkstra, sendo este relativamente simples e eficiente (i.e., $O(n \log(n))$, onde n é a quantidade de vértices do grafo).

Um aspecto a ser considerado na computação do melhor caminho fim-a-fim é que, segundo foi demonstrado na teoria de *Network Calculus* e explicado na subseção 3.4.1, a soma dos atrasos experimentados em cada nó da rede (i.e., $h(\alpha, \beta_1) + h(\alpha_1^*, \beta_2) + h(\alpha_2^*, \beta_3) \dots + h(\alpha_{n-1}^*, \beta_n)$) (TFA), é maior do que a convolução das curvas de serviço oferecidas pelos nós no caminho (i.e., $h(\alpha, \beta_1 \otimes \beta_2 \otimes \beta_3 \dots, \otimes \beta_n)$) (SFA). Portanto, devemos adaptar o algoritmo de Dijkstra visando ser coerente com a equação 3.10.

O algoritmo de Dijkstra convencional, utiliza um grafo com os pesos das arestas imutáveis. Cada iteração até alcançar o nó final, soma os pesos das arestas do caminho

com menor custo. Quando aplicamos NetCalc, o algoritmo mantém a sua lógica original, mas, as operações e dinâmica com as quais são determinados os pesos das arestas tem que ser modificados, como pode ser observado na Figura 17. A análise da rede que adota o componente explicado nesta secção é a SFA, a qual foi detalhada na Secção 3.4.1 do Capítulo 3.

Figura 17: Características do algoritmo de Dijkstra utilizando NetCalc

O algoritmo 3 mostra como a operação que visa determinar o custo para alcançar um nó, é a convolução e não a soma tradicional utilizada em Dijkstra. A rede sobre a qual o algoritmo itera representa a partição lógica correspondente ao tipo de atendimento necessário para o fluxo de interesse. Isto é, conforme representado na Figura 15, os fluxos de classe X trafegam pela topologia de classe X.

O balanceamento de carga é garantido neste componente devido á atualização das curvas de serviço remanescentes nos servidores. Quando um fluxo é aceito na rede, o componente deve atualizar os recursos disponíveis na infraestrutura como é mostrado na Figura 18. Com isto a matriz de tráfego vai sendo atualizada conforme os fluxos são aceitos na rede e a decisão de encaminhamento é baseada em informações que refletem o estado atual da rede de forma consistente.

Algoritmo 3: Dijkstra utilizando NetCalc

Data: *rede, servidor_fonte, servidor_destino, fluxo_de_interesse*
Result: *melhor_rota, atraso_máximo*

```

begin
  atraso[servidor_fonte] := 0 ;           // Atraso para chegar em [servidor]
  atraso_máximo := 0;
  /* Inicialização */
  for servidor ∈ rede do
    if servidor ≠ fonte then
      atraso[servidor] := INFINITO;
      anterior[servidor] := INDEFINIDO;
    end
    adiciona servidor em nao_visitados
  end
  while nao_visitados ≠ ∅ do
    /* c é a fonte na 1ra iteração */
    c := servidor ∈ nao_visitados com menor atraso[c];
    remover c de nao_visitados;
    for vizinho v de c do
      /* Atraso para chegar em v com uma análise SFA */
      a := h(fluxo_de_interesse.alpha, c.betal.o.foi);
      if a < atraso[v] then
        /* v.betal.o.foi, mantem a curva de serviço remanescente dos
           servidores percorridos */
        v.betal.o.foi = c.betal.o.foi ⊗ v.betal.o ;           // Concatenação
        atraso[v] := a ;           // Atualização do atraso
        anterior[v] := c ;           // Construção da rota
      end
      if v = destino then
        /* a, possui o valor do atraso com ao qual o fluxo de
           interesse chega ao destino */
        atraso_máximo := a ;
        /* retorna a rota fonte -> destino */
        return anterior[].invertido(), atraso_máximo ;
      end
    end
  end
end
end
end

```

- (a) Todas as portas experimentam o mesmo atraso. A rota é escolhida de forma aleatória
- (b) É considerado o fluxo já existente na computação do atraso

Figura 18: Balanceamento de carga

4.3.1 Exemplo ilustrativo

Visando esclarecer a aplicação prática dos conceitos até agora vistos, foi criado um cenário hipotético o qual pode ser observado na Figura 19. A configuração dos enlaces assim como os serviços oferecidos na infraestrutura são detalhados na Tabela 1. Os controladores de tráfego são de tipo *Leaked-Bucket*, mecanismo previamente explicado na Subseção 3.4.1 e os fluxos de vídeo h2->h4 e h3->h4 possuem uma taxa de transmissão de 5 Mbps e 3Mbps respectivamente. Por sua vez o fluxo VoIP é de 60kbps. O objetivo deste exemplo é descrever o funcionamento do Algoritmo 3 para um novo fluxo de vídeo com origem h1 e destino h4 (h1->h4) com taxa de transmissão de 5 Mbps.

Primeiramente a rede é particionada como foi explicado na Seção 4.1, para o atendimento das duas classes de fluxos previamente especificadas (i.e., Vídeo, VoIP). Este particionamento pode ser observado na Figura 20. Como o fluxo de interesse é de vídeo o Algoritmo 3 itera sobre a partição 20(a) correspondente a esta classe de fluxo. A modelagem dos fluxos e portas dos comutadores em termos de curvas de chegada e curvas de serviço respetivamente utiliza as expressões 4.1 e 4.2 podendo ser observado nas Tabelas 2 e 3.

Uma vez particionada a rede o próximo passo é verificar se existem recursos sufi-

Figura 19: Cenário hipotético de uma rede com fluxos de vídeo e VoIP

Tabela 1: Configuração dos enlaces e os serviços oferecidos

Enlace	Capacidade	Serviços
sw1-sw2	10Mbps	Vídeo, VoIP
sw1-sw3	10Mbps	Vídeo
sw1-sw4	1Mbps	VoIP
sw2-sw4	10Mbps	Vídeo
sw2-sw3	1Mbps	VoIP
sw3-sw4	10Mbps	Vídeo, VoIP

(a) Recursos reservados para transmissões de vídeo

(b) Recursos reservados para transmissões de VoIP

Figura 20: Reserva de recursos por classe de fluxos

cientes para que o fluxo h1->h4 possa ser aceito na rede. Neste exemplo é considerado que a rede possui os recursos necessários para o fluxo de interesse. Por último é aplicado o Algoritmo 3 visando encontrar o caminho com menor atraso desde h1 até h4. A Figura 21 mostra o funcionamento por iterações do algoritmo o qual é detalhado a continuação:

1. Levando em consideração que a porta p-sw1-sw2 não possui outros fluxos, a curva de chegada remanescente para o fluxo de interesse está dada pela expressão

$$\beta_{p-sw1-sw2}^{l.o.foi} = \beta_{9,0}. \text{ O atraso experimentado pelo fluxo de vídeo h1->h4 pode ser representado como: } d = h(\alpha^{foi}, \beta_{p-sw1-sw2}^{l.o.foi}) = \frac{b}{R} + T \text{ para } r \leq R$$

$$\frac{5}{9} + 0 = \frac{5}{9}.$$

De forma análoga o atraso experimentado pelo fluxo na porta p-sw1-sw3 pode

$$\text{ser representado como: } d = h(\alpha^{foi}, \beta_{p-sw1-sw3}^{l.o.foi}) = \frac{b}{R} + T \text{ para } r \leq R$$

$$\frac{5}{10} + 0 = \frac{1}{2}$$

2. Nesta iteração deve ser calculada a curva de serviço remanescente

$$\beta_{p-sw3-sw4}^{l.o} = \beta_{p-sw3-sw4} \oslash \alpha_{h3->h4} \text{ devido a influência do fluxo de video existente.}$$

Para isso:

$$R^{l.o} = R - r$$

$$T^{l.o} = \frac{b}{R} + T$$

$$\beta_{p-sw3-sw4}^{l.o} = \beta_{6,5/9}. \text{ O atraso experimentado pelo fluxo de interesse pode ser expressado como: } d = h(\alpha^{foi}, \beta_{p-sw1-sw3}^{l.o.foi} \otimes \beta_{p-sw3-sw4}^{l.o.foi})$$

$$= \frac{b}{\min(R_{p-sw1-sw3}, R_{p-sw3-sw4})} + T_{p-sw1-sw3} + T_{p-sw3-sw4}$$

$$= \frac{5}{6} + 5/9 = 25/18.$$

3. De forma análoga nesta iteração deve ser calculada a curva de serviço remanescente $\beta_{p-sw2-sw4}^{l.o} = \beta_{p-sw2-sw4} \oslash \alpha_{h2->h4}$ devido a influência do fluxo de video existente. Para isso:

$$R^{l.o} = R - r$$

$$T^{l.o} = \frac{b}{R} + T$$

$\beta_{p-sw2-sw4}^{l.o} = \beta_{5,1/2}$. Logo o atraso experimentado pelo fluxo de interesse pode ser expressado como: $d = h(\alpha^{foi}, \beta_{p-sw1-sw2}^{l.o.foi} \otimes \beta_{p-sw2-sw4}^{l.o.foi})$

$$= \frac{b}{\min(R_{p-sw1-sw2}, R_{p-sw2-sw4})} + T_{p-sw1-sw2} + T_{p-sw2-sw4}$$

$$= \frac{5}{5} + 1/2 = 3/2. \text{ Como } 3/2 > 25/18 \text{ o valor para chegar em sw4 é mantido com predecessor sw3.}$$

4. Finalmente a rota com menor atraso para o fluxo de vídeo h1->h4 é **sw1->sw3->sw4**.

Tabela 2: Modelagem da rede para fluxos de vídeo aplicando NetCalc

Porta	Curva de serviço
p-sw1-sw2	$\beta_{9,0}$
p-sw1-sw3	$\beta_{10,0}$
p-sw2-sw4	$\beta_{10,0}$
p-sw3-sw4	$\beta_{9,0}$

Tabela 3: Modelagem dos fluxos de vídeo aplicando NetCalc

Fluxo	Curva de chegada
h1->h4	$\gamma_{5,5}$
h2->h4	$\gamma_{5,5}$
h3->h4	$\gamma_{3,5}$

Iteração	Inicialização	1	2	3	4	Menor atraso	Predecessor
nó ativo		sw1	sw3	sw2	sw4		
sw1	0					0	-
sw2	∞	5/9				5/9	sw1
sw3	∞	1/2				1/2	sw1
sw4	∞	∞	25/18	25/18		25/18	sw3

Figura 21: Funcionamento por iterações do Algoritmo 3

Como foi visto no exemplo ilustrativo o *framework* proposto nesta dissertação é capaz de tratar com diferentes classes de tráfego e diferentes configurações nas capacidades dos enlaces da rede. Através de aplicação de NetCalc o Algoritmo 3 calcula o

caminho com menor atraso, levando em consideração os fluxos existentes na infraestrutura.

4.4 Componente para a resiliência

Dos métodos encontrados na literatura e que foram explicados no Capítulo 2, o componente de resiliência explora a *restauração*. Isto se justifica pelo grande controle analítico que caracteriza o *framework* proposto nesta dissertação. O dinamismo com que a matriz de tráfego é construída e atualizada, dificultam a interação eficiente com as regras de recuperação instaladas na camada de dados.

O componente para a resiliência trata a *restauração* da seguinte forma:

Cada fluxo aceito na rede deve possuir várias rotas associadas como é observado na Figura 22 (métrica a ser avaliada segundo a quantidade de comutadores da rede). Para isso é aplicado o Algoritmo 3 várias vezes. A primeira rota encontrada pelo Algoritmo 3 (principal) deve ser instalada na rede imediatamente. Uma vez detectada a falha, a rota principal afetada deve ser retirada da rede e instalada a alternativa correspondente. No caso de não houver uma rota pré-computada, deve ser aplicado a Algoritmo 3 novamente.

Figura 22: Mecanismo de restauração

As rotas alternativas de cada fluxo são tão dinâmicas quanto os custos de transmissão na matriz de tráfego. Existem dois eventos de rede, mostrados nas Figuras 23(a), 23(b), que podem causar uma modificação na política de restauração, estes são:

- Novo fluxo é aceito na rede: Os recursos disponíveis nas rotas alternativas podem mudar se estes já não cumprem com as políticas do *framework* (e.i., atraso da rota é maior que o atraso permissível especificado no cadastro do fluxo)
- Ocorrência de uma falha não associada ao fluxo principal: O componente deve verificar todas as rotas afetadas na ocorrência de uma falha na rede.

Figura 23: Eventos que ativam o componente de resiliência

4.5 Modelagem e implementação

Este capítulo detalha a metodologia, os procedimentos e as tecnologias adotadas na implementação da proposta desta dissertação. As decisões de implementação em um ambiente SDN possuem alta importância devido à influência direta em três aspectos fundamentais: a disponibilidade, a confiabilidade e a performance do sistema. A presente dissertação não visa aprofundar em aspectos teóricos da construção de software

(e.g., Programação Orientada a Objetos, Padrões de desenho, Análise de algoritmos), mas apoia-se em modelos UML (BOOCH; RUMBAUGH; JACOBSON, 2005) que guiarão a implementação dos componentes que compõem a nossa solução.

4.5.1 Requisitos

A decomposição do objetivo geral em pequenos requisitos funcionais provê uma visão detalhada das responsabilidades de cada um dos componentes mencionados no Capítulo 4.

O componente para o atendimento diferenciado permitirá:

- Classificar o tráfego (vídeo, voz, data , etc.);
- Disponibilizar uma API REST para o cadastro no controlador SDN das diferentes classes de tráfego;
- Alocação de recursos na rede conforme as diferentes classes de tráfego cadastradas no controlador SDN.

O componente para admissão de fluxos permitirá:

- Caracterizar os fluxos de pacotes em perfis de clientes;
- Disponibilizar uma API REST para o cadastro no controlador SDN dos diferentes perfis;
- Processar os pacotes IP que chegam ao controlador através das mensagens PACKET-IN do protocolo OpenFlow;
- Identificar os fluxos de pacotes conforme os perfis cadastrados no controlador SDN;
- Determinar se um novo fluxo de pacotes pode ser admitido na rede;

- Determinar se um fluxo de pacotes já admitido pode ser rejeitado devido à falta de recursos, posterior a uma alteração na infraestrutura de rede.

O componente para o balanceamento de carga permitirá:

- Calcular os caminhos de menor atraso entre uma fonte e um destino;
- Atualizar os recursos de rede disponíveis quando um fluxo de pacotes é admitido na infraestrutura;
- Criar as regras de encaminhamento específicas para cada fluxo de pacotes;
- Definir, atualizar e remover rotas fim-a-fim nos comutadores da rede.

O componente para a resiliência permitirá:

- Calcular os caminhos de respaldo com menor atraso entre uma fonte e um destino;
- Detectar a ocorrência de falhas na rede;
- Identificar os fluxos afetados pela ocorrência de uma falha na rede;
- Restaurar os fluxos afetados pelas falhas de rede.

4.5.2 Estruturas

A modelagem estática das informações que interagem dentro de um contexto específico é uma fase fundamental no desenvolvimento de um projeto. Nesta dissertação foram agrupadas as informações em forma de classes. Isto permitirá adotar o paradigma orientado a objetos como pode ser visto na Figura 24.

4.5.2.1 Modelo de classes principais que sustentam o *framework* proposto nesta dissertação.

- **NCClientTrafficProfile:** Classe que representa as características dos fluxos que serão transmitidos. O módulo de admissão utiliza objetos desta classe para identificar e controlar o acesso de novos pacotes na rede;
- **NCFlow:** Representa um fluxo de pacotes IP;
- **NCIntent:** Abstração de regras de encaminhamento tratadas como um circuito virtual fim-a-fim;
- **NCQueue:** Classe que define a alocação de recursos para uma determinada classificação de fluxos;
- **NCTrafficClass:** Classifica o tráfego de rede.

Figura 24: Diagrama de classes

4.5.2.2 Componente para o atendimento diferenciado:

- **NCDiffServService:** Interface que descreve as operações relacionadas com a reserva de recursos na rede para diferentes classes de tráfego;
- **NCDiffServManager:** Classe que implementa as operações relacionadas com a reserva de recursos na rede para diferentes classes de tráfego;
- **NCDiffServMonitor:** Classe responsável por monitorar os eventos relacionados com as portas dos comutadores para a criação, atualização ou remoção das filas que representam o atendimento diferenciado para diferentes classes de tráfego.

A modelagem deste componente pode ser visto na Figura 25.

Figura 25: Diagrama de classes do componente para o atendimento diferenciado

4.5.2.3 Componente para admissão de fluxos:

- **NCA AdmissionService:** Interface que descreve as operações para determinar a admissão de um fluxo na rede;
- **NCA AdmissionManager:** Classe que implementa as operações para determinar a admissão de um fluxo na rede.

A modelagem deste componente pode ser visto na Figura 26.

Figura 26: Diagrama de classes do componente para a admissão de fluxos

4.5.2.4 Componente para o balanceamento de carga:

- **NCTrafficEngineeringService:** Interface que descreve as operações de engenharia de tráfego;
- **NCTrafficEngineeringManager:** Classe que implementa as operações de engenharia de tráfego;
- **NCTrafficEngineeringMonitor:** Classe responsável por monitorar os eventos relacionados com as rotas instaladas na infraestrutura;
- **NCLinkWeight:** Classe que determina o atraso máximo experimentado por um fluxo em um comutador.

A modelagem deste componente pode ser visto na Figura 27.

4.5.2.5 Componente para resiliência:

- **NCResiliencyService:** Interface que descreve as operações de resiliência;
- **NCResiliencyManager:** Classe que implementa as operações de resiliência;
- **NCResiliencyMonitor:** Classe responsável por monitorar os eventos relacionados com as falhas de enlaces na infraestrutura de rede.

Figura 27: Diagrama de classes do componente para o balanceamento de carga

A modelagem deste componente pode ser visto na Figura 28.

Figura 28: Diagrama de classes do componente para a resiliência

A Figura 29 mostra como interagem os componentes dentro do sistema. Esta visão arquitetural estabelece as dependências e os serviços utilizados ou fornecidos por cada um dos componentes.

Figura 29: Diagrama de componentes

5 RESULTADOS

Neste capítulo é detalhado o processo de avaliação do *framework* proposto. Na Seção 5.1 são mencionadas algumas considerações iniciais com relação as dificuldades encontradas na avaliação de um trabalho científico dentro do âmbito SDN. Depois são especificados quais aspectos da engenharia de tráfego serão avaliados e quais as métricas de desempenho que permitirão a análise do *framework* o qual, daqui em diante, será citado como DSDN-TE.

5.1 Considerações iniciais

Um dos requisitos fundamentais em um trabalho científico é a capacidade de ser reproduzível. SDN traz um novo paradigma para enfrentar os problemas de flexibilidade das redes tradicionais, e nesse contexto *OpenFlow* fornece uma base comum de desenvolvimento, a partir da qual o usuário final é capaz de implementar soluções diversas não regidas, a maioria das vezes, por padrões.

Atualmente existem dezenas de plataformas de controle, implementadas em diferentes linguagens de programação e definições arquiteturais diversas. Estas plataformas são simples programas que são executados em hardwares nos quais o desempenho pode ser comprometido. Igualmente ocorre com as ferramentas de virtualização atuais, que atuam dentro do contexto de um sistema operacional e que são sujeitas as políticas de priorização e gerenciamento de tarefas dos programas comuns. Todos estes elementos fazem com que, embora existam na literatura várias pesquisas com objetivos

convergentes aos nossos, a falta de disponibilidade do código das soluções, assim como o pobre detalhamento das características do ambiente de testes, dificultam a realização de uma análise rigorosa.

A implementação do DSDN-TE está disponível em <https://github.com/michaelph/deterministic-sdn-te.git>. O controlador SDN ONOS (BERDE et al., 2014), foi escolhido como nosso ambiente de desenvolvimento, devido á ampla aceitação dentro da comunidade SDN, assim como por ser um projeto continuamente atualizado e documentado. O ambiente de virtualização foi criado através das ferramentas Mininet (LANTZ; HELLER; MCKEOWN, 2010) e Open vSwitch (PFAFF et al., 2009), também amplamente utilizadas pela comunidade científica, de código aberto e gratuitas. Como ferramenta para a geração de tráfego foi usada iPerf, a qual permite medir a vazão do sistema, assim como as perdas experimentadas pelos fluxos. IPerf permite gerar tráfego UDP/TCP e é muito utilizada em pesquisas científicas. Esta ferramenta é de código aberto e já está inclusa nos programas padrões de sistemas GNU/Linux.

O trabalho analítico relacionado com a teoria de *Network Calculus*, utiliza a biblioteca DiscoDNC (BONDORF; SCHMITT, 2014). Os principais motivos para a escolha desta biblioteca, depois de considerar o estudo realizado em (BOYER, 2010), foram o rigor científico com a qual está sendo construída, a linguagem de programação em que é escrita (i.e., Java), que é a mesma do controlador ONOS, e o fato da biblioteca DiscoDNC possuir classes prontas que representam as definições apresentadas na Seção 3.4. A utilização de DiscoDNC, valida a uso correto da teoria de NetCalc e diminui os erros de programação que poderiam existir no tratamento desta teoria com estruturas nativas da plataforma Java.

5.2 Experimentos

O aparato experimental utilizado nesta dissertação (i.e., Software e Hardware) pode ser observado nas Tabelas 4 e 5.

Tabela 4: Características dos equipamentos utilizados no experimento

	CPU	Memória
Rede virtual e ONOS	Intel Xeon E3-1240v5	32 GB

Tabela 5: Versionamento dos softwares e protocolos utilizados no experimento

	Versão
ONOS	1.5.0 "Falcon"
Mininet	2.1
DiscoDNC	2.2.6 "Hydra"
Open vSwitch	2.5.0
OpenFlow	1.3
iperf	2.0.5
Sistema operacional	Ubuntu 14.04 LTS "Trusty Tahr"

Com o intuito de diminuir os erros humanos que poderiam influenciar os resultados dos experimentos, foi construído um software capaz de realizar de forma automatizada as tarefas necessárias para a execução da fase experimental. O software desenvolvido é capaz de:

- Requisitar a execução de uma transmissão;
- Especificar os detalhes da transmissão em termos de duração, protocolo, taxa, etc.;
- Cadastrar no controlador SDN diversos perfis de tráfegos;
- Determinar o número de iterações de determinada transmissão ou conjunto de transmissões;
- Determinar o tempo entre cada transmissão, ou conjunto de transmissões, assim como tempo entre as iterações.

A metodologia utilizada nos experimentos pode ser observada na Figura 30. Inicialmente são cadastrados os perfis de tráfego no controlador SDN. Na sequência são requisitadas e executadas as transmissões na rede e finalmente são persistidos os resultados.

Figura 30: Ambiente para a automatização de testes em SDN

As aplicações utilizadas no controlador SDN podem ser vistas na Figura 31¹ e são detalhadas a seguir:

- *Default Device Drivers*: Permite a interação com os dispositivos de rede (e.g., *hosts* e comutadores);
- *Host Location Provider*: Descobre a localização dos *hosts* da rede analisando as mensagens PACKET-IN com informações do protocolo ARP;
- *LLDP Link Provider*: Descobre e disponibiliza a distribuição dos enlaces da rede;
- *OpenFlow Meta App* e *OpenFlow Provider*: Permite ao controlador interagir com comutadores OpenFlow;

¹As figuras de 28 a 30 mostram telas capturadas do controlador ONOS

- *usp.larc.intentnc*: Implementação do *framework* proposto nesta dissertação;
- *usp.larc.restnc*: Interface REST que permite o cadastro de perfis de tráfego no controlador;
- *usp.larc.ncprofiles*: Permite a visualização dos perfis de tráfego cadastrados no controlador como pode ser observado na Figura 33;
- *usp.larc.ncqueues*: Permite a visualização dos nós da rede em termos de curvas de serviço, assim como o número de fluxos em cada nó, o tipo de fluxo e a carga em tempo real do nó (Figura 32).

Icon	Title	App ID	Version	Category	Origin
✓	Default Device Drivers	org.onosproject.drivers	1.5.0	Drivers	ON.Lab
✓	Host Location Provider	org.onosproject.hostprovider	1.5.0	Provider	ON.Lab
✓	LLDP Link Provider	org.onosproject.lldpprovider	1.5.0	Provider	ON.Lab
✓	OpenFlow Meta App	org.onosproject.openflow	1.5.0	Provider	ON.Lab
✓	OpenFlow Provider	org.onosproject.openflow-base	1.5.0	Provider	ON.Lab
✓	usp.larc.intentnc	usp.larc.intentnc	1.5.0	utility	LARC
✓	usp.larc.restnc	usp.larc.restnc	1.5.0	utility	LARC
✓	usp.larc.utab.ncprofiles	usp.larc.utab.ncprofiles	1.5.0	utility	Foo, Inc.
✓	usp.larc.utab.ncqueues	usp.larc.utab.ncqueues	1.5.0	utility	Foo, Inc.

Figura 31: Aplicações instaladas no controlador ONOS

Number	Traffic class	Service Curve	Service Curve L.O	Flows	Load
of:00000000000000309:4:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 1.0E7 LATENCY: 0.014775	3	75.0
of:00000000000000058:3:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 1.5E7 LATENCY: 0.007299999999999999	2	50.0
of:0000000000000004d:3:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 2.0E7 LATENCY: 0.0024	1	25.0
of:00000000000000016:2:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 2.0E7 LATENCY: 0.0024	1	25.0
of:0000000000000006f:2:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 2.0E7 LATENCY: 0.0024	1	25.0
of:000000000000000de:1:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 2.0E7 LATENCY: 0.0024	1	25.0
of:00000000000000003:4:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 2.0E7 LATENCY: 0.0024	1	25.0
of:000000000000000de:2:0	0	RATE: 2.0E7 LATENCY: 0.0024	RATE: 2.0E7 LATENCY: 0.0024	1	25.0

Figura 32: Rede SDN em termos de curvas de serviço

Nas seguintes seções são detalhados como serão utilizados os componentes desenvolvidos para a automatização dos testes, assim como quais serão as métricas avaliadas e os objetivos específicos de cada experimento.

The screenshot shows the 'Open Network Operating System' interface. At the top, there is a navigation bar with a menu icon, the system name, and a 'logout' link. Below the navigation bar, the title 'Profiles (9 total)' is displayed. A table lists nine traffic profiles with columns for Profile ID, Classification, From, To, and Max Delay. The profiles are: h10-h1-load, h11-h6-load, h13-h8-load, h15-h5-load, h2-h14-video, h3-h14-video, h4-h14-video, h7-h12-load, and h9-h16-load. All profiles have a classification of 0 and a max delay of 100. The 'From' and 'To' columns contain MAC addresses.

Profile ID	Classification	From	To	Max Delay
h10-h1-load	0	92:88:E9:9C:0D:32	B6:29:CE:E1:DB:11	100
h11-h6-load	0	92:88:E9:9C:0D:33	92:88:E9:9C:0D:22	100
h13-h8-load	0	92:88:E9:9C:0D:41	92:88:E9:9C:0D:24	100
h15-h5-load	0	92:88:E9:9C:0D:43	92:88:E9:9C:0D:21	100
h2-h14-video	0	B6:29:CE:E1:DB:12	92:88:E9:9C:0D:42	100
h3-h14-video	0	92:88:E9:9C:0D:13	92:88:E9:9C:0D:42	100
h4-h14-video	0	92:88:E9:9C:0D:14	92:88:E9:9C:0D:42	100
h7-h12-load	0	92:88:E9:9C:0D:23	92:88:E9:9C:0D:34	100
h9-h16-load	0	92:88:E9:9C:0D:31	92:88:E9:9C:0D:44	100

Figura 33: Perfis de tráfego cadastrados no controlador

5.2.1 Balanceamento de carga

O principal objetivo da elaboração dos experimentos nesta Seção é obter resultados que permitam fazer uma comparação com pesquisas recentes relacionadas com a engenharia de tráfego em SDN e, especificamente, ao balanceamento de carga. As métricas para avaliar a eficiência da distribuição do tráfego na rede serão a *vazão do sistema* e a *porcentagem de perda de pacotes*. O conhecimento das rotas utilizadas pelos fluxos numa infraestrutura de rede com dezenas de nós é inviável, devido a isso, neste experimento a capacidade de balanceamento de carga do DSDN-TE será medida pela vazão e perdas percebidas pelos nós destinos de uma transmissão.

Na revisão da literatura, o trabalho realizado em (FIZI; ASKAR, 2016) foi o de maior relevância com respeito do balanceamento de carga utilizando SDN. O algoritmo proposto pelos autores de (FIZI; ASKAR, 2016), quando comparado com ECMP em um cenário de *datacenter*, demonstrou um aumento da vazão na faixa de 21.923% a 36.94%, ademais, reduziu as perdas no sistema em até 90.4%. A implementação do algoritmo proposto não foi disponibilizada pelos autores, mas a clareza com que descrevem as características da topologia utilizada, assim como a metodologia de distribuição do tráfego e métricas de avaliação, fazem possível a reprodução exata dos experimentos, utilizando para isso o mesmo ferramental de emulação e geração de tráfego (i.e., Mininet, iPerf).

Na Subseção seguinte é feita a reprodução dos experimentos realizados em (FIZI; ASKAR, 2016), utilizando DSDN-TE na tarefa de balancear a carga da rede. Isto significa que a presente dissertação não realiza uma reprodução dos resultados obtidos em (FIZI; ASKAR, 2016). Aqui é reproduzido todo o ambiente virtual e cenário de teste (i.e., topologia, distribuição de tráfego, escolha de parâmetros e configurações, métricas avaliadas) utilizado pelos autores em (FIZI; ASKAR, 2016), para depois realizar uma comparação na subseções 5.2.1 e 5.2.2 com os resultados originais publicados em (FIZI; ASKAR, 2016).

5.2.1.1 Ambiente virtual e cenário de teste

A topologia virtual de um *datacenter* foi emulada através da ferramenta Mininet . Esta topologia, responde ao modelo *fat-tree*, amplamente utilizado pela comunidade científica e no mercado. Este modelo possui como principais vantagens a eliminação de um único ponto de falha e a disponibilização de vários caminhos alternativos de igual custo (*hops*) entre qualquer nó final da rede, como pode ser observado na Figura 34. O modelo *fat-tree* possui grande capacidade de escalabilidade e pode ser construído utilizando as seguintes fórmulas:

K-ary fat-tree: Topologia de três camadas (borda, agregação e núcleo).

- Cada *pod* (point of delivery) possui $(k/2)^2$ servidores e duas camadas de $k/2$ comutadores com k-portas;
- Cada comutador de borda está conectado a $k/2$ servidores e $k/2$ comutadores de agregação;
- Cada comutador de agregação está conectado a $k/2$ comutadores de borda e $k/2$ comutadores de núcleo;
- $(k/2)^2$ comutadores de núcleo, onde cada um está conectado a k *pods*.

Neste experimento os enlaces entre comutadores (representados por quadrados) possuem uma capacidade de 10Mbps e os enlaces que conectam os nós finais (*host*) têm capacidade de 20Mbps.

Figura 34: Topologia Fat-Tree $k=4$

Visando emular o funcionamento real de um *datacenter*, foi gerada na rede, utilizando a ferramenta iPerf, uma carga inicial de 15% da capacidade total dos enlaces. O que significa que devem existir transmissões UDP de 1.5Mbps entre diversos pares de *hosts*. A distribuição do tráfego de carga inicial pode ser observada na Tabela 6.

Tabela 6: Nós responsáveis pela geração da carga inicial na rede

Nó origem	Nó destino
h10	h1
h11	h6
h7	h12
h13	h8
h9	h16
h15	h5

As medições neste experimento são realizadas nos *hosts* que recebem os fluxos de dados, a cada 1 segundo, durante 60 segundos. Visando diminuir a influência do sistema operacional no desempenho dos testes, as capacidades dos enlaces são relativamente pequenas e as taxas de transmissão baixas. Cada um dos testes será repetido 10 vezes, buscando uma maior confiabilidade nos dados a serem obtidos.

Inicialmente foi simulada a transmissão de um vídeo com uma taxa de 5Mbps de (h2) e (h3) para (h14). Depois de 20 segundos, é realizada outra transmissão de vídeo com as mesmas características, de (h4) para (h14), gerando assim fluxos concorrentes. Durante todas as transmissões dos vídeos, a rede mantém a carga inicial de 15%.

A vazão medida no host (h14), com relação as transmissões realizadas desde (h2), (h3) e (h4), assim como a porcentagem de perda de pacotes, podem ser observadas nas Figuras 36 e 38 respectivamente.

Figura 35: Algoritmo proposto em (FIZI; ASKAR, 2016)

A Tabela 7 resume os valores de vazão obtidos por ambos mecanismos de balanceamento de carga. Os autores de (FIZI; ASKAR, 2016) não especificam as porcentagens de perdas obtidas na aplicação do algoritmo. Analisando a figura 37, pode-se observar que o sistema começa a se degradar a partir do segundo 20, que corresponde ao início da terceira transmissão. Aproximadamente no segundo 34 o sistema parou de experimentar perdas. Os valores obtidos utilizando DSDN-TE podem ser vistos na Tabela 8.

Figura 36: DSDN-TE

Figura 37: Algoritmo proposto em (FIZI; ASKAR, 2016)

Tabela 7: Aumento da vazão com a utilização de DSDN-TE

Métrica	h2	h3	h4
Vazão (FIZI; ASKAR, 2016)	4.1Mbps	4.12Mbps	4.23Mbps
Vazão DSDN-TE	5.0Mbps	4.91Mbps	4.86Mbps

Tabela 8: Perdas experimentadas pelos fluxos com a utilização de DSDN-TE

Métrica	h2	h3	h4
Porcentagem de perdas	0.38	0.31	0.87

Figura 38: DSDN-TE

5.2.1.2 Discussão

O primeiro aspecto a ser analisado é o modo de funcionamento dos mecanismos comparados na Seção anterior para o balanceamento de carga. Na fase inicial do algoritmo proposto em (FIZI; ASKAR, 2016) é realizada a monitoração das condições da rede (i.e., vazão, perdas de pacotes). Quando uma ou várias métricas de QoS monitoradas alcançam um valor predefinido, devido a uma degradação da rede, a fase de escalonamento é ativada. Esta fase calcula as melhores rotas e atualiza as tabelas de roteamento dos comutadores visando redistribuir a carga da rede.

Diferentemente, o DSDN-TE controla a entrada de novos fluxos na rede através do componente de admissão. Isto explica as perturbações observadas nas Figuras 36 e 38, que representam o tempo utilizado pelo componente de admissão. Quando um fluxo é aceito, o componente responsável pelo balanceamento da carga já disponibilizou para este a rota com menor atraso e atualizou os recursos remanescentes. De forma que, com a chegada de um novo fluxo, a alocação será feita levando em consideração o tráfego presente na rede. Assim a distribuição dos fluxos é realizada onde exista uma maior disponibilidade de recursos.

O trabalho em (FIZI; ASKAR, 2016), preconditiona as transmissões (h2)->(h14),

(h3)->(h14) e (h4)->(h14) para trafegar pela rota (5)->(1)->(17), isto ocasiona uma degradação da rede em termos de vazão e perda de pacotes. O algoritmo em (FIZI; ASKAR, 2016) foi configurado no experimento para redistribuir o tráfego quando for alcançado o limiar de 2.5Mbps ou 20% de perdas em um fluxo de vídeo. Como pode ser observado na Figura 35.

Para distribuir eficientemente a carga da rede, os três fluxos de vídeo deveriam ser balanceados entre os enlaces (17)-(19) e (18)-(19). Nos experimentos, ambos mecanismos (i.e., (FIZI; ASKAR, 2016) e DSDN-TE) balancearam adequadamente, porém, depois da recuperação da rede, DSDN-TE consegue ainda maior vazão. Este fato poderia parecer inconsistente, mas um balanceamento desequilibrado da carga inicial da rede poderia causar este fenômeno. Os autores de (FIZI; ASKAR, 2016), não especificam como foram encaminhados os fluxos iniciais, no entanto fica claro através dos gráficos mostrados que o algoritmo não foi utilizado pois a fase de monitoração não detectou uma degradação na rede. Por sua vez, DSDN-TE balanceia a carga inicial da rede, como pode ser observado na Figura 39.

Figura 39: Vazão da carga inicial

Por outro lado, na Figura 37 podemos observar que ao menos durante 14 segundos os fluxos de vídeo experimentaram perdas. Fazendo uma análise conservadora, juntando as curvas de h2 até alcançar o valor aproximado de 16% de perda de pacotes,

e h3 de 15 à 0% de perda de pacotes, podemos aproximar o mínimo valor de perdas experimentado. A construção desta curva pode ser observada na Figura 40.

Figura 40: Curva construída a partir de h2 e h3

A análise realizada anteriormente significa que a porcentagem de perdas mínima experimentada por ao menos um fluxo de vídeo de 60 segundos em (FIZI; ASKAR, 2016), foi de 0.931%. Comparando com os valores apresentados na Tabela 8 podemos observar que todos os fluxos balanceados por DSDN-TE obtiveram melhores resultados.

5.2.2 Resiliência

O componente para a resiliência, detalhado no Capítulo 4, visa prover ao *framework* proposto neste trabalho capacidade de restaurar os fluxos afetados por alterações nas condições da rede (e.g., falha nos enlaces). Este componente utiliza a estratégia de *restauração*, aplicando rotas alternativas previamente computadas quando um fluxo é aceito na rede.

Considerando o *testbed* do experimento anterior, o objetivo deste experimento é ter condições de medir a vazão do sistema, assim como as perdas de pacotes percebidos pelo nó h14, simulando a falha do enlace por onde trafega o último vídeo transmitido (h4->h14). Para isso foi modificada a topologia utilizada anteriormente (i.e., *fat-tree*

$k=4$) como pode ser observado na Figura 41.

Enlaces entre (1)-(18) e entre (1)-(19) foram adicionados possibilitando que os fluxos de vídeo não utilizem os mesmos enlaces para chegar no nó h14, evitando assim a concorrência. Uma vez induzida a perturbação a rede estaria nas mesmas condições que no experimento de balanceamento de carga. Se esta modificação não fosse realizada os três fluxos de vídeo utilizariam o mesmo enlace e o módulo de admissão rejeitaria um deles.

Figura 41: Topologia fat-tree $k=4$ com enlaces adicionais

Inicialmente foi reproduzido de forma idêntica o experimento descrito na Seção 5.2.1. Depois de 10 segundos do vídeo (h4->h14) ser transmitido, foi simulada uma ruptura do enlace utilizado nesta transmissão. Uma vez alteradas as condições da rede o componente para a resiliência é executado automaticamente. A vazão total alcançada pelo fluxo afetado, assim como a porcentagem de perda de pacotes, podem ser observadas na Figura 42 e 43.

As Tabelas 9 e 10 resumem a comparação entre os valores obtidos com a utilização do componente para a resiliência do *framework* proposto e o método usado no trabalho (FIZI; ASKAR, 2016).

Figura 42: Resiliência do sistema com DSDN-TE

Figura 43: Resiliência do sistema com DSDN-TE

Tabela 9: Aumento da vazão com a utilização de DSDN-TE

Métrica	h4
Vazão (FIZI; ASKAR, 2016)	4.23Mbps
Vazão DSDN-TE	4.85Mbps

Tabela 10: Aumento das perdas com a utilização de DSDN-TE

Métrica	h4
Porcentagem de perdas (FIZI; ASKAR, 2016)	1.40
Porcentagem de perdas DSDN-TE	4.37

5.2.2.1 Discussão

O componente de resiliência especificado neste trabalho possui grande semelhança com o algoritmo proposto em (FIZI; ASKAR, 2016). Ambos são ativados quando as condições da rede são alteradas. Em (FIZI; ASKAR, 2016) a execução do algoritmo depende dos níveis de degradação da rede em termos de vazão e/ou quantidade de perda de pacotes (i.e., 2.5Mbps, 20% de perdas). Em DSDN-TE o componente de resiliência é ativado quando é detectado uma falha nos enlaces da rede. De certa forma é como se fosse ativado quando a rede sofresse uma grande degradação (i.e., vazão igual a 0Mbps e perdas de 100%). Levando estes elementos em consideração poderia-se esperar menor distúrbio nos fluxos redistribuídos pelo método utilizado em (FIZI; ASKAR, 2016); porém a Tabela 9 mostra como a vazão obtida ao longo da transmissão de 40 segundos de (h4->h14) utilizando o componente de resiliência de DSDN-TE é maior.

Observando a Figura 35 é possível deduzir que o sistema demora ao menos 3 segundos partindo do limiar inferior de 2.5Mbps para começar a aumentar gradualmente a vazão. No caso de DSDN-TE a conectividade é interrompida no *slot* de tempo 10-11 e a partir dos doze segundos apresenta sinais de recuperação. Isto permite afirmar que DSDN-TE diminui o tempo de recuperação em 50% com relação ao algoritmo utilizado em (FIZI; ASKAR, 2016). Este fato explica porque o fluxo afetado (h4-h14) possui ganhos em termos de vazão ao longo dos 40 segundos de transmissão.

O comportamento das perdas de pacotes, observado da Figura 43, reafirma ganhos na eficiência da recuperação quando comparado com as Figuras 37 ou 40. Porém a Tabela 10 mostra como a utilização do módulo de resiliência do DSDN-TE gera maior porcentagem de perda de pacotes. Este fenômeno se justifica pelo fato de que o algoritmo utilizado em (FIZI; ASKAR, 2016) é ativado quando o sistema experimenta 20% de perdas. No caso de DSDN-TE o módulo de resiliência atua só quando existe uma ruptura da conectividade.

5.2.3 Aspectos de desempenho

Nesta Seção será feito um estudo do comportamento dos diferentes módulos que formam parte de *framework* proposto, quando são utilizadas diferentes topologias de rede. O objetivo do experimento é caracterizar os componentes em termos de métricas de desempenho (i.e., tempo de execução). Isto permitirá determinar qual componente é o gargalo do sistema e que poderia influenciar negativamente na escalabilidade do *framework*.

Inicialmente foi utilizada a topologia *fat-tree* $k=4$, que é constituída por 64 enlaces, 20 comutadores e 16 *hosts* conforme observado na Figura 34. Foi reutilizado o experimento realizado na Seção 5.2.1 e os resultados em termos de tempos de execução podem ser observados no primeiro conjunto de barras da Figura 44.

Figura 44: Comportamento do sistema com diferentes topologias

Os diversos elementos visualizados na Figura 44 podem ser explicados da seguinte forma:

- **balanceamento:** Tarefa realizada pelo módulo de balanceamento de carga responsável por encontrar o melhor caminho aplicando o Algoritmo 3;
- **admissão:** Corresponde a totalidade das tarefas realizadas no módulo de admissão, detalhadas na Seção 4.2 e na Subseção 4.5.1;

- **atualização:** É a atualização da matriz de tráfego realizada pelo módulo de balanceamento de carga depois que um fluxo é aceito na rede, como foi especificado na Subseção 4.5.1. Devido a relevância desta tarefa foi analisada separadamente;
- **instalação:** Processo de instalação dos fluxos na rede feito pelo controlador SDN. Os detalhes deste processo são explicados nos parágrafos seguintes.

Uma observação a ser feita na Figura 44 é que o tempo do balanceamento é um subconjunto do tempo da admissão devido ao fato que o módulo de admissão utiliza o balanceamento para determinar o atraso máximo que o fluxo ira experimentar. Como pode ser observado na Figura 44, quase o 100% do tempo do módulo de admissão (13,72 ms) é utilizado na tarefa de balanceamento (13,54 ms).

O segundo conjunto de barras da Figura 44 corresponde aos testes realizados utilizando uma topologia *fat-tree* $k=6$, constituída por 216 enlaces, 45 comutadores e 54 *hosts* como pode ser observado na Figura 45.

Figura 45: Topologia fat-tree $k=6$

É evidente que a tarefa de maior custo do *framework*, é a atualização da matriz de tráfego, realizada pelo módulo de balanceamento. Comparando os tempos consumidos em ambas topologias, 296.27 ms e 313.83 ms respectivamente, observa-se um incremento de 5.6%. Uma análise linear destes dados da como resultado que a tarefa

de atualização da matriz de tráfego consumiria $115 \mu s$ para cada enlace adicionado na topologia ou $704 \mu s$ para cada comutador ou $462 \mu s$ para cada *host*.

A tarefa de instalação é medida a partir do instante em que é requisitada a rota na rede (i.e., criação das regras *OpenFlow*), até que o controlador verifique que a rota foi definida. Sendo assim, o tempo efetivo de instalação das regras nos comutadores é menor do que o tempo notificado pelo controlador, que foi de 384,15 ms com a topologia *fat-tree* $k=4$ e 432,02 ms com *fat-tree* $k=6$ respectivamente. A instalação dos fluxos embora não seja função do *framework*, permite calcular o limiar superior do tempo total de funcionamento o qual foi de 694,15 ms e 781,96 ms respectivamente em cada uma das topologias utilizadas.

5.3 Conclusão

No presente Capítulo foi avaliado o *framework* proposto nesta dissertação com foco na capacidade de balancear a carga da rede, assim como de recuperação as perturbações que possam ocorrer na infraestrutura. Para dar subsídio aos testes foram utilizadas ferramentas de virtualização de rede SDN e de geração de tráfego amplamente usadas pela comunidade científica. O trabalho realizado em (FIZI; ASKAR, 2016) foi escolhido como referência para posterior comparação em termos de vazão total do sistema e porcentagem de perdas de pacotes. Quando comparados os resultados na função de balanceamento de carga, o *framework* proposto nesta dissertação mostrou um incremento da vazão total do sistema de 15.7% e uma diminuição da porcentagem de perdas de 44,15%, como pode ser inferido das Tabelas 7 e 8.

Por outro lado a recuperação do sistema quando é utilizado DSDN-TE, foi mais eficiente aumentando em 12.8% a vazão total percebida pelo fluxo afetado. Porém foi detectado um aumento de 67,96% nas perdas de pacotes. Este aumento foi explicado com detalhe na Subseção 5.2.2.1.

Finalizando este Capítulo foi realizado um estudo do comportamento dos diferentes módulos que formam parte de *framework* proposto, quando são utilizadas diferentes topologias de rede. Observou-se que a tarefa de maior custo do *framework* é a atualização da matriz de tráfego realizada pelo módulo de balanceamento a qual foi de até 42% do tempo total utilizado pelo controlador SDN para definir uma rota com balanceamento de carga na rede. Porém, resultados preliminares mostram que o tempo de instalação cresce mais rápido que a atualização da matriz de tráfego quando tem-se uma topologia de rede maior. Isto significa que o gargalo do sistema poderia ser o próprio funcionamento do paradigma SDN e não o tempo consumido pelo *framework* proposto nesta dissertação.

6 CONCLUSÕES

Nesta dissertação foi proposto um *framework* para engenharia de tráfego em redes definidas por software utilizando a teoria matemática *Network Calculus*. Este *framework* é constituído por 4 componentes que permitiram tratar os fluxos de pacotes de forma diferenciada, controlar o acesso do tráfego à infraestrutura, assim como possibilitar um balanceamento de carga eficiente e ainda fornecer capacidade de recuperação a falhas de rede.

Na literatura estudada, a maioria dos trabalhos científicos analisados têm como objetivo o desenvolvimento da arquitetura SDN e dedicam menos esforços na especificação de mecanismos de TE. Os métodos atuais já foram intensamente estudados ou explorados em redes ATM assim como em redes IP baseadas em MPLS. Porém, não estão disponíveis relatos científicos detalhados do desempenho destes métodos considerando diferentes padrões de tráfego, e como obter as informações mais significativas dos fluxos e recursos disponíveis na rede na implantação de uma SDN.

O *framework* proposto visa contribuir na solução dos desafios relacionados a se estabelecer Engenharia de Tráfego para SDN com a especificação de mecanismos de balanceamento de carga dinâmicos e adaptáveis as mudanças topológicas da rede, considerando as características do tráfego como as rajadas e a taxa de transmissão.

O núcleo analítico do *framework*, baseado na teoria NetCalc, permitiu caracterizar os comutadores da rede em termos de *curvas de serviço* e o tráfego em termos de *curvas de chegada*. Isto possibilitou a construção de uma matriz de tráfego responsá-

vel por representar o estado atual da infraestrutura, considerando a disponibilidade de recursos e os custos de transmissão. Através de NetCalc, foi possível possuir o ferramental analítico necessário para que as políticas de distribuição do tráfego estivessem baseadas não só em valores obtidos tradicionalmente monitorando a rede, como também em *previsões determinísticas* do comportamento do tráfego.

A utilização da rede definida por software forneceu capacidade de armazenamento, assim como a disponibilidade e a consistência das informações persistidas na matriz de tráfego. Através de SDN e especificamente do protocolo *OpenFlow*, foi possível manter um controle ativo da infraestrutura de rede o que contribuiu no planejamento e recuperação de forma dinâmica da rede, assim como na utilização eficiente dos recursos disponíveis.

Com o intuito de validar o *framework* foram realizados experimentos que permitiram obter resultados satisfatórios quando comparados com pesquisas recentes sobre engenharia de tráfego em SDN. A capacidade de balanceamento de carga do *framework* demonstrou ganhos em termos de vazão do sistema e permitiu a diminuição da porcentagem de perda de pacotes dos fluxos trafegados.

Com a realização desta dissertação foi comprovado o benefício da integração entre o paradigma de Redes Definidas por Software e a teoria matemática *Network Calculus*. Esta integração demonstrou a viabilidade do uso de métodos tradicionais de avaliação de desempenho no desenvolvimento de mecanismos de engenharia de tráfego. O *framework* proposto avança o estado da arte, provendo uma abordagem analítica que beneficia o gerenciamento e o controle de tráfego em SDN.

Em trabalhos futuros se continuará explorando os benefícios do uso do módulo para o atendimento diferenciado especificado nesta dissertação. Por outro lado pretende-se explorar de forma aprofundada a escalabilidade do *framework*, determinando com precisão os níveis de eficiência quando são utilizadas topologias de maior porte. Além disso se realizará uma análise da consistência das informações persistidas

na matriz de tráfego quando aumenta a concorrência de fluxos na infraestrutura.

Por outra parte será estudado o impacto do uso de valores mínimos e máximos na alocação eficiente dos recursos da rede. Além disso serão analisadas as probabilidades de um fluxo não ser aceito na rede quando existam recursos suficientes e o como poderiam-se utilizar estes recursos ociosos.

Por último pretende-se implantar DSDN-TE no ambiente de experimentação de escala nacional para redes definidas por *software* FIBRE (CIUFFO et al., 2016). Com isso será validado o funcionamento do *framework* usando comutadores de fabricantes comerciais com *hardwares* especializados e diversos *pipelines*. Esta implantação permitirá utilizar enlaces de longa distância com atrasos reais e um maior conjunto de aplicações com diferentes requisitos de QoS.

REFERÊNCIAS

- AGARWAL, S.; KODIALAM, M.; LAKSHMAN, T. V. Traffic engineering in software defined networks. In: *2013 Proceedings IEEE INFOCOM*. [S.l.]: IEEE, 2013. p. 2211–2219. ISBN 978-1-4673-5946-7. ISSN 0743-166X.
- AKYILDIZ, I. F.; LEE, A.; WANG, P.; LUO, M.; CHOU, W. A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, v. 71, p. 1–30, oct 2014. ISSN 13891286.
- AKYILDIZ, I. F.; LEE, A.; WANG, P.; LUO, M.; CHOU, W. Research challenges for traffic engineering in software defined networks. *IEEE Network*, v. 30, n. 3, p. 52–58, may 2016. ISSN 0890-8044.
- AL-FARES, M.; RADHAKRISHNAN, S.; RAGHAVAN, B.; HUANG, N.; VAHDAT, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In: *NSDI*. [S.l.: s.n.], 2010. v. 10, p. 19.
- BENSON, T.; ANAND, A.; AKELLA, A.; ZHANG, M. MicroTE. In: *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies on - CoNEXT '11*. New York, New York, USA: ACM Press, 2011. p. 1–12. ISBN 9781450310413.
- BERDE, P.; GEROLA, M.; HART, J.; HIGUCHI, Y.; KOBAYASHI, M.; KOIDE, T.; LANTZ, B. ONOS: towards an open, distributed SDN OS. *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, p. 1–6, 2014.
- BLAKE, S.; MICROSYSTEMS, S.; WANG, Z. An Architecture for Differentiated Services. *RFC*, p. 1–37, 1998.
- BONDORF, S.; SCHMITT, J. B. The DiscoDNC v2 A Comprehensive Tool for Deterministic Network Calculus. In: *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)*. [S.l.: s.n.], 2014.
- BONDORF, S.; SCHMITT, J. B. On the Potential to Improve Accuracy of Network Calculus Analyses. v. 0, 2015.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *Unified Modeling Language User Guide, The (2Nd Edition)*. [S.l.: s.n.], 2005. ISBN 0321267974.
- BOYER, M. NC-maude: A rewriting tool to play with network calculus. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.: s.n.], 2010. v. 6415 LNCS, p. 137–151. ISBN 3642165575. ISSN 03029743.

BRADEN, R.; CLARK, D.; SHENKER, S. . Integrated Services in the Internet Architecture: an Overview. *IETF RFC 1633, July*, RFC Editor, p. 1–28, jun 1994.

CISCO. *The Zettabyte Era: Trends and Analysis, White Paper*. Cisco, 2016. Disponível em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html>. Acesso em: 31 de agosto de 2017.

CIUFFO, L.; SALMITO, T.; REZENDE, J.; MACHADO, I. Testbed fibre: Passado, presente e perspectivas. In: *Anais do WPEIF 2016 Workshop de Pesquisa Experimental da Internet do Futuro*. [S.l.: s.n.], 2016. p. 3–6.

CRAWLEY, E.; NAIR, R.; RAJAGOPALAN, B.; SANDICK, H. *RFC 2386: A Framework for QoS-based Routing in the Internet*. [S.l.], 1998. (Request for Comments). Disponível em: <<http://www.ietf.org/rfc/rfc2386.txt>>. Acesso em: 31 de agosto de 2017.

CRUZ, R. A calculus for network delay. I. Network elements in isolation. *IEEE Transactions on Information Theory*, v. 37, n. 1, p. 114–131, 1991. ISSN 00189448.

CRUZ, R. A calculus for network delay. II. Network analysis. *IEEE Transactions on Information Theory*, v. 37, n. 1, p. 132–141, 1991. ISSN 00189448.

CURTIS, A. R.; KIM, W.; YALAGANDULA, P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In: *2011 Proceedings IEEE INFOCOM*. [S.l.]: IEEE, 2011. p. 1629–1637. ISBN 978-1-4244-9919-9.

DAVOLI, L.; VELTRI, L.; VENTRE, P. L.; SIRACUSANO, G.; SALSANO, S. Traffic Engineering with Segment Routing: SDN-Based Architectural Design and Open Source Implementation. In: *2015 Fourth European Workshop on Software Defined Networks*. [S.l.]: IEEE, 2015. p. 111–112. ISBN 978-1-5090-0180-4.

DIJKSTRA, E. W. A note on two problems in connection with graphs. *Numerische Mathematik*, v. 1, p. 269–271, 1959. ISSN 0029-599X.

DIXIT, A.; PRAKASH, P.; HU, Y. C.; KOMPELLA, R. R. On the impact of packet spraying in data center networks. In: *2013 Proceedings IEEE INFOCOM*. [S.l.]: IEEE, 2013. p. 2130–2138. ISBN 978-1-4673-5946-7.

EGILMEZ, H.; DANE, S. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. *Signal & Information ...*, p. 1–8, 2012.

EGILMEZ, H. E.; GORKEMLI, B.; TEKALP, A. M.; CIVANLAR, S. Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing. *2011 18th IEEE International Conference on Image Processing*, IEEE, p. 2241–2244, 2011. ISSN 1522-4880.

ENNS, R.; BJORKLUND, M.; BIERMAN, A.; SCHÖNWÄLDER, J. *Network Configuration Protocol (NETCONF)*. RFC Editor, jun 2011. RFC 6241. (Request for Comments, 6241). Disponível em: <<https://rfc-editor.org/rfc/rfc6241.txt>>. Acesso em: 31 de agosto de 2017.

ERICKSON, D. The Beacon OpenFlow Controller. In: ACM. *HotSDN*. [S.l.], 2013.

FILSFILS, C.; PREVIDI, S.; DECRAENE, B.; LITKOWSKI, S.; SHAKIR, R. *Segment Routing Architecture*. [S.l.], jul 2016. Disponível em: <<https://tools.ietf.org/html/draft-ietf-spring-segment-routing-09>>. Acesso em: 31 de agosto de 2017.

FIROIU, V.; Le Boudec, J.-Y.; TOWSLEY, D. Theories and models for Internet quality of service. *Proceedings of the IEEE*, v. 90, n. 9, p. 1565–1591, sep 2002. ISSN 0018-9219.

FIZI, F. S.; ASKAR, S. A novel load balancing algorithm for software defined network based datacenters. In: *2016 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom)*. [S.l.]: IEEE, 2016. p. 1–6. ISBN 978-1-5090-2270-0.

FOSTER, N.; HARRISON, R.; FREEDMAN, M. J.; MONSANTO, C.; REXFORD, J.; STORY, A.; WALKER, D. Frenetic: A network programming language. In: *Proceeding of the 16th ACM SIGPLAN international conference on Functional programming - ICFP '11*. [S.l.: s.n.], 2011. v. 46, p. 279. ISBN 9781450308656. ISSN 03621340.

GOOGLE. Inter-Datacenter WAN with centralized TE using SDN and OpenFlow. In: *Open Networking Summit*. [S.l.: s.n.], 2012. p. 1–2.

GUTIÉRREZ, P. A. A.; LÓPEZ, D. R. *Fighting Your Way Through the Jungle of Intent*. sep 2016. Disponível em: <<http://sdn.ieee.org/newsletter/september-2016/fighting-your-way-through-the-jungle-of-intent>>. Acesso em: 31 de agosto de 2017.

HAILONG, Z.; XIAO, G.; JINYAO, Y.; BO, L.; QIANJUN, S. SDN-based ECMP algorithm for data center networks. In: *2014 IEEE Computers, Communications and IT Applications Conference*. [S.l.]: IEEE, 2014. p. 13–18. ISBN 978-1-4799-4811-6.

HINRICHS, T.; GUDE, N.; CASADO, M.; MITCHELL, J.; SHENKER, S. Practical declarative network management. *Proceedings of the 1st ACM workshop on Research on Enterprise Networking*, p. 1–10, 2009. ISSN 01464833.

HUANG, W.-Y.; CHOU, T.-Y.; HU, J.-W.; LIU, T.-L. Automatic End to End Topology Discovery and Flow Viewer on SDN. In: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*. [S.l.]: IEEE, 2014. p. 910–915. ISBN 978-1-4799-2653-4.

JAIN, S.; KUMAR, A.; MANDAL, S.; ONG, J.; POUTIEVSKI, L.; SINGH, A.; VENKATA, S.; WANDERER, J.; ZHOU, J.; ZHU, M.; ZOLLA, J.; HÖLZLE, U.; STUART, S.; VAHDAT, A. B4: experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, ACM, v. 43, n. 4, p. 3–3–14–14, sep 2013. ISSN 0146-4833.

JIANG, Y. Network calculus and queueing theory: two sides of one coin: invited paper. In: *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. [S.l.: s.n.], 2009. p. 1–12. ISBN 9789639799707.

J.MOY. OSPF Version 2. *IETF RFC 2328, April*, p. 1–245, 1998.

KREUTZ, D.; RAMOS, F. Software-Defined Networking: A Comprehensive Survey. *arXiv preprint arXiv: ...*, p. 49, 2014.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. ... *Workshop on Hot Topics in Networks*, p. 1–6, 2010. ISSN 1450304095.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G. M.; PETERSON, L. L.; REXFORD, J.; SHENKER, S.; TURNER, J. S.; LOUIS, S. OpenFlow: enabling innovation in campus networks. *Computer Communication Review*, v. 38, p. 69–74, 2008. ISSN 01464833.

MONSANTO, C.; FOSTER, N.; HARRISON, R.; WALKER, D. *A compiler and run-time system for network programming languages*. 2012. 217 p.

NG, E. Maestro: A system for scalable openflow control. *Rice University*, 2010.

NI, L. Internet QoS: a big picture. *IEEE Network*, v. 13, n. 2, p. 8–18, 1999. ISSN 08908044.

NUNES, B. A. a.; MENDONCA, M.; NGUYEN, X. N.; OBRACZKA, K.; TURLETTI, T. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys and Tutorials*, v. 16, n. 3, p. 1617–1634, 2014. ISSN 1553877X.

ONF. *OpenFlow Switch Specification version 1.0.0*. 2009. Disponível em: <<http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf>>. Acesso em: 31 de agosto de 2017.

ONF. *OpenFlow Switch Specification version 1.2*. 2011. Disponível em: <<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf>>. Acesso em: 31 de agosto de 2017.

ONF. *OpenFlow Switch Specification version 1.3.0*. 2012. Disponível em: <<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>>. Acesso em: 31 de agosto de 2017.

ONF. *Software-Defined Networking : The New Norm for Networks*. 2012. 1–12 p. Disponível em: <<https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>>. Acesso em: 31 de agosto de 2017.

ON.LAB. *Intent Framework - ONOS - Wiki*. 2016. Disponível em: <<https://wiki.onosproject.org/display/ONOS/Intent+Framework>>. Acesso em: 31 de agosto de 2017.

OPENDAYLIGHT. *OpenDaylight: A Linux Foundation Collaborative Project*. 2013. Disponível em: <<http://www.opendaylight.org>>. Acesso em: 31 de agosto de 2017.

OPENDAYLIGHT. *Network Intent Composition*. 2015. Disponível em: <https://wiki.opendaylight.org/view/Network_Intent_Composition:Main>. Acesso em: 31 de agosto de 2017.

OPENSOURCESDN. *Project Boulder*. Disponível em: <<http://opendaylight.org/projects/project-boulder-intent-northbound-interface-nbi/>>. Acesso em: 31 de agosto de 2017.

OWENS, H.; DURRESI, A. Video over Software-Defined Networking (VSDN). In: *2013 16th International Conference on Network-Based Information Systems*. [S.l.]: IEEE, 2013. p. 44–51. ISBN 978-1-4799-2510-0.

PADMA, V.; YOGESH, P. Proactive failure recovery in OpenFlow based Software Defined Networks. In: *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*. [S.l.]: IEEE, 2015. p. 1–6. ISBN 978-1-4673-6823-0.

PAKZAD, F.; PORTMANN, M.; TAN, W. L.; INDULSKA, J. Efficient topology discovery in software defined networks. In: *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*. [S.l.]: IEEE, 2014. p. 1–8. ISBN 978-1-4799-5255-7.

PFAFF, B.; DAVIE, B. *The Open vSwitch Database Management Protocol*. [S.l.], 2013. Disponível em: <<https://rfc-editor.org/rfc/rfc7047.txt>>. Acesso em: 31 de agosto de 2017.

PFAFF, B.; PETTIT, J.; KOPONEN, T.; AMIDON, K.; CASADO, M.; SHENKER, S. Extending Networking into the Virtualization Layer. In: *8th ACM Workshop on Hot Topics in Networks*. [s.n.], 2009. VIII, p. 6. Disponível em: <<http://www.icsi.berkeley.edu/pubs/networking/extendingnetworking09.pdf>>. Acesso em: 31 de agosto de 2017.

QIN, Z.; DENKER, G.; GIANNELLI, C.; BELLAVISTA, P.; VENKATASUBRAMANIAN, N. A Software Defined Networking architecture for the Internet-of-Things. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. [S.l.]: IEEE, 2014. p. 1–9. ISBN 978-1-4799-0913-1.

ROSEN, E.; VISWANATHAN, A.; CALLON, R. Multiprotocol label switching architecture. *Vasa*, p. 1–62, 2001.

SALSANO, S.; VENTRE, P. L.; LOMBARDO, F.; SIRACUSANO, G.; GEROLA, M.; SALVADORI, E.; SANTUARI, M.; CAMPANELLA, M.; PRETE, L. Hybrid IP/SDN Networking: Open Implementation and Experiment Management Tools.

IEEE Transactions on Network and Service Management, v. 13, n. 1, p. 138–153, mar 2016. ISSN 1932-4537.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. OpenStack: Toward an Open-Source Solution for Cloud Computing. *International Journal of Computer Applications*, v. 55, n. 03, p. 38–42, 2012. ISSN 09758887.

SHARMA, S.; STAESSENS, D.; COLLE, D.; PICKAVET, M.; DEMEESTER, P. Enabling fast failure recovery in OpenFlow networks. In: *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*. [S.l.]: IEEE, 2011. p. 164–171. ISBN 978-1-61284-125-0.

SHENKER, S.; PARTRIDGE, C.; GUERIN, R. RFC-2212: Specification of Guaranteed Quality of Service. 1997.

SHU, Z.; WAN, J.; LIN, J.; WANG, S.; LI, D.; RHO, S.; YANG, C. Traffic engineering in software-defined networking: Measurement and management. *IEEE Access*, v. 4, p. 3246–3256, 2016. ISSN 2169-3536.

SMIT, H.; LI, T. Intermediate System to Intermediate System (IS-IS) Extensions for Traffic Engineering (TE). *RFC 3784*, 2004. Disponível em: <<http://www.ietf.org/rfc/rfc3784.txt>>. Acesso em: 31 de agosto de 2017.

SZIGETI, T.; HATTINGH, C. *End-to-end QoS network design: quality of service in LANs, WANs, and VPNs*. [S.l.]: Cisco Press, 2004.

VINOSKI, S. RESTful web services development checklist. *IEEE Internet Computing*, v. 12, p. 440, 2008. ISSN 10897801. Disponível em: <<http://medcontent.metapress.com/index/A65RM03P4874243N.pdf>>. Acesso em: 31 de agosto de 2017.

WROCLAWSKI, J. RCF-2211: Specification of the Controlled-Load Network Element Service. 1997. Disponível em: <<http://rfc.askapache.com/rfc2211/rfc2211.txt>>. Acesso em: 31 de agosto de 2017.

YAN, J.; ZHANG, H.; SHUAI, Q.; LIU, B.; GUO, X. HiQoS: An SDN-based multipath QoS solution. *China Communications*, v. 12, n. 5, p. 123–133, may 2015. ISSN 1673-5447.

YEGANEH, S.; TOOTOONCHIAN, A.; GANJALI, Y. *On scalability of software-defined networking*. [S.l.]: Communications Magazine, IEEE, 2013. 136–141 p.

ZHANG, Y.; XIA, Y.; JIANG, S.; ZHOU, T.; HARES, S. *NEMO NEtwork MOdeling Language*. [S.l.], apr 2016. Disponível em: <<https://tools.ietf.org/html/draft-xia-sdnrg-nemo-language-04>>. Acesso em: 31 de agosto de 2017.