DASH sobre OpenFlow: estimando métricas de QoS a partir da rede

Marta Calasans Costa Lacerda

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2020

Marta Calasans Costa Lacerda

DASH sobre OpenFlow: estimando métricas de QoS a partir da rede

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Rafael Pasquini

Uberlândia
2020

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

L131 Lacerda, Marta Calasans Costa, 1983-
2020 DASH sobre OpenFlow: estimando métricas de QoS a partir da rede [recurso eletrônico] / Marta Calasans Costa Lacerda. - 2020.

Orientador: Rafael Pasquini.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2020.305
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Pasquini, Rafael,1981-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Ciência da Computação. III. Título.

CDU: 681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Dissertação de Mestrado Acadêmico, 08/2020, PPGCO				
Data:	28 de fevereiro de 2020	Hora de início:	**14:00**	Hora de encerramento:	**17:00**
Matrícula do Discente:	11712CCP009				
Nome do Discente:	Marta Calasans Costa Lacerda				
Título do Trabalho:	DASH sobre OpenFlow: estimando métricas de QoS a partir da rede				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se na sala 1B132, Bloco 1B, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Rodrigo Sanches Miani - FACOM/UFU, Fábio Luciano Verdi - DC/UFSCAR e Rafael Pasquini - FACOM/UFU, orientador da candidata.

Ressalta-se que o Prof. Dr. Fábio Luciano Verdi participou da defesa por meio de videoconferência desde a cidade de Sorocaba - SP. Os outros membros da banca e a aluna participaram in loco.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Rafael Pasquini, apresentou a Comissão Examinadora e a candidata, agradeceu a presença do público, e concedeu à Discente a palavra para a exposição do seu trabalho. A duração da apresentação da Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir a candidata. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando a candidata:

Aprovada

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Rodrigo Sanches Miani**, **Professor(a) do Magistério Superior**, em 04/03/2020, às 11:17, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Rafael Pasquini**, **Professor(a) do Magistério Superior**, em 04/03/2020, às 11:30, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fábio Luciano Verdi**, **Usuário Externo**, em 04/03/2020, às 13:19, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1892106** e o código CRC **000B59FE**.

Este trabalho é dedicado a minha família.

Agradecimentos

Até a conclusão deste trabalho, um longo caminho foi percorrido. Houve algumas pausas e recomeços, durante muitos anos. Eu agradeço a todos aqueles que fizeram parte desta história. Seria difícil enumerá-los aqui, mas todos têm a minha verdadeira gratidão. Limito-me apenas a agradecer nominalmente a alguns que mais diretamente colaboraram para que este trabalho se realizasse, nestes anos finais, podendo estes ser considerados representantes de todos os aqui não citados:

- ao Centro de Tecnologia de Informação da UFU, especialmente à equipe da Divisão de Redes e ao diretor, Prof. Dr. Luís Fernando Faina;
- aos funcionários e professores do programa de Pós-graduação da Faculdade de Computação da UFU, em particular a Erisvaldo Fialho pelo profissionalismo de sempre;
- aos colegas, por todos os momentos compartilhados, em especial a Aryadne Guardieiro, Gustavo Silveira, Juliano Matos, Muriel Alves, Pedro Damaso e Rafael Marinho por se disponibilizarem a auxiliar diretamente em etapas deste trabalho;
- ao meu orientador, Prof. Dr. Rafael Pasquini, pelo incentivo, orientação, paciência e pela empatia demonstradas ao longo desses anos;
- aos membros da banca pela disponibilidade para avaliar o trabalho;
- à equipe da DIRQS da UFU, principalmente ao médico Dr. Tiago Castro e Couto, pelo profissionalismo, empatia e disponibilidade.
- aos meus pais, Marisaide e Faustino, e minha irmã, Fernanda, e sua família, por todo amor compartilhado desde sempre e por se fazerem presentes em todos os instantes da minha vida, apesar da distância física;
- a Deus, ao Divino Mestre Jesus e ao "Amigo que me guarda sempre".

É melhor estar aproximadamente certo do que exatamente errado.
(Carveth Read, 1848-1931)

Resumo

Obter métricas de QoS no cliente não é algo trivial para a maioria dos provedores de serviço em rede. Trabalhos recentes indicaram que, para redes OpenFlow, parte dessas informações podem ser extraídas da rede, utilizando-se apenas estatísticas de tráfego agregado como dados de entrada para métodos de aprendizagem de máquina capazes de gerar estimadores. O objetivo da pesquisa realizada e aqui apresentada foi prosseguir com essa investigação, aplicando a mesma metodologia a uma das aplicações de maior ascensão na Internet, ainda não investigada: as aplicações de vídeo sob demanda com taxa de bit dinamicamente adaptável. Para isso, foi implementado um ambiente de testes com um serviço DASH, cuja comunicação entre servidor e clientes é realizada via rede OpenFlow, com comutadores físicos. Nesse ambiente, foram coletados os dados para geração dos modelos empregando-se dois métodos de aprendizagem de máquinas supervisionada: árvore de regressão e floresta aleatória. As métricas de QoS investigadas foram a taxa de quadros por segundo (componente de vídeo) e a taxa de *buffers* por segundo (componente de áudio), as quais foram avaliadas segundo o erro absoluto médio normalizado (NMAE) e o tempo de treinamento do modelo. Para o cenário em análise, os resultados foram extremamente satisfatórios com relação ao tempo de treinamento para as duas métricas de QoS investigada, ficando na ordem de milissegundos. No que se refere ao erro, a métrica de áudio apresentou desempenho melhor para os algoritmos estudados, ficando em torno de 13%, enquanto que a de vídeo obteve erros pouco acima de 16,5%. Também foi verificado que o emprego do método *ensemble* não trouxe benefícios significativos aos resultados. Além disso, os resultados revelaram que o mesmo conhecimento sobre essas métricas poderia ser gerado empregando-se predominantemente estatísticas de dados agregadas do comutador que conecta o cliente à rede OpenFlow. Todos esses resultados, entretanto, precisam ser melhor investigados em trabalhos futuros que empreguem mais amostras de dados e tornem o ambiente de teste cada vez mais próximo do real.

Palavras-chave: DASH. HAS. OpenFlow. SDN. QoS. Aprendizagem de máquina.

Abstract

Obtaining QoS metrics on the client is not trivial for most networked service providers. Recent work has indicated that, for OpenFlow networks, part of this information can be extracted from the network, using only aggregated traffic statistics as input data for machine learning methods capable of generating estimators. The objective of the research carried out and presented here was to proceed with this investigation, applying the same methodology to one of the most popular applications on the Internet, not yet investigated: video on demand applications with dynamically adaptable bit rates. For this, a test environment was implemented with a DASH service, whose communication between server and clients is carried out via the OpenFlow network, with physical switches. In this environment, data were collected to generate the models using two supervised machine learning methods: regression tree and random forest. The QoS metrics investigated were the frame rate per second (video component) and the textit buffer rate per second (audio component), which were evaluated according to the normalized mean absolute error (NMAE) and time training model. For the scenario under analysis, the results were extremely satisfactory with respect to the training time for the two QoS metrics investigated, staying in the order of milliseconds. With regard to the error, the audio metric showed better performance for the studied algorithms, staying around 13 %, while the video metric had errors just above 16.5 %. It was also found that the use of the ensemble method did not bring significant benefits to the results. In addition, the results revealed that the same knowledge about these metrics could be generated using predominantly aggregated data statistics from the switch that connects the client to the OpenFlow network. All of these results, however, need to be better investigated in future works that employ more data samples and bring the test environment closer to the real one.

Keywords: DASH. HAS. OpenFlow. SDN. QoS. Machine Learning.

Lista de ilustrações

Lista de tabelas

Lista de siglas

3GPP	3rd Generation Partnership Project
A-CPI	Application-Controller Plane Interface
ABR	Adaptive Bitrate Streaming
API	Application Programming Interface
AVC	Advanced Video Coding
BPP	Bits per Pixel
CDN	Content Delivery Network
CRC	Cyclic Redundancy Check
D-CPI	Data-Controller Plane Interface
DASH	Dynamic Adaptive Streaming over HTTP
DNS	Domain Name System
FPS	Frames per Second
GOP	Group of Pictures
HAS	HTTP Adaptive Streaming
HDS	Adobe HTTP Dynamic Streaming
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IPTV	Internet Protocol Television

LTE	Long Term Evolution
MP4	MPEG-4
MPD	Media Presentation Description
MPEG	Moving Picture Experts Group
MSS	Microsoft Smooth Streaming
NAT	Network Address Translations
NBI	Northbound Interface
NMAE	Normalized Mean Absolute Error
NOS	Networking Operating System
NTP	Network Time Protocol
ONF	Open Networking Foundation
OVS	Open vSwitch
QoS	Quality of Service
QoE	Quality of Experience
RSS	Residual Sum of Square
RTP	Real Time Transfer Protocol
RTT	Round Trip Time
RTSP	Real Time Streaming Protocol
RUSBoost	Random Undersampling Boosting
SBI	Southbound Interface
SDN	Software-Defined Networking
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus
XML	Extensible Markup Language

Sumário

Capítulo 1

Introdução

Um fato incontestável na sociedade atual é a popularização do uso das aplicações multimídia na Internet. Os sítios especializados em compartilhamento de conteúdo em vídeo tornaram possível que os usuários os utilizem inclusive como fonte de renda, como uma atividade profissional (BBC BRASIL, 2017). Os novos serviços de vídeo sob demanda via Internet têm angariado facilmente novos adeptos devido à comodidade gerada pelo acesso ubíquo a um custo relativamente baixo. Até mesmo as companhias de transmissão tradicionais, como as emissoras de TV aberta e a cabo, têm se adaptado a essa realidade e disponibilizado conteúdo sob demanda via Internet aos seus espectadores (SALGADO, 2017). As redes sociais também são responsáveis por essa disseminação, existindo inclusive aquelas fundamentadas na interação via formato multimídia (KARHOFF, 2017). Outra área em que essas aplicações estão em ascensão é a educacional, ampliando a chamada educação a distância (BATAEV, 2017). O uso dessas aplicações não se limita a conteúdo estático, mas também a conteúdo em tempo real, como transmissões ao vivo e videoconferências (CISCO, 2018).

Nesse contexto, novos desafios foram surgindo para manutenção e garantia de qualidade de serviço (*Quality of Service* - QoS). A popularização também gerou usuários mais exigentes. As limitações para garantia de QoS das novas aplicações no modelo TCP/IP tradicional, dentre outras questões (como: complexidade, políticas inconsistentes, dependência de fornecedor (ONF, 2012)), impulsionaram novas pesquisas e as redes de nova geração são agora uma realidade. Dentre elas, destacam-se as redes definidas por software (*Software-Defined Networking* - SDN), mais especificadamente o padrão OpenFlow (ONF, 2019). A principal característica dessas redes é a separação entre o plano de controle e o plano de dados, viabilizando um melhor gerenciamento das redes e da qualidade dos serviços fornecidos (KARAKUS; DURRESI, 2017).

No âmbito das aplicações, novos métodos de codificação e transmissão dos dados foram propostos nos últimos anos, principalmente na tentativa de adaptar o vídeo às limitações de transmissão das redes e contornar os bloqueios de segurança, como os impostos pelos *firewalls*. O objetivo principal desses novos métodos e aplicações é garantir a satisfação do

usuário e permitir que o conteúdo se adéque aos vários dispositivos pelos quais ele pode ser acessado. O padrão DASH (*Dynamic Adaptive Streaming over HTTP*) (ISO, 2014) foi desenvolvido com este objetivo, pois permite o acesso via protocolo HTTP e possibilita que o vídeo seja entregue dinamicamente ao cliente na resolução mais apropriada, adaptando-se periodicamente às oscilações de processamento ou de capacidade de transmissão do caminho até o cliente (SODAGAR, 2011).

Paralelamente a essas questões, houve um avanço no desenvolvimento e emprego de técnicas de aprendizagem de máquina em diversas áreas (JORDAN; MITCHELL, 2015). A aprendizagem de máquina constitui o estudo e a modelagem computacional do processo de aprendizagem nas suas múltiplas manifestações (MICHALSKI; CARBONELL; MITCHELL, 1983). Classicamente, objetiva desenvolver modelos que aprendam a partir de dados prévios (características e resultados), numa abordagem supervisionada, ou que sejam capazes de inferir conhecimento a partir de atributos dos dados, numa perspectiva não supervisionada. Mas novas técnicas têm surgido como aprendizagem de reforço (*reinforcement learning*) e aprendizagem profunda (*deep learning*). Todas elas têm sido amplamente usadas em redes de computadores seja para classificação de tráfego, detecção de intrusão, escolha de rotas, monitoramento de QoS, dentre outros (SHAFIQ et al., 2016; BUCZAK; GUVEN, 2016; WANG et al., 2017; ZHAO et al., 2019).

Dessa forma, é mais que justificável a realização de pesquisas relacionadas a QoS de aplicações DASH em redes SDN, como o trabalho aqui apresentado. O projeto desenvolvido objetivou entender como métricas de QoS observadas no cliente estão relacionadas com as estatísticas de tráfego agregado observadas nos dispositivos de rede. Para isso, empregou-se a metodologia proposta em Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017), nos quais os autores propõem a aplicação de métodos de aprendizagem de máquina supervisionada como forma de modelar essa relação, extraindo conhecimento a partir da rede. A aprendizagem de máquina foi escolhida porque a intenção é, futuramente, com o aprofundamento das pesquisas, desenvolver métodos de monitoramento de QoS em tempo real. Os resultados obtidos foram então comparados com os alcançados nesses estudos para a aplicação de vídeo sob demanda não adaptável.

1.1 Motivação

Conforme citado anteriormente, estudos anteriores (PASQUINI; STADLER, 2017; STADLER; PASQUINI; FODOR, 2017) indicaram que é possível extrair conhecimento da rede relativos à qualidade de serviços utilizando métodos de aprendizagem de máquina. Portanto, a principal motivação desta pesquisa foi prosseguir com essa investigação, aplicando a metodologia proposta nesses trabalhos a uma das aplicações de maior ascensão na Internet: as aplicações de vídeo sob demanda com taxa de bit dinamicamente adaptável.

Apesar de também se tratar de uma aplicação de vídeo, como a já estudada, a apli-

cação usada neste trabalho apresenta taxa de bit dinamicamente adaptável, ao passo que a previamente estudada possuía taxa de bit alvo fixa. Essa diferença, ou seja, o fato da taxa de transmissão ajustar-se dinamicamente conforme a sobrecarga na rede ou no processamento, poderia alterar a forma como as métricas de QoS se comportariam, quais delas se fariam válidas, e, consequentemente, tornou a aplicação DASH digna de um novo estudo com a mesma metodologia.

O monitoramento de QoS a partir da rede é relevante, pois, do ponto de vista dos provedores de serviços, nem sempre é possível obter informações nos clientes. Obter essa informação de forma precisa e automatizada possibilitaria um melhor controle dos serviços e, consequentemente, uma melhor experiência para os usuários. Além disso, como o DASH se tornou o padrão preferencial para a transmissão de vídeo (KUA; ARMITAGE; BRANCH, 2017), é interessante analisar se há formas mais eficientes para o gerenciamento de capacidade, utilização otimizada dos recursos e melhoria da qualidade de serviço.

1.2 Objetivos e Desafios da Pesquisa

O objetivo geral do projeto foi investigar como métricas de qualidade de serviço em um cliente DASH se relacionam com estatísticas de tráfego de rede agregado em uma rede OpenFlow. Ou seja, se os indícios de que é possível extrair conhecimento da rede sobre métricas desse tipo, obtidos em trabalhos anteriores, confirmam-se quando a aplicação em questão é de transmissão de vídeo com taxa de bit dinamicamente adaptável sobre HTTP.

De forma mais específica, a pesquisa focou nas seguintes ações:

a) Examinar conjuntos de atributos para obter um conjunto satisfatório para elaboração dos modelos;

b) Investigar se os métodos de árvore de regressão e de floresta aleatória apresentam resultados satisfatórios ou se outros métodos devem ser empregados quando a transmissão de vídeo é dinamicamente adaptável;

c) Implementar um ambiente de experimentação e mecanismos de automatização de experimentos que possam nortear trabalhos futuros relacionados ao projeto;

d) Empregar *switches* OpenFlow físicos no ambiente de experimentação proposto.

Um dos desafios da pesquisa era encontrar o conjunto de estatísticas de rede que maximizasse a acurácia e minimizasse o tempo de treinamento. Como isso talvez não fosse possível, buscou-se obter um modelo com acurácia e tempo de processamento satisfatórios. Outro desafio da pesquisa foi tentar integrar os *switches* OpenFlow físicos, obtendo dele as mesmas estatísticas a partir do controlador, uma vez que os trabalhos anteriores utilizaram comutadores virtualizados.

1.3 Hipótese

O trabalho fundamentou-se na hipótese de que é possível empregar a metodologia proposta por Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017) – e descrita na Seção 3.3 – para estimar métricas de QoS de uma aplicação DASH a partir de estatísticas de tráfego de dispositivos da rede SDN subjacente, utilizando métodos de aprendizagem de máquina. A possibilidade dessa predição, por assim dizer, contribuiria para o monitoramento de QoS, principalmente se o mesmo pudesse ser efetuado em tempo real.

Nesse contexto, alguns questionamentos se mostravam válidos:

a) Quais estatísticas de rede geram melhores estimativas de quais medidas de QoS?

b) Quais dos métodos de aprendizagem de máquina geram melhores estimativas de quais medidas de QoS?

c) O tempo computacional obtido com os métodos empregados é indicativo para a possibilidade de monitoramento em tempo real?

d) Quais dispositivos da rede mais influenciam os resultados: os de maiores cargas, mais próximos do cliente, ou mais próximos do servidor?

e) Quais parâmetros dos métodos de aprendizagem de máquina empregados influenciam nos resultados, seja na precisão ou no tempo de processamento?

f) Os resultados obtidos com o serviço DASH se assemelham com os já obtidos com aplicações de vídeo sob demanda não adaptável?

1.4 Contribuições

Uma das contribuições do trabalho é ter investigado se o comportamento de uma aplicação de vídeo com taxa de bit dinamicamente adaptável é semelhante a uma de vídeo sob demanda padrão, no que se refere à extração de conhecimento sobre o cliente a partir dos dispositivos de rede. A metodologia e os métodos de aprendizagem de máquina utilizados neste projeto baseiam-se nos empregados em Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017), de modo a permitir uma comparação com os resultados alcançados anteriormente.

Um resultado positivo sobre a acurácia das estimativas evidenciaria que o tráfego agregado de rede pode oferecer mais informações sobre os serviços do que comumente se extrai dele. Os resultados demonstraram que o erro obtido para o serviço DASH para a métrica de vídeo quadros por segundo é maior do que para a aplicação de vídeo sob demanda tradicional, mas, para a métrica de áudio *buffers* por segundo, ele é menor. Entretanto, nenhuma das métricas analisadas obteve erro inferior a 10%. Isso não necessariamente inviabiliza o uso dessa metodologia, mas direciona os estudos para incluir outros métodos

de aprendizagem de máquina ou outras estatísticas de tráfego agregado, como estatísticas por fluxo. Além disso, o tamanho menor do *trace* utilizado (série temporal formada com as amostras coletadas) para geração dos modelos pode ter influenciado na queda da acurácia, o que deve ser analisado em trabalhos futuros.

A investigação sobre o tempo de computação dos modelos contribuiu para indicar se essa extração de conhecimento pode ser feita ou não em tempo real (ou próxima disso). Como os melhores resultados foram obtidos quando se empregou um número relativamente pequeno de estimadores (árvores) e deram-se na ordem de milissegundos, há uma indicação de que é possível chegar a tempos satisfatórios mesmo para séries temporais com mais amostras. Para conjuntos de atributos maiores, métodos de redução de atributos podem ser empregados para se manter o tempo satisfatório.

1.5 Organização da Dissertação

O Capítulo 2 aborda os principais conceitos teóricos empregados neste trabalho e necessários à sua compreensão. Inicialmente, é apresentada uma visão geral do paradigma SDN e de como ele se diferencia das redes tradicionais TCP/IP. Em seguida, os fundamentos do padrão DASH e da codificação dos vídeos são descritos. A abordagem de aprendizagem de máquina supervisionada, em especial a regressão, os métodos de árvore de regressão e de florestas aleatórias, e os métodos de redução de atributos utilizados aparecem na sequência. Nesse contexto também é descrito o erro absoluto médio normalizado, utilizado para avaliação dos resultados. Por fim, são relacionados os trabalhos correlatos.

No Capítulo 3, o ambiente de experimentação é apresentado, expondo as particularidades de cada componente e como ocorre a interação entre eles. Além disso, são descritas quais estatísticas de rede são coletadas, bem como os métodos de obtenção de cada uma delas. Também são especificadas as métricas de serviço monitoradas no cliente e como este monitoramento é realizado. Finalmente, discorre-se sobre como essas informações são utilizadas na construção das séries temporais que são usadas como entrada para a aprendizagem e validação dos modelos.

Os experimentos realizados (e como os modelos são gerados) são detalhados no Capítulo 4. Para cada um desses experimentos, é feito um exame minucioso dos dados obtidos. As conclusões decorrentes dessas análises são sumarizadas no Capítulo 5. Nesse último capítulo, são feitas considerações finais, indicando pontos positivos e negativos da pesquisa desenvolvida, as contribuições alcançadas são apresentadas, e são descritas propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo, discorre-se sobre os fundamentos dos três principais temas envolvidos na pesquisa e necessários ao seu entendimento: o paradigma SDN (Seção 2.1); o padrão DASH (Seção 2.2); e a aprendizagem de máquina supervisionada (Seção 2.3). Por fim, são relacionados os principais trabalhos correlatos (Seção 2.4).

2.1 O Paradigma SDN

O paradigma SDN surgiu em meio a necessidade de separação dos planos de controle e dados, os quais são fortemente integrados nas redes tradicionais. As redes em uso seguem uma abordagem centrada em hardware (*hardware-centering networking*), a qual se mostrou ineficiente diante das novas demandas de serviços surgidas na última década. Consequentemente, isso levou ao desenvolvimento de propostas alternativas que considerassem uma abordagem centrada na informação (*information-centering networking*) (FARHADY; LEE; NAKAO, 2015; MASOUDI; GHAFFARI, 2016).

Como pode ser visto na Figura 1, as redes de computadores podem ser divididas em três planos/camadas de funcionalidades (KREUTZ et al., 2015):

a) Plano de gerenciamento[1]: onde as políticas da rede são definidas. Inclui os serviços de software;

b) Plano de controle: onde as políticas da rede são aplicadas. Engloba os protocolos utilizados para popular as tabelas de encaminhamento dos componentes do plano de dados;

c) Plano de dados[2]: onde as políticas de rede são executadas, por meio do encaminhamento de dados apropriado. Corresponde, desse modo, aos dispositivos de rede

[1] Alguns autores utilizam o termo plano/camada de aplicação em vez de plano de gerenciamento (FARHADY; LEE; NAKAO, 2015; KARAKUS; DURRESI, 2017; MASOUDI; GHAFFARI, 2016; ONF, 2012; XIA et al., 2015).

[2] Também chamado de camada de infraestrutura (JARRAYA; MADI; DEBBABI, 2014; ONF, 2012) ou plano de encaminhamento (SON; BUYYA, 2018).

(comutadores, roteadores, etc.).

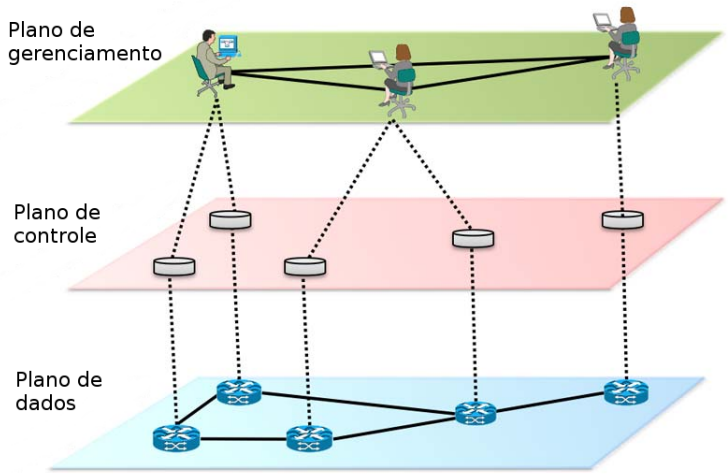

Figura 1 – Visão das redes em termos de funcionalidade.

Fonte: Adaptado de Kreutz et al. (2015, p. 17).

Uma consequência da integração dos planos de dados e controle nas redes tradicionais é a implementação dos mesmos de forma descentralizada nos dispositivos de rede (MASOUDI; GHAFFARI, 2016). Embora esse modelo distribuído possa ser visto como uma das razões para o crescimento e popularização das redes, por permitir resiliência e proporcionar um bom desempenho, ele tornou-se um obstáculo para desenvolvimento de novos protocolos e serviços de rede, pois gerou redes complexas, dificultando a gerência e a reconfiguração em termos de resposta a falhas, carga e outras mudanças (KREUTZ et al., 2015).

A Figura 2 permite entender essa questão de forma mais direta. Como cada dispositivo possui uma parte independente do plano de controle, ter uma visão e controle globais da rede não é trivial (CASADO; FOSTER; GUHA, 2014). Além disso, uma alteração incorreta em um desses dispositivos pode gerar falhas em toda a estrutura, nem sempre de fácil detecção e correção (XIA et al., 2015). Em contrapartida, o paradigma SDN proporciona uma separação dos planos de dados e controle, possibilitando um controle logicamente centralizado – embora possa estar fisicamente distribuído (KREUTZ et al., 2015). Os dispositivos de rede passam a ser essencialmente encaminhadores de pacotes.

Outro ponto chave do paradigma SDN é garantir a habilidade de programar a rede (JARRAYA; MADI; DEBBABI, 2014). Além de proporcionar o controle logicamente centralizado, é importante que sejam definidos mecanismos que permitam a manipulação da rede de maneira direta e eficiente, fazendo com que aplicações e serviços tratem-na como uma entidade única (ONF, 2012).

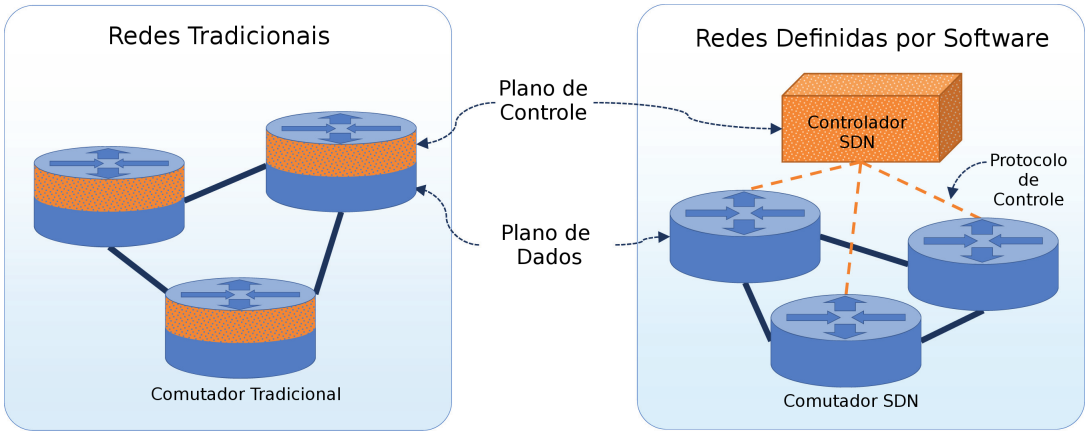

Figura 2 – Comparação entre as redes tradicionais e o paradigma SDN em termos de integração dos planos de dados e controle.

Fonte: Adaptado de Son e Buyya (2018, p. 59:5).

Portanto, nesse novo paradigma, faz-se necessário que o plano de dados forneça interface padronizada ao plano de controle, e este ao plano de gerenciamento. Essas interfaces são conhecidas, respectivamente, como *southbound interface* (SBI) e *northbound interface* (NBI), conforme apresentado na Figura 3. Elas desempenham papéis fundamentais na arquitetura SDN, pois, além de permitirem a integração entre os planos, proporcionam a interoperabilidade entre equipamentos e softwares heterogêneos. A comunicação entre os planos é implementada por meio de interfaces de programação de aplicativos (*Application Programming Interface* – API) (KREUTZ et al., 2015).

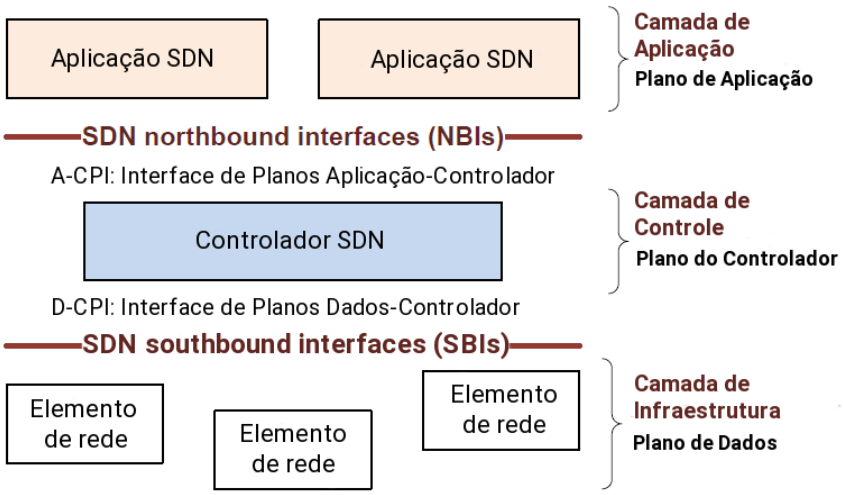

Figura 3 – Componentes SDN básicos.

Fonte: Adaptado de ONF (2014, p. 13).

No plano de controle, destaca-se ainda o controlador SDN[3]. De fato, segundo Xia et al. (2015, p. 34), "o controlador é o componente mais importante na arquitetura SDN, onde a complexidade reside". Ele é responsável por controlar o estado da rede, topologia e configuração de dispositivos (implementa as políticas de rede), provendo abstrações, serviços básicos e APIs aos desenvolvedores (KREUTZ et al., 2015). Consequentemente, também viabiliza a inovação e desenvolvimento de novos protocolos e aplicações de rede, pois a implantação dos mesmos passa a ser menos complexa e mais rápida.

Para possibilitar a padronização e impulsionar o amplo uso do paradigma SDN pelos fornecedores de dispositivos e tecnologias de redes, foi criada a *Open Networking Foundation* – ONF (ONF, 2019). De acordo com ONF (2012, p. 4), a "ONF é um consórcio da indústria, sem fins lucrativos, que está liderando o avanço do SDN e a padronização de elementos críticos da arquitetura SDN, tais como o protocolo OpenFlow™". A próxima subseção discorre um pouco sobre esse protocolo.

2.1.1 O Protocolo OpenFlow

O OpenFlow foi originalmente proposto por McKeown et al. (2008), em um projeto desenvolvido na Universidade de Standford, nos Estados Unidos, com o intuito de possibilitar que pesquisadores executassem protocolos experimentais na rede da universidade. A ideia básica era utilizar roteamento por fluxo em vez de roteamento baseado em endereços de destino. Um fluxo pode ser definido como um conjunto de valores de campos dos pacotes que correspondem a um determinado critério e para os quais há algumas ações definidas (KREUTZ et al., 2015).

Essa abordagem de roteamento por fluxo proporciona grande flexibilidade, apenas limitada pelas capacidades de implementação das tabelas de fluxo (MCKEOWN et al., 2008). A abordagem de roteamento tradicional, baseada em endereços de origem e destino, não proporciona a mesma granularidade e capacidade de adaptação (programação) às necessidades dos usuários e aplicações. Ela se mostrou eficiente quando as aplicações cliente-servidor e, consequentemente, o tráfego nesse sentido eram dominantes. Atualmente, novos padrões (leste-oeste), volume (*big data*) e dinâmica (tempo de reposta a alterações) do tráfego, proveniente das aplicações distribuídas e da computação em nuvem (local e externa), exigem mais flexibilidade e capacidade de automação (ONF, 2012).

Além das regras e ações que definem um fluxo, os dispositivos de encaminhamento OpenFlow mantêm estatísticas sobre os dados trocados em cada um deles. Como mostra a Figura 4, um dispositivo de encaminhamento OpenFlow realiza o roteamento por meio de um conjunto de tabelas de fluxos, onde cada entrada na tabela tem uma regra, uma ação e contadores (KREUTZ et al., 2015). Dessa forma, cada linha representa um fluxo.

[3] O controlador também é conhecido como sistema operacional de rede (*Networking Operating System* – NOS)(KREUTZ et al., 2015).

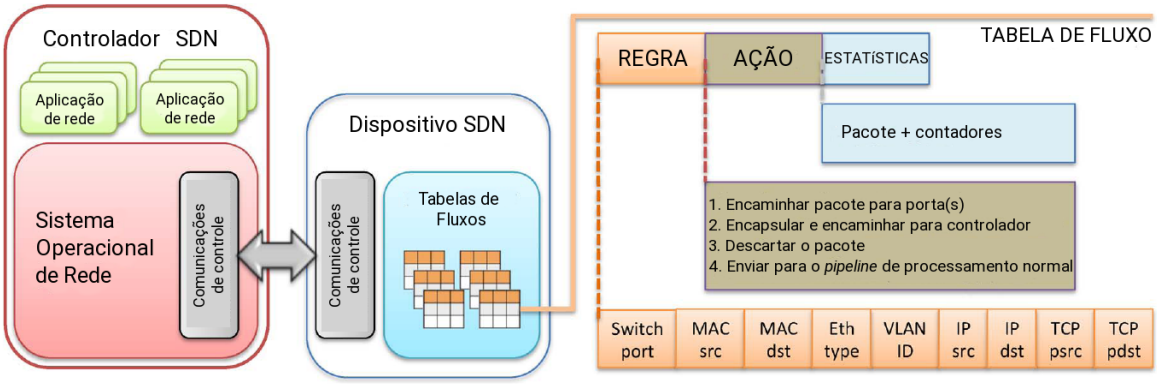

Figura 4 – Dispositivos SDN habilitados para OpenFlow.

Fonte: Adaptado de Kreutz et al. (2015, p. 9).

As tabelas são percorridas linha a linha e, assim que uma regra apropriada é encontrada numa delas, o pacote é encaminhado conforme a ação especificada para o fluxo em questão.

A Tabela 1 apresenta os contadores disponibilizados pela especificação do protocolo. Esses dados são importantes para questões de gerenciamento e controle. Neste trabalho, por exemplo, eles foram utilizados para extração de conhecimento da rede, conforme metodologia apresentada em Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017). A escolha desses contadores foi norteada pelos resultados obtidos nesses estudos e compreendem pacotes e bytes recebidos e transmitidos por porta, os quais foram obtidos através de uma extensão ao controlador OpenFlow proposta por Rezende (2016).

É importante observar que o paradigma SDN não está restrito ao padrão OpenFlow, havendo outras implementações para as interfaces entre controladores e dispositivos de encaminhamento. Revisões amplas sobre o paradigma podem ser encontradas em Farhady, Lee e Nakao (2015), Kreutz et al. (2015), Masoudi e Ghaffari (2016) e Xia et al. (2015). Nelas, é possível entender que o padrão SDN é resultado de um conjunto de conceitos e iniciativas pesquisadas ao longo dos anos na tentativa de tornar as redes mais adaptáveis e gerenciáveis. A arquitetura proposta pela ONF está descrita em ONF (2016) e a especificação para os comutadores OpenFlow em ONF (2015).

Finalmente, em síntese, pode-se afirmar que o SDN está baseado em três princípios fundamentais: (i) plano de dados e controle desacoplados; (ii) controle centralizado; e (iii) infraestrutura da rede abstraída para aplicações (capacidade de programar a rede) (ONF, 2014). Também encontra-se na literatura o roteamento por fluxo sendo citado como um quarto pilar (KREUTZ et al., 2015). Destaca-se ainda o fato de o paradigma SDN considerar a interoperabilidade com os sistemas legados, ou seja, os dispositivos tradicionais de rede, possibilitando uma transição menos traumática e mais gradual entre as tecnologias. Para uma análise sobre a evolução e implementações de redes SDN comerciais e de

Tabela 1 – Lista de contadores OpenFlow.

Contador		**Bits**	**Requerido**
Por tabela de fluxo	Contagem de referência (entradas ativas)	32	Sim
	Pesquisa de pacotes	64	Não
	Correspondências de pacotes	64	Não
Por entrada de fluxo	Pacotes recebidos	64	Não
	Bytes recebidos	64	Não
	Duração (segundos)	32	Sim
	Duração (nanosegundos)	32	Não
Por porta	Pacotes recebidos	64	Sim
	Pacotes transmitidos	64	Sim
	Bytes recebidos	64	Não
	Bytes transmitidos	64	Não
	Descartes no recebimento	64	Não
	Descartes na transmissão	64	Não
	Erros no recebimento	64	Não
	Erros na transmissão	64	Não
	Erros de alinhamento de quadros no recebimento	64	Não
	Erros de *overrun* no recebimento	64	Não
	Erros de CRC no recebimento	64	Não
	Colisões	64	Não
	Duração (segundos)	32	Sim
	Duração (nanosegundos)	32	Não
Por fila	Pacotes transmitidos	64	Sim
	Bytes transmitidos	64	Não
	Erros de *overrun* na transmissão	64	Não
	Duração (segundos)	32	Sim
	Duração (nanosegundos)	32	Não
Por grupo	Contagem de referência (entradas de fluxos)	32	Não
	Contagem de pacotes	64	Não
	Contagem de bytes	64	Não
	Duração (segundos)	32	Sim
	Duração (nanosegundos)	32	Não
Por *group bucket*	Contagem de pacotes	64	Não
	Contagem de bytes	64	Não
Por medição (*meter*)	Contagem de fluxo	32	Não
	Contagem de pacotes de entrada	64	Não
	Contagem de bytes de entrada	64	Não
	Duração (segundos)	32	Sim
	Duração (nanosegundos)	32	Não
Por banda de medição	Contagem de pacotes (*In Band*)	64	Não
	Contagem de bytes (*In Band*)	64	Não

Fonte: Adaptado de ONF (2015, p. 25).

pesquisa, sugere-se consultar SDNCentral (2016).

2.2 O Padrão DASH

O padrão internacional DASH (ISO, 2014), também conhecido como MPEG-DASH, foi desenvolvido pelo *Motion Picture Experts Group* (MPEG) em conjunto com o *3rd*

Generation Partnership Project (3GPP), em 2012, com intuito de possibilitar a interoperabilidade entre aplicações de transmissão de vídeo com taxa de bits dinamicamente adaptável sobre HTTP (*Adaptive Bitrate Streaming* – ABR – *over* HTTP)[4] (BENTALEB; BEGEN; ZIMMERMANN, 2016). Antes dele, existiam apenas iniciativas proprietárias que não interagiam entre si, dentre as quais pode-se citar: *HTTP Live Streaming* (HLS) da Apple (APPLE, 2019); *Microsoft Smooth Streaming* (MSS), da Microsoft (MICROSOFT, 2019); e *Adobe HTTP Dynamic Streaming* (HDS), da Adobe (ADOBE, 2019).

O uso do protocolo HTTP para tráfego de vídeo ocorreu de forma natural em meio a dificuldade de transpor dispositivos de segurança (*firewalls)*, traduções de endereçamento (NAT) e outros *middleboxes* via protocolos específicos para aplicações multimídia (como o RTP e o RTSP) (KUA; ARMITAGE; BRANCH, 2017). Além disso, com a popularização das CDNs[5], percebeu-se que utilizar as estruturas para tráfego HTTP tradicional por elas disponibilizadas também para o tráfego multimídia reduziria custos e permitiria melhor desempenho, uma vez que o conteúdo dessas aplicações ficaria disponível mais próximo do usuário (STOCKHAMMER, 2011). Outra vantagem que a transmissão via HTTP oferece é o fato do controle da sessão ficar a cargo do cliente, não sendo necessário manter o estado no servidor (SODAGAR, 2011).

A necessidade de adaptabilidade durante a execução do conteúdo tornou-se crucial com a computação ubíqua e os avanços nas taxas de transmissão de dados, nos métodos de codificação de vídeo e na capacidade de processamento e armazenamento dos dispositivos. Nas transmissões de vídeo sob demanda tradicionais, não eram consideradas as variações existentes entre os dispositivos no que se refere a resolução, *bufferização* e taxa de bits (BOYD, 2016). Quando muito, eram disponibilizadas algumas versões do conteúdo de acordo com os dispositivos, porém, uma vez escolhida a versão do vídeo, esta era executada até o final, sem considerar as possíveis variações de recursos disponíveis ao longo do tempo.

O objetivo principal da transmissão de vídeo ABR é possibilitar uma melhor experiência para os usuários das aplicações multimídia, independente dos dispositivos escolhidos para visualização do conteúdo, largura de banda ou processamento disponíveis (KUA; ARMITAGE; BRANCH, 2017). Para isso, essa técnica adapta dinamicamente a versão do vídeo, conforme o conteúdo vai sendo executado, de modo a fornecer a melhor qualidade aplicável para cada usuário num dado intervalo de tempo (BOYD, 2016). Como resultado, o usuário consegue assistir o conteúdo com menor número de interrupções de vídeo (SEUFERT et al., 2015).

O padrão DASH é especificado como um modelo cliente-servidor, onde diversas versões do conteúdo ficam disponibilizadas no servidor para escolha do cliente (SODAGAR, 2011). O algoritmo de seleção da versão a ser exibida em um dado instante não é especificado no padrão, sendo implementado em cada visualizador de conteúdo (*player*) da maneira

[4] Por simplicidade, ABR é usado neste trabalho como sinônimo de ABR sobre HTTP, o qual também é conhecido como *HTTP Adaptive Streaming* (HAS).

[5] *Content Delivery Network* – rede de distribuição de conteúdo (WIKIMEDIA, 2019a).

que melhor lhe aprouver (KUA; ARMITAGE; BRANCH, 2017). Para isso, a especificação estabelece que o servidor deve disponibilizar também um arquivo de manifesto, onde estão contidas todas as informações necessárias ao cliente para a seleção e execução do conteúdo. O arquivo de manifesto é descrito na Seção 2.2.1. Na Seção 2.2.2, é apresentado um outro conceito importante para a codificação dos vídeos no formato apropriado às transições entre suas versões, o chamado grupo de imagens.

Outras funcionalidades previstas no padrão DASH incluem permitir que o cliente obtenha informações sobre acessibilidade, direitos autorais, localização diversificada do conteúdo, manifesto fragmentado, dentre outras. Uma característica importante é que ele não exige que o cliente tenha suporte a todas as versões de codificação disponíveis no servidor. Como a execução do conteúdo fica a cargo do cliente, ele pode ignorar as versões que não consiga executar.

Vale destacar que são descritas, na especificação do DASH, métricas a serem coletadas pelo cliente que poderiam ser usadas pelo servidor para melhorar a qualidade de serviço. Entretanto, não é definido como isso deve ser coletado ou reportado, deixando a cargo de padronização externa. De qualquer forma, o serviço fica dependente do cliente implementar o repasse de informações ao servidor. Soluções mais apropriadas devem compreender o desenvolvimento de novos métodos para obtenção de dados de QoS sem participação do cliente, como a proposta investigada neste trabalho.

2.2.1 O Arquivo de Manifesto

O arquivo de manifesto do padrão DASH é chamado Descrição de Apresentação de Mídia (*Media Presentation Description* – MPD) (ISO, 2014). O arquivo MPD corresponde a um arquivo XML, o qual possui uma estrutura hierárquica que possibilita a escolha apropriada do trecho de vídeo a ser executado pelo cliente num dado momento. Conforme pode ser observado na Figura 5, essa hierarquia é composta por: (i) período; (ii) conjunto de adaptação; (iii) representação; e (iv) segmento.

No padrão DASH, o conteúdo a ser executado é pensado como um conjunto de períodos consecutivos, onde cada período corresponde a um intervalo de tempo no qual o conteúdo usualmente é executado sem alterações de qualidade de vídeo, áudio, idioma, legenda, etc. Os períodos podem ser estáticos ou dinâmicos no que se refere a duração. Dessa forma, no MPD, o período deve conter todas as informações que o cliente necessita para a execução apropriada daquele intervalo. Essa divisão em períodos, ou seja, unidades de tempo de execução, também foi desenvolvida para permitir a inclusão de propaganda e outras interrupções de conteúdo programadas (STOCKHAMMER, 2011).

Os períodos são compostos por um ou mais conjuntos de adaptação. Cada conjunto de adaptação é responsável por fornecer as informações sobre as alternativas disponíveis para um ou mais componentes da mídia. Um componente de mídia pode ser o vídeo, o áudio num determinado idioma, legenda, dentre outros. Cada alternativa disponibilizada

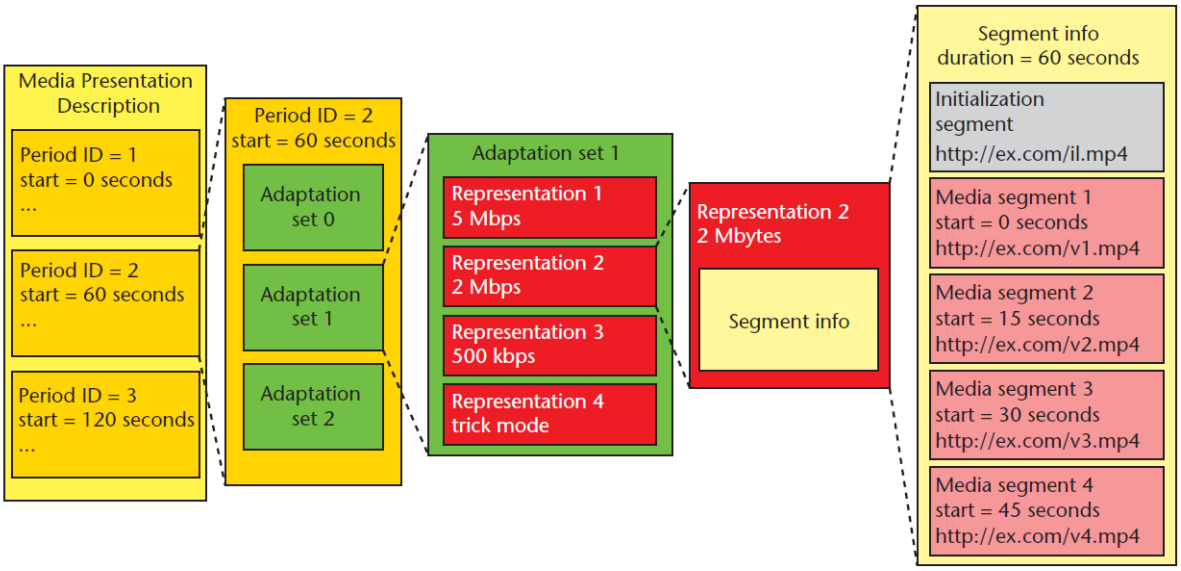

Figura 5 – Modelo de dados hierárquico do arquivo MPD.

Fonte: Sodagar (2011, p. 65).

é considerada uma representação para o(s) componente(s). Por exemplo, o vídeo pode ser disponibilizado por meio de um conjunto de adaptação que possui algumas representações variando em termos de resolução, taxa de bits e/ou quadros por segundo.

Finalmente, cada representação possui os dados sobre os segmentos que compõem o intervalo de tempo ao qual o período se refere. O segmento é a unidade básica para execução de vídeo dinamicamente adaptável. Assim, cada segmento pode ser acessado pelo cliente via um endereço único no servidor (URI) através de conexões HTTP GET ou HTTP GET com intervalo de bytes (SODAGAR, 2011). Por isso, o primeiro segmento da representação deve ser um segmento de inicialização, seguido de segmentos de mídia.

Para que o cliente consiga obter os segmentos apropriados de cada componente do conteúdo para a correta execução do mesmo num dado instante, este conteúdo deve ser disponibilizado de forma apropriada. A especificação do DASH é independente de codificação, tratando apenas de como o conteúdo deve estar estruturado, não impondo qualquer codificação de conteúdo. Existem ferramentas que permitem codificar os componentes de forma adequada, como o FFmpeg (FFMPEG, 2019), e gerar o arquivo MPD correspondente e as versões segmentadas, como o MP4Box (GPAC, 2019).

2.2.2 Grupo de Imagens

Algumas codificações, como é o caso do padrão H.264/MPEG-4 AVC (empregado neste trabalho e um dos mais populares atualmente)[6], empregam o conceito de grupo

[6] O detalhamento do padrão de compressão H.264/MPEG-4 AVC foge ao escopo deste estudo, mas pode ser encontrado em Manoel (2007).

de imagens, também conhecido como GOP – do inglês *Group Of Pictures*. O conceito de GOP é extremamente importante e deve ser devidamente respeitado entre as versões do vídeo disponibilizadas no serviço DASH para permitir a correta transição entre elas (KAZAKOV, 2015).

Conforme apresentado na Figura 6, um GOP nada mais é do que um conjunto de quadros em sequência que possui toda a informação necessária para sua exibição (ALTHOS, 2009). Para permitir um maior grau de compressão, são usados três tipos de quadros (WIKIMEDIA, 2019b; WIKIMEDIA, 2019c):

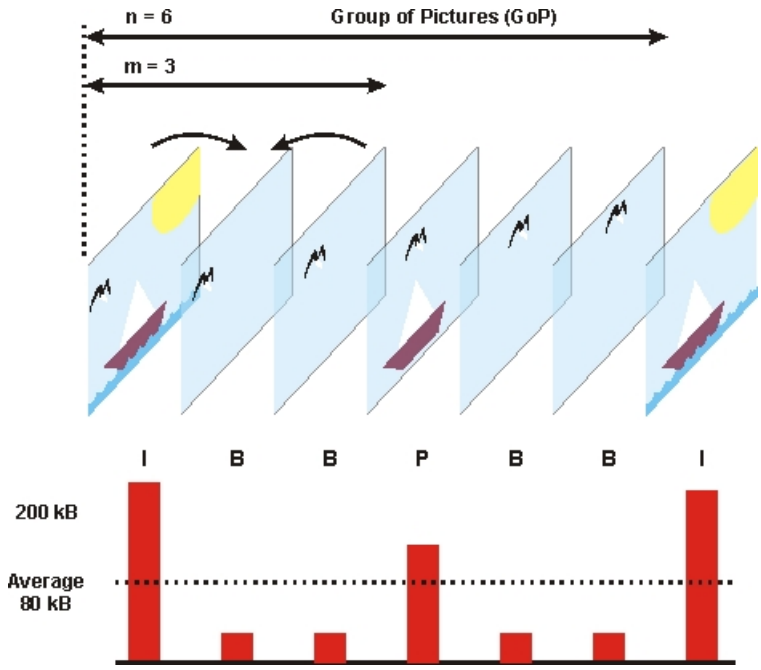

Figura 6 – Exemplo de um grupo de imagens (GOP).

Fonte: Althos (2009).

a) *I-frame* (*intra-codec picture*): é o quadro que inicia o GOP[7] e corresponde a uma codificação completa da imagem, não dependendo de nenhum outro quadro para que a mesma seja decodificada e exibida;

b) *P-frame*(*predictive coded picture*): contém apenas informações sobre diferenças compensadas por movimento em relação às imagens decodificadas anteriormente;

c) *B-frame* (*bipredictive coded picture*): semelhante aos *P-frames*, porém podem referenciar tanto dados de quadros anteriores como posteriores, permitindo uma maior compressão.

[7] Na verdade, há possibilidade de codificação com GOP que não possua *I-frame* inicial, mas referencie o último frame do último GOP (KAZAKOV, 2015). Contudo, devido à complexidade para os decodificadores, ela não é recomendada.

O tamanho do GOP é a distância entre dois *I-frames*. No exemplo da Figura 6, o GOP possui tamanho $n = 6$. O outro parâmetro necessário para definição do GOP é a distância entre os âncoras (I e *P-frames*), que no exemplo citado corresponde a $m = 3$. É possível ter vários *P-frames* entre dois *I-frames*: para um caso de GOP com $n = 12$ e $m = 3$ tem-se um GOP no formato $IBBPBBPBBPBBI$.

Como dito previamente, para o correto funcionamento do serviço DASH é importante que as diferentes versões do vídeo estejam alinhadas em relação ao GOP. Isso significa que elas devem conter o mesmo número de GOPs, de modo que os segmentos sejam iniciados por quadros *I-frame*. Porém, isso não necessariamente implica que elas não possam ser codificadas com diferentes taxas de quadros por segundo, embora seja mais usual manter essa taxa uniforme durante a codificação das mesmas.

2.3 Aprendizagem de Máquina Supervisionada

A aprendizagem de máquina constitui o estudo e a modelagem computacional do processo de aprendizagem nas suas múltiplas manifestações (MICHALSKI; CARBONELL; MITCHELL, 1983). De modo mais intuitivo, é possível entendê-la como segue:

> Aprendizagem de máquina, então, é sobre fazer com que computado-resinvestigação *modifiquem* ou *adaptem* suas ações (sejam estas ações realização de predições, ou controle de um robô), de modo que estas ações obtenham melhor acurácia, onde acurácia é medida por quão bem as ações escolhidas refletem as ações corretas (MARSLAND, 2009, p. 5).

A Figura 7 apresenta um paralelo entre a aprendizagem de máquina e o pensamento humano, do qual conclui-se que a primeira consiste em processos de indução e síntese para extração de conhecimento e não em dedução (ZHAO et al., 2019). Na aprendizagem de máquina, os dados históricos assumem o papel das experiências e a ação de induzir é realizada por meio de treinamento de um modelo, aplicando-se algoritmo específico, com base nesses dados.

Classicamente, a aprendizagem de máquina possibilita desenvolver modelos que aprendam a partir de dados prévios (características e resultados), numa abordagem supervisionada, ou que sejam capazes de inferir conhecimento a partir de atributos dos dados, numa perspectiva não supervisionada. Mas também há métodos que combinam dados de ambos os tipos, numa abordagem semissupervisionada, e outras técnicas como aprendizagem de reforço (*reinforcement learning*) e aprendizagem profunda (*deep learning*) (ZHAO et al., 2019; WANG et al., 2017).

O aprendizado supervisionado, portanto, "refere-se a qualquer processo de aprendizagem de máquina que aprende uma função de um tipo de entrada para um tipo de saída usando dados que incluem exemplos que têm valores de entrada e de saída"(SAMMUT; WEBB, 2011, p. 941). Diz-se que as amostras disponíveis para elaboração dos modelos

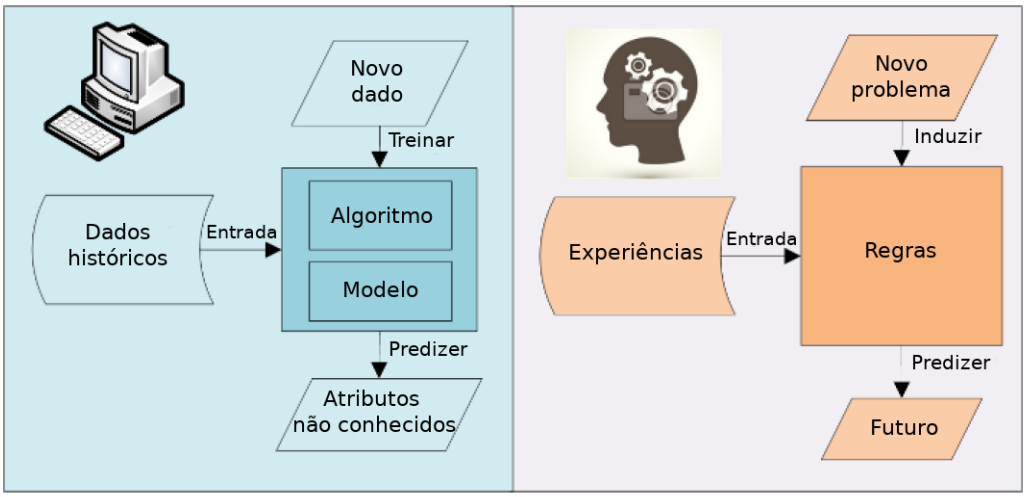

Figura 7 – Um contraste entre aprendizagem de máquina e o pensamento humano.

Fonte: Adaptado de Zhao et al. (2019, p. 95399).

nesses métodos são dados rotulados, pois possuem informações sobre o alvo (saída), enquanto que, na aprendizagem não supervisionada, os dados são considerados não rotulados (Ibid., p. 571 e 1008).

Os problemas em aprendizado supervisionado são divididos tradicionalmente em dois tipos distintos: regressão, quando as entradas são utilizadas para estimar saídas quantitativas, ou seja, valores contínuos; e classificação, quando elas são usadas para predizer saídas qualitativas, isto é, valores discretos (HASTIE; TIBSHIRANI; FRIEDMAN, 2009). Um exemplo de problema de regressão é estimar valores monetários de venda de um imóvel a partir de informações sobre área, idade da construção, dentre outros atributos, de outros imóveis já vendidos. No caso da classificação, pode-se citar o tradicional problema de classificação de novas mensagens de e-mail em *spam* ou não-*spam*, a partir de atributos de um conjunto de mensagens previamente classificadas.

Em termos de nomenclatura, os dados de entrada são muitas vezes referenciados como preditores, atributos, características, variáveis dependentes ou apenas variáveis; as saídas, por sua vez, como respostas ou variáveis independentes (JAMES et al., 2013). Valores discretos de saída também são denominados categorias ou classes: um classificador possui a habilidade de classificar o dado de entrada em uma das possíveis categorias tratadas no problema em questão (HASTIE; TIBSHIRANI; FRIEDMAN, 2009; TAN; STEINBACH; KUMAR, 2009).

Formalmente, a ação de predição é modelada matematicamente como o problema de se encontrar uma função de aproximação que relacione apropriadamente as variáveis de entrada com as saídas correspondentes, permitindo obter as saídas estimadas para novos valores de entrada (JAMES et al., 2013). Em geral, o conjunto de variáveis de entrada

é representado como um vetor (ou matriz) denominado X, as saídas referenciadas como o conjunto Y e os valores de saída preditos por $\hat{Y}$. Há autores que diferenciam as saídas quantitativas das qualitativas, representando as últimas por G e o conjunto de estimativas por $\hat{G}$(HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

Assim, para o problema de regressão, sejam $X = (X_1, X_2, ..., X_n)$ e Y, respectivamente, os conjuntos de variáveis de entrada e de respostas. Então, tem-se idealmente que:

$$Y = f(X) + \epsilon, \tag{1}$$

onde $f(X)$ é uma função que mapeia o relacionamento entre X e Y, com imagem em $\mathbf{R}$, e ϵ é um termo de erro aleatório, independente de X e com média zero. A predição consiste em encontrar o conjunto $\hat{Y} = \hat{f}(X)$, de modo que $\hat{f}(X) \approx f(X)$ (JAMES et al., 2013). Para isso, várias técnicas e algoritmos de aprendizagem podem ser empregados (SAMMUT; WEBB, 2011).

O desenvolvimento de modelos de aprendizagem de máquina costuma envolver três fases: treinamento, onde o modelo é ajustado; validação, onde o erro de predição é estimado para seleção do modelo; e teste, para avaliação do erro de generalização do modelo final escolhido. Neste caso, as amostras são divididas aleatoriamente em conjunto de treinamento, de validação e de teste. O percentual costuma ser de 50, 25 e 25 por cento respectivamente para cada um deles (HASTIE; TIBSHIRANI; FRIEDMAN, 2009, p. 222).

Porém, é comum encontrar na literatura a fase de validação incorporada a fase de treinamento, principalmente em trabalhos aplicados. Quando isso acontece, o modelo é desenvolvido com fases de treinamento e teste, sendo que o percentual de divisão do conjunto de amostras disponíveis fica em torno de 70 por cento para o treinamento e 30 por cento para os testes (PASQUINI; STADLER, 2017).

Tão fundamentais quanto as técnicas para encontrar uma predição $\hat{Y}$ satisfatória são as métricas de avaliação de desempenho dos modelos gerados. Uma dessas métricas é o erro absoluto médio normalizado (*Normalized Mean Absolute Error* - NMAE), que corresponde a média normalizada dos erros absolutos das predições efetuadas, sendo calculado como segue:

$$NMAE = \frac{1}{\bar{y}} \left(\frac{1}{m} \sum_{i=1}^{m} |y_i - \hat{y}_i| \right), \tag{2}$$

onde y_i corresponde à resposta da amostra i do conjunto de teste; $\hat{y}_i$ refere-se ao valor estimado para a amostra i do conjunto de teste; $\bar{y}$ é a média das respostas das amostras do conjunto de teste; e m é o tamanho do conjunto de testes (PASQUINI; STADLER, 2017).

Além dessa métrica, a qual é uma forma de calcular a acurácia do modelo, é essencial analisar a sua viabilidade, isto é, se as predições podem ser executadas em tempo hábil

ou se o custo computacional do modelo o torna inaplicável ao problema em questão. Comumente, o tempo que mais influencia é o tempo necessário para treinar (e validar) o modelo, fase na qual, em geral, ocorre a aprendizagem e que trata um maior número de amostras. Portanto, considerando a divisão em fase de treinamento e teste, a métrica habitualmente usada para avaliar a viabilidade é o tempo de treinamento.

Os algoritmos de aprendizagem de máquina supervisionada empregados neste trabalho são o de árvore de regressão e de floresta aleatória. Optou-se por manter os mesmos algoritmos utilizados em Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017), trabalhos que norteiam essa pesquisa, por dois motivos: eles apresentaram bom desempenho em relação a outros nos estudos iniciais (YANGGRATOKE et al., 2015); o uso dos mesmos métodos que na análise para a aplicação de vídeo sob demanda não adaptável permite uma comparação mais direta do desempenho obtido pela aplicação DASH. Por essa última razão, foram mantidos os mesmos métodos de redução de atributos. Esses algoritmos são descritos a seguir.

2.3.1 Árvores de Regressão

Trata-se de uma árvore de decisão aplicada ao problema de regressão (JAMES et al., 2013). Compreende duas etapas:

1. Dividir o espaço X de n características, ou seja, o conjunto de possíveis valores para X_1, X_2,..., X_n, em J regiões distintas e não sobrepostas, R_1, R_2,...,R_J.

2. Para cada observação que cair numa dada região R_j, realizar a mesma predição, a qual consiste na média de valores de resposta para as observações de treino em R_j .

O objetivo é encontrar regiões R_1,...,R_j que minimizem a soma dos quadrados residuais (*Residual Sum of Square* – RSS) dada por:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \quad onde\, \hat{y}_{R_j} = \sum_{i \in R_j} \frac{y_i}{|R_j|}. \tag{3}$$

Ou seja, $\hat{y}_{R_j}$ é a resposta média para as $|R_j|$ observações de treinamento da j-ésima região.

Devido a custos computacionais, entretanto, o cálculo das regiões é feito por meio de uma divisão binária recursiva, empregando um algoritmo guloso, numa abordagem *top-down*. A cada passo, escolhe-se a melhor divisão (atributo e valor limite) considerando apenas esta etapa particular, partindo-se do topo da árvore para as folhas. Isso resulta num algoritmo com complexidade $O(m^2n)$, onde m é o número de amostras e n o número de atributos (PASQUINI; STADLER, 2017). Para evitar o *overfitting*, ou seja, a incorporação de ruídos ou variação dos dados de amostragem no modelo de modo a não refletir a distribuição apropriada para os dados do problema (SAMMUT; WEBB, 2011), pode-se

ainda construir uma árvore inicialmente maior e realizar a poda da mesma (JAMES et al., 2013).

2.3.2 Florestas Aleatórias

O algoritmo floresta aleatória é um método de agregação (*ensemble*) que se utiliza de um conjunto de árvores de regressão (ou de decisão, no caso de mapeamento em valores discretos) para obter uma melhor estimativa do valor resposta. Ele foi proposto em (BREIMAN, 2001) e adiciona uma camada de aleatoriedade ao algoritmo *Bagging*[8], mudando a forma como as árvores de regressão são construídas e evitando a correlação entre as mesmas.

Ambos algoritmos empregam a técnica de *bootstrap*[9] para elaboração de novos conjuntos de instâncias que servirão de entrada na construção das árvores. Porém, tradicionalmente, no *Bagging*, todos os atributos participam a cada divisão na construção de uma dada árvore. Em contrapartida, no algoritmo floresta aleatória, apenas um subconjunto deles é utilizado.

As etapas do método de floresta aleatória podem ser descritas como segue (LIAW; WIENER, 2002):

1. Utilizando a técnica de *bootstrap*, crie T conjuntos de amostras de tamanho n a partir dos dados originais;

2. Para cada conjunto de amostras criado, construa uma árvore não podada, com a seguinte modificação:

   A cada nó, em vez de escolher a melhor divisão entre todos os n atributos, aleatoriamente amostre p atributos, com $p \leq n$, e escolha a melhor divisão entre estas variáveis;

3. Estime dados novos agregando as predições. Para a regressão, isso significa, em geral, calcular a média dos resultados obtidos em cada uma das árvores.

T e p são passados como parâmetro e correspondem, respectivamente, ao número de árvores geradas e o número de atributos utilizados em cada divisão (constante). Dessa forma, se $p = n$, onde n é o número total de atributos, tem-se o algoritmo *Bagging*. Uma escolha recomendada para p é utilizar $p \approx \sqrt{n}$ (JAMES et al., 2013). O algoritmo floresta aleatória é da ordem de complexidade $O(Tm^2n)$, onde m é o número de amostras

[8] *Bagging*, um acrônimo para *Bootstrap AGGregatING*, é um método de agregação no qual cada membro é construído com um conjunto diferente de treinamento, os quais correspondem a uma amostra do conjunto original, sendo os modelos combinados ao final por uma média uniforme ou votação (SAMMUT; WEBB, 2011, p. 73).

[9] Amostragem *bootstrap* é um processo para criação de uma distribuição de conjuntos de dados a partir de um único conjunto de dados (Ibid., p. 137).

(PASQUINI; STADLER, 2017). Ele apresenta também a vantagem de ser robusto em relação ao *overfitting* (JAMES et al., 2013).

2.3.3 Métodos de Redução de Atributos

Os métodos de redução de atributos são empregados para encontrar, de maneira automatizada, um conjunto de atributos que maximize a métrica de avaliação das estimativas realizadas. Isso porque algum dos atributos pode ser desnecessário ou até mesmo prejudicar a predição. Descartar atributos desnecessários no modelo é importante, pois reduz a dimensionalidade do conjunto de atributos, o que usualmente diminui o tempo de processamento para obtenção dos resultados.

Encontrar um conjunto ótimo de atributos pode ser uma tarefa inviável, pois requer a análise de 2^n subconjuntos de tamanho n (STADLER; PASQUINI; FODOR, 2017). Portanto, os métodos de redução de atributos são heurísticas usadas para se chegar a um conjunto de atributos aceitável, dentro de um período de tempo que não comprometa o tempo de processamento total do modelo.

Dois desses métodos de redução de atributos foram utilizados no trabalhos que propuseram a metodologia empregada nesta pesquisa (PASQUINI; STADLER, 2017; STADLER; PASQUINI; FODOR, 2017). São eles (JAMES et al., 2013):

a) Seleção passo a frente (*forward-stepwise-selection*): parte de um conjunto vazio de atributos e, a cada iteração, inclui um novo atributo que minimize o erro estimado. Quando não houver mais atributos que reduzam este erro, o processo é finalizado;

b) Seleção univariada (*univariate-feature-selection*): consiste em calcular, para cada atributo, a correlação cruzada entre o regressor e o alvo, ordenar os atributos de acordo com os valores de avaliação e selecionar os k atributos melhor posicionados para obter o conjunto de atributos reduzidos. Exige que sejam avaliados n subconjuntos, cada um contendo um único atributo.

2.4 Trabalhos Correlatos

Os trabalhos que mais se relacionam com a proposta apresentada neste documento são os propostos por Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017), que serviram de base para metodologia e comparação. A diferença está na aplicação aqui analisada, a qual tem ganhado popularidade, sendo responsável por grande volume do tráfego dentre as aplicações multimídia para Internet. Além disso, essa aplicação possui particularidades até então não investigadas por meio da metodologia em questão, as quais precisavam ser analisadas para melhor avaliar o desempenho e a aplicabilidade desta metodologia.

Segundo os autores desses trabalhos, por sua vez, o estudo que mais se aproxima do por eles desenvolvido é o apresentado em (HANDURUKANDE et al., 2011). Nele é descrita

uma solução, denominada Magneto, para estimar métricas de QoS numa aplicação de *streaming* de IPTV. As estimativas são feitas considerando estatísticas em nível de rede, como atraso, perda e variação do atraso (*jitter*). Apesar de os responsáveis pela proposta Magneto afirmarem que o método possui precisão satisfatória ao estimar as métricas, ela possui os inconvenientes de gerar tráfego extra na rede para obter as informações necessárias para as estimativas e de o método ser sensível à perda de pacotes.

Além do presente trabalho, a metodologia em estudo foi empregada em duas outras pesquisas. Em Samani e Stadler (2018), é analisado o uso de redes neurais para predizer a distribuição condicional de métricas de serviços para as mesmas aplicações originalmente estudadas. Os resultados encontrados foram tão satisfatórios quanto os obtidos utilizando o método de florestas aleatórias, considerado como uma referência. No segundo trabalho, Moradi, Stadler e Johnsson (2019), a metodologia foi expandida para investigar o retreinamento de modelos de redes neurais por meio de transferência de aprendizagem como forma de se adaptar a mudanças no ambiente e garantir que o modelo continue satisfatório. Os resultados indicaram que o método proposto significativamente reduz o número de novas medições para computar um novo modelo após a alteração.

De modo mais amplo, pode-se considerar que o presente trabalho integra quatro temas centrais: qualidade de serviço; redes SDN; aplicações HAS; e aprendizagem de máquina. Nos últimos anos, vários trabalhos trataram da qualidade da experiência (*Quality of Experience* – QoE) do usuário nas aplicações de vídeo dinamicamente adaptável. Em Seufert et al. (2015), é possível encontrar um revisão sobre essas pesquisas, bem como uma análise sobre os fatores que influenciam na QoE e desafios ainda em aberto. Embora QoE e QoS sejam conceitos distintos, é possível encontrar na literatura trabalhos que estudam a correlação entre eles, permitindo inferir QoE a partir de dados de QoS, inclusive com o uso de aprendizagem de máquina (AROUSSI; MELLOUK, 2016; KATSARAKIS et al., 2016; MUSHTAQ; AUGUSTIN; MELLOUK, 2012).

O foco de muitos desses estudos sobre QoE em aplicações HAS é no modo como o cliente seleciona a versão a ser exibida, ou seja, na estratégia de adaptação do vídeo (KUA; ARMITAGE; BRANCH, 2017). Algumas delas utilizam fatores de QoE no desenvolvimento do algoritmo (COELHO; MELO, 2016; MORI; BANDAI, 2018), bem como aprendizagem de máquina (GADALETA et al., 2017; ABAR; LETAIFA; ELASMI, 2018). Também pode-se encontrar na literatura estudos que buscam predizer o gerenciamento de QoE usando aprendizagem de máquina (VEGA et al., 2018). Em Vasilev et al. (2018), é desenvolvido um modelo, baseado em aprendizagem de máquina, para predição de eventos de *re-buffering* a partir de métricas de QoS. A frequência e duração do tempo de *re-buffering* é uma das questões que influenciam a experiência do usuário, sendo considerada um fator de QoE. Segundo os autores, o estudo mostrou que o uso de informação sobre congestionamento de rede e características básicas dos fluxos de vídeo melhoram a predição.

Há ainda estudos direcionados à investigação desse tema em enlaces sem fio. Ji et al. (2016), por exemplo, analisa os problemas de QoE das aplicações DASH em redes LTE. São desenvolvidos no trabalho dois modelos baseados em aprendizagem de máquina para estimar métricas de QoE. O primeiro deles é desenvolvido para o usuário estimar QoE antes de iniciar a exibição do conteúdo, enquanto que o segundo tem a finalidade de permitir que o módulo de controle do serviço possa ajustar a taxa de bits do vídeo em tempo real. A contribuição do estudo foi demostrar que a característica que mais interfere na degradação do vídeo é o RTT (*Round Trip Time*).

Com relação ao uso conjunto de DASH e SDN, o mesmo é objeto de estudo dos trabalhos Bentaleb, Begen e Zimmermann (2016) e Bentaleb et al. (2017). Neles, os autores investigam como as redes SDN podem melhorar a experiência dos usuários de aplicações DASH. Para isso, propõem uma nova arquitetura para essas aplicações, primeiramente chamada SDNDASH, a qual foi aprimorada e renomeada para SDNHAS. Segundo os autores, os resultados experimentais indicaram que a arquitetura melhora a escalabilidade e a justiça na experiência do usuário (QoE *fairness*) em mais de 30%, além de melhorar o uso de recursos de rede em quase 30%. Porém, diferente do trabalho aqui proposto, compreende uma solução completa, que exige modificações nos clientes e não emprega aprendizagem de máquina, citando-a como trabalho futuro. A modificação exigida no cliente na metodologia empregada no presente trabalho é apenas para coleta de dados para definição do modelo.

Em Petrangeli et al. (2017), os pesquisadores desenvolveram um *framework* para prevenção de interrupções (*freezes*) de vídeo em aplicações HAS, com o objetivo de melhorar QoE. Um mecanismo de aprendizagem de máquina, baseado em um algoritmo de *boosting* (*Random Undersampling Boosting - RUSBoost*) e lógica *fuzzy*, instalado no controlador SDN é responsável por determinar e priorizar os pacotes que podem causar as interrupções. Apenas informações da rede influenciam nessas decisões. Segundo os autores, os resultados obtidos com o *framework* superam os de algoritmos de *benchmarking*.

Outros trabalhos também podem ser citados por investigarem os benefícios das redes SDN para as aplicações de vídeo. Em Georgopoulos et al. (2015), os autores apresentam uma forma de como as redes SDN podem ser utilizadas para promover cache de vídeo sob demanda, melhorando a distribuição para os usuários. Uma proposta de utilização de aprendizagem de reforço para determinar o instante em que as rotas dos fluxos de vídeo devem ser alteradas, bem como as taxas de bits dos vídeos, é descrita em Uzakgider, Cetinkaya e Sayit (2015). A proposta é avaliada utilizando o protocolo RTP sobre UDP para transmissão dos pacotes de vídeo.

Finalmente, cabe citar as recentes revisões de literatura apresentadas em Sanaei e Mostafavi (2019) e Zhao et al. (2019). A primeira trata das pesquisas sobre as técnicas de entrega de conteúdo multimídia em redes SDN; a segunda permite verificar o amplo número de pesquisas e aplicações em que o uso conjunto de SDN e aprendizagem de

máquina tem sido empregado.

O presente trabalho difere, entretanto, das pesquisas citadas, pois o interesse foi identificar, para a aplicação DASH, a relação entre métricas de QoS e as estatísticas de tráfego agregado dos dispositivos de rede numa arquitetura SDN, utilizando-se uma metodologia específica.

Capítulo 3

Ambiente e Metodologia

O objetivo geral do proposta aqui apresentada foi investigar como métricas de QoS em um cliente DASH se relacionam com estatísticas de tráfego de rede agregado em uma rede OpenFlow. A metodologia empregada aborda este problema como um problema de regressão em aprendizagem de máquina supervisionada: a partir de um determinado conjunto de dados rotulados, busca-se mapear o relacionamento entre as características (atributos) em análise e os valores alvos observados (rótulos).

O ambiente de teste desenvolvido para a geração da série temporal, a qual serviu de entrada para os métodos de aprendizagem de máquina, é descrito neste capítulo na Seção 3.1. Para melhor entendimento, optou-se pela apresentação dos componentes que integram o ambiente por meio de visões física (Seção 3.1.1) e lógica (Seção 3.1.2), já que algum deles são virtualizados. Em seguida, são especificados os parâmetros de codificação empregados na formatação dos vídeos disponibilizados no servidor DASH e acessados pelo cliente e pelo gerador de carga (Seção 3.2). A especificação do problema, objeto desta pesquisa, e a metodologia usada para abordá-lo são formalizadas e detalhadas na Seção 3.3.

3.1 Componentes do Ambiente de Teste

O ambiente de teste para a pesquisa aqui descrita correspondeu à implementação de um serviço de vídeo dinamicamente adaptável via HTTP, mais especificamente, utilizando-se o padrão DASH, o qual é disponibilizado para os clientes por meio de uma rede SDN, empregando-se o protocolo OpenFlow. Dessa forma, tem-se, no ambiente, duas entidades principais: a rede OpenFlow e o serviço DASH.

A rede OpenFlow é formada por controlador e três dispositivos de rede. Aqui, ressalta-se o fato desses dispositivos serem físicos, enquanto que em Pasquini e Stadler (2017) e em Stadler, Pasquini e Fodor (2017), que investigaram previamente a metodologia, empregou-se apenas comutadores virtuais. O serviço DASH é composto por servidor, cliente e gerador de carga, sendo este último responsável por simular tráfego de outros clientes.

Há ainda um servidor auxiliar, o qual fornece um serviço NTP (MILLS et al., 2010), cuja função é garantir que os relógios dos componentes onde são obtidos os atributos e rótulos estejam corretamente sincronizados.

3.1.1 Visão Física

A estrutura física do ambiente de testes pode ser observada na Figura 8 e compreende os seguintes elementos:

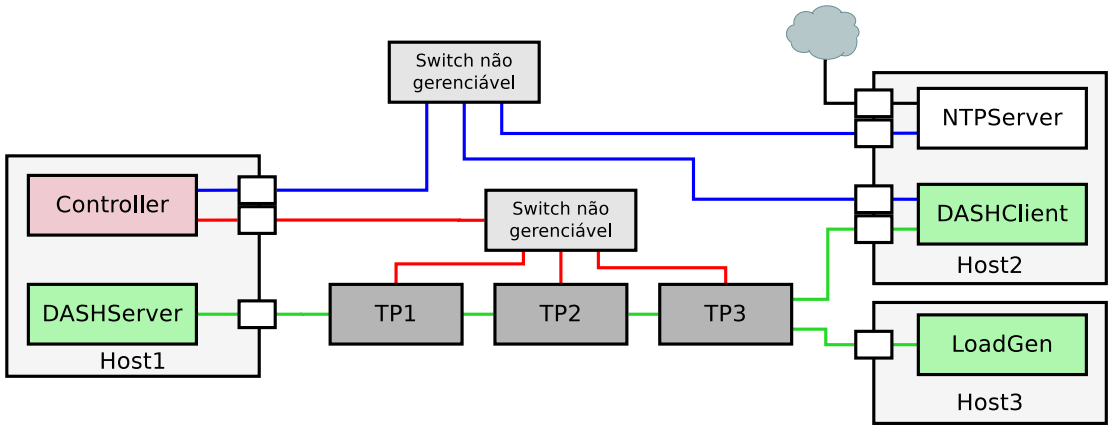

Figura 8 – Visão física dos componentes do ambiente de testes.

a) Máquinas hospedeiras: foram utilizadas três máquinas físicas para implementação virtualizada dos servidores e *desktops*. Suas configurações são descritas a seguir:
   - `Host1`: trata-se de um *notebook* Dell Inspiron™ 14 7460, com processador Intel® Core™ i7-7500U (quatro núcleos de 2.7 GHz) e 16 GB de memória RAM. O sistema operacional instalado é o Ubuntu 18.04.3 LTS em disco SSD de 128 GB, com partição de dados em disco rígido de 1 TB. Nele foram virtualizados o servidor DASH e o controlador SDN. A virtualização foi efetuada por meio do VirtualBox (ORACLE, 2020), versão 5.2.34. Além da interface de rede cabeada da máquina, foram usadas duas placas adicionais por meio de adaptadores USB para RJ-45 com velocidades de até 100 Mbps.
   - `Host2`: corresponde a um *notebook* HP ProBook 440, com processador Intel® Core™ i5-4200M (quatro núcleos de 2.5 GHz) e 8 GB de memória RAM. O sistema operacional usado é o OpenSUSE 15.1, instalado em disco rígido de 1 TB. A versão do VirtualBox empregada foi a 6.0.12, sendo nele implementadas a máquina cliente e o servidor NTP. Aqui, como no `Host1`, fez-se uso de dois adaptadores USB para RJ-45 com velocidades de até 100 Mbps para ampliar o número de interfaces de rede disponíveis. Também foi utilizada a interface sem fio da máquina conectada à Internet para permitir que o servidor NTP sincronizasse com os servidores de referência.

  - `Host3`: corresponde a um *notebook* Dell Latitude 3490, com processador Intel® Core™ i5-8250U (oito núcleos de 1.6 GHz) e 8 GB de memória RAM. O sistema operacional usado é o OpenSUSE 15.1, instalado em disco SSD de 256 GB. A versão do VirtualBox empregada foi a 6.1.2, sendo nele implementado o gerador de carga.

b) `Controller`: máquina virtual com 1 processador, 4 GB de memória RAM, disco rígido de 30 GB e duas placas de rede 10/100 Mpbs em modo *bridge* com adaptadores de rede exclusivos. Uma das placas é utilizada para conexão com a rede OpenFlow de controle dos dispositivos de rede e, a outra, destinada a comunicação com a rede do servidor NTP. O sistema operacional instalado é o Ubuntu Server 18.04 e o controlador SDN, o Floodlight (FLOODLIGHT, 2020), com suporte à versão 1.3 do padrão OpenFlow.

c) *`Switches`* `não gerenciáveis`: foram utilizados para conexão dos roteadores com o controlador e para conexão do servidor NTP com o controlador e o cliente. Correspondem a equipamentos D-Link® DES-1008-C, com 8 portas Fast-Ethernet;

d) `TP1`, `TP2` e `TP3`: três roteadores físicos idênticos, modelo TP-Link® TL-WR1043ND (TP-LINK, 2019). O *firmware* instalado em todos eles é o OpenWrt (OPENWRT, 2020) com Open vSwitch (OVS) (LINUX, 2019). A conexão com o controlador OpenFlow está configurada na interface WAN dos roteadores.

e) `DASHServer`: servidor DASH virtualizado, com 2 processadores, memória RAM de 2 GB e disco rígido de 100 GB. Possui uma interface de rede configurada a 10/100 Mbps em modo *bridge* com a interface de rede cabeada da máquina local. O sistema operacional instalado é o Ubuntu Server 18.04.3 LTS, com servidor HTTP Apache, versão 2.4.29 (APACHE, 2020).

f) `DASHClient`: trata-se de uma máquina virtual com Ubuntu 18.04.3 LTS, disco de 20 GB, memória RAM de 2 GB, e processador único. Possui duas placas de rede de 10/100 Mbps em modo *bridge* com adaptadores de rede exclusivos: uma para conexão com o serviço DASH; e a outra para conexão com o servidor NTP. O cliente DASH utilizado é o reprodutor de mídia VLC (VIDEOLAN, 2019c), versão 3.0.8, compilado com a modificação para *log* dos parâmetros de monitoramento (métricas de QoS) desenvolvida em Matos (2019).

g) `LoadGen`: máquina virtual semelhante ao cliente, com o diferencial de possuir o dobro de memória e de processamento e apenas uma interface de rede, configurada a 10/100 Mbps em modo *bridge* com a interface cabeada da máquina hospedeira. A versão do VLC, apesar de ser a mesma, é, entretanto, a padrão, sem qualquer modificação para *log* de dados. Nesta máquina estão os *scripts* que geram instâncias do VLC para inclusão de tráfego semelhante na rede, por meio de requisições adicionais ao servidor DASH.

h) `NTPServer`: servidor virtual com sistema operacional Ubuntu Server 18.04.3 LTS, 1 GB de memória RAM, um processador, disco rígido de 40 GB e duas interfaces de rede. Uma das interfaces funciona em modo *bridge* com a interface cabeada *on board* do hospedeiro; a outra é configurada como NAT, possibilitando a conexão com a Internet via conexão sem fio do hospedeiro. Essa conexão é necessária para sincronização com servidores NTP de referência. O servidor NTP é implementado por meio do pacote `ntp`, disponibilizado por padrão nos repositórios do Ubuntu (CANONICAL, 2019).

3.1.2 Visão Lógica

A Figura 9 exibe a visão lógica do ambiente de testes. Por uma questão de simplicidade, foi omitida a conexão do servidor NTP com a Internet, pois é suficiente saber que o `NTPServer` fornece horário idêntico aos componentes nos quais ocorrem a geração e coleta de dados que constituem a série temporal. Conforme será detalhado adiante, esses componentes compreendem o controlador SDN, os *switches* OpenFlow e o cliente DASH. Foram instalados clientes NTP configurados para sincronizar unicamente com esse servidor apenas no controlador e no cliente, sendo que os roteadores têm seus relógios sincronizados com o controlador.

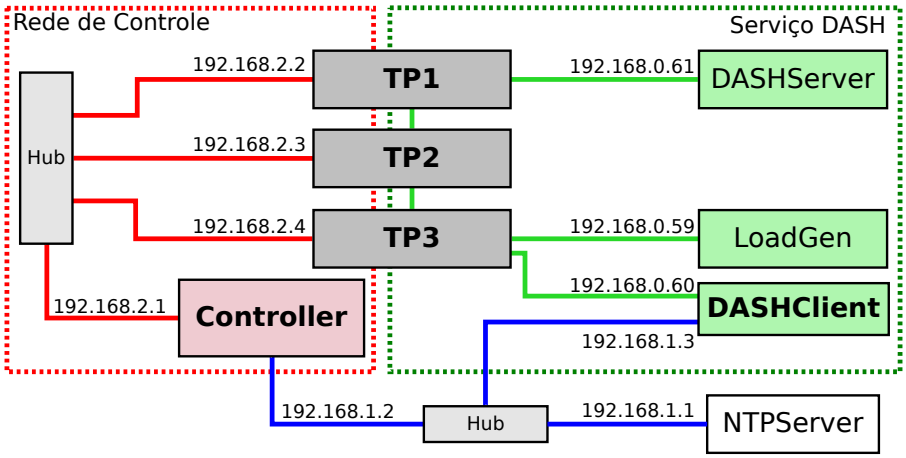

Figura 9 – Visão lógica dos componentes do ambiente de testes.

A rede OpenFlow que interliga os clientes ao servidor DASH é formada por três nós físicos, conectados diretamente entre si de forma sequencial: *switch* de acesso ao servidor (borda) – `TP1`; *switch* de núcleo – `TP2`; *switch* de acesso aos clientes (borda) – `TP3`. O `Controller` gerencia esses equipamentos por meio de uma rede de controle exclusiva, a qual opera na faixa de endereçamento `192.168.2.0/24`.

No `DASHServer`, estão armazenados os vídeos, as *playlists*, e há um servidor Web instalado para permitir o acesso a esses conteúdos. Cada um dos vídeos, na verdade,

corresponde a seis arquivos distintos: um arquivo MPD, dois arquivos com versões diferentes do componente de áudio e três arquivos com versões diferentes do componente de vídeo, todos eles em formato compatível com o padrão DASH. Os detalhes sobre os vídeos, ferramentas e parâmetros de codificação utilizados são apresentados mais adiante, na Seção 3.2. Por razões de performance e espaço de armazenamento, os vídeos não foram codificados para o padrão DASH no servidor, mas na máquina hospedeira (`Host1`).

O `LoadGen` é responsável por emular clientes VLC, provendo requisições para o serviço DASH conforme um determinado padrão. O objetivo é produzir tráfego concorrente para simular, de forma aproximada, as oscilações de carga existentes num ambiente real. O padrão analisado neste trabalho é chamado de *periodic load* ou padrão de carga periódica (PASQUINI; STADLER, 2017). Ele segue um processo Poisson cuja taxa de chegada é modulada por uma função senoidal. Possui como parâmetros: o valor inicial P_S; a amplitude P_A; e o período P_T da função senoidal.

A aplicação geradora de carga usada para implementar esse padrão consiste em um *script Python* que instancia e encerra clientes VLC seguindo uma dada distribuição durante um determinado período de tempo – determinada pelos parâmetros citados anteriormente. Caso um cliente em particular fique instanciado por mais tempo que a duração do vídeo em exibição (selecionado aleatoriamente em uma *playlist* com os vídeos disponíveis), ele passa a repetir o vídeo em questão até ser encerrado[1]. Por limitações de desempenho da máquina virtual (e hospedeira), os vídeos são executados em janelas com tamanho correspondente a 15% do padrão de abertura.

A Figura 10 reproduz o comando executado pela aplicação geradora de carga para criação de instâncias do VLC. As informações sobre os parâmetros foram extraídas de VideoLAN (2019b). Porém, não foi possível encontrar na documentação detalhes sobre as opções de adaptação implementadas que não fossem os códigos-fonte (VIDEOLAN, 2019a). Como esta é uma informação secundária para a pesquisa, optou-se pela lógica de adaptação `predictive` sem que houvesse prejuízo para a coleta de dados.

Os componentes envolvidos no monitoramento e coleta de dados estão indicados na Figura 9 com o nome em negrito. São eles: `Controller`, onde são requeridas e armazenadas as estatísticas de tráfego agregado da rede (conjunto X); equipamentos de rede `TP1`, `TP2` e `TP3`, os quais são os pontos de origem das estatísticas de tráfego agregado; e `DASHClient`, onde são registrados os valores das métricas de QoS alcançados durante a exibição de vídeos (conjunto Y).

As informações correspondentes às entradas (X) e respostas (Y) são obtidas e registradas por meio de arquivos de *log*, em formato texto, em intervalos de um segundo. No cliente VLC do `DASHClient`, o código foi alterado para gerar essas informações automaticamente quando o vídeo é executado (MATOS, 2019), sendo armazenadas no diretório

[1] Como os vídeos utilizados neste trabalho são de longa duração, essa situação não é observada nos testes.

```
vlc --random --repeat -I dummy --zoom=0.15 --adaptive-logic=predictive \
  http://dashserver/playlist_final.xspf
```

Inicia o aplicativo VLC, executando a lista de reprodução remota

`http://dashserver/playlist_final.xspf`,

onde:

`--random` = executa a lista de reprodução de maneira aleatória;
`--repeat` = executa o item atual até que seja forçada a mudança para outro item;
`-I` = interface a ser usada;
    `dummy` = não usar uma interface;
`--zoom` = adiciona um fator de *zoom* de exibição;
`--adaptive-logic` = lógica de adaptação para *streaming* DASH/HLS. Valores possíveis:
`{,predictive,nearoptimal,rate,fixedrate,lowest,highest}`.

Figura 10 – Comando usado pelo gerador de carga para criar instâncias do VLC.

`/home/cloud/log`. No controlador Floodlight, foi instalado um módulo que coleta os dados dos comutadores (REZENDE, 2016).

O comando utilizado para execução do VLC no `DASHClient` é apresentado na Figura 11. Conforme pode ser verificado, faz-se uso do comando `timeout` para controlar a duração da coleta de dados: após o tempo especificado em segundos, o processo VLC é encerrado. Durante o monitoramento, foram registrados os *logs* de execução da aplicação em arquivos para fins de controle. Esses *logs*, contudo, não devem ser confundidos com o registro das métricas de QoS, realizado em outros arquivos, cuja localização e padrão de nome foram determinados durante a compilação do VLC.

Deve-se observar que, diferente do que ocorre em Pasquini e Stadler (2017) e Stadler, Pasquini e Fodor (2017), não há monitoramento no servidor, ou seja, o conjunto de características X não inclui dados provenientes do servidor (processamento, memória, etc.). Isso ocorre porque a intenção da pesquisa foi analisar apenas a influência dos dados da infraestrutura de rede, pois esses mesmos estudos demonstraram que a inclusão de dados de servidores não traz melhoras significativas aos resultados. Mais que isso, demonstraram que, entre optar pelas estatísticas da rede ou de servidores, melhor seria optar pelas primeiras, porque as mesmas compõem um conjunto de atributos menor, diminuindo o tempo de treinamento dos modelos sem perda significativa de acurácia.

3.2 Codificação dos Vídeos

Os vídeos utilizados na pesquisa foram obtidos no serviço de vídeo YouTube (YOUTUBE, 2018), em formato MP4, sendo empregados apenas para fins acadêmicos. Foram escolhidos cinco vídeos de longa duração, gerando uma lista de reprodução de quase

```
timeout 7200 vlc --verbose=2 --file-logging \
  --logfile=vlclog-`date +%Y%m%d-%H%M`.txt \
  --random --loop --adaptive-logic=predictive \
  http://dashserver/playlist_final.xspf
```

Inicia o aplicativo `vlc`, executando a lista de reprodução remota

`http://dashserver/playlist_final.xspf`,

pelo período de `7200` segundos, onde:

`--verbose` = grau de verbosidade;
`2` = depuração (*debug*).
`--file-logging` = registrar o *log* da aplicação em arquivo;
`--logfile` = nome do arquivo onde será salvo o *log*;
`--random` = executa a lista de reprodução de maneira aleatória;
`--loop` = reprodução continuada da lista de reprodução;
`--adaptive-logic` = lógica de adaptação para *streaming* DASH/HLS. Valores possíveis:
`{,predictive,nearoptimal,rate,fixedrate,lowest,highest}`.

Figura 11 – Comando usado para instanciar o cliente DASH.

5h20min para execução sem repetições. Os critérios de seleção dos vídeos foram a duração, a qualidade do vídeo (resolução e taxa de quadros por segundo) e a diversidade do tipo de movimento das cenas (imagens aceleradas, entrevistas, paisagens, etc.).

A razão para escolha de vídeos de longa duração foi permitir tempo suficiente para que as transições de vídeo ocorressem. De fato, é intuitivo pensar que vídeos de curta duração não são tão beneficiados pela tecnologia de transmissão de vídeo com taxa de bits dinamicamente adaptável quanto os de longa duração, pois essa tecnologia foi desenvolvida para compensar as variações de desempenho que ocorrem ao longo do tempo de exibição de um vídeo.

Os dados dos vídeos originais são apresentados na Tabela 2. São eles: duração; resolução (largura x altura); taxa de quadros por segundos (fps); taxa de transmissão de bits de vídeo (kbps); bits por *pixel* (bpp); e taxa de transmissão de bits de áudio (kbps). Os bits por *pixel* (em inglês, *bit per pixel* – bpp) correspondem a um parâmetro calculado para auxiliar na escolha das taxas de bits utilizadas nas codificações de vídeo e indicam "a média da quantidade de dados aplicada para cada pixel em um arquivo de vídeo"(WOWZA, 2018)[2].

O cálculo do bpp é feito dividindo-se a taxa de transmissão de bits (*bitrate*) pelo total de *pixels* por segundo, o qual é obtido multiplicando-se a quantidade de *pixels* de um quadro (indicada pela resolução) pelo número de quadros por segundo (OZER, 2011). Assim, tem-se que:

[2] Não confundir com o conceito homônimo bits por *pixel* (bpp) relacionado à profundidade de cor (*color depth*).

Tabela 2 – Dados dos vídeos originais.

Arquivo	**Duração (s)**	**Vídeo**				**Áudio**
		resolução	**fps**	**kbps**	**bpp**	**kbps**
aroundTheWorld	3600	1280 x 720	30	17578	0,636	125
biosphere	4033	1920 x 1080	30	2309	0,037	383
einstein	5377	1280 x 720	30	754	0,027	125
spanishIsles	3107	1920 x 1080	30	3359	0,054	125
teslaMotors	3016	1920 x 1080	30	2149	0,035	125

$$bpp = \frac{bits\_por\_segundo}{largura * altura * quadros\_por\_segundo} \tag{4}$$

Consequentemente, o bpp é inversamente proporcional à resolução do vídeo, de modo que quanto menor a resolução maior o bpp e vice-versa. Por isso, ele é uma ferramenta de auxílio na escolha das taxas de transmissão dentre as várias versões do componente de vídeo disponibilizado no serviço DASH. Para se manter a qualidade, quando usado o mesmo número de quadros por segundo, pode-se definir o valor de bpp para a maior resolução e usar um valor semelhante ou superior nas resoluções mais baixas. Usar um valor elevado para o bpp em altas resoluções implica em maior necessidade de largura de banda.

A escolha do bpp também é influenciada pela quantidade de movimentos da cena e, neste caso, de forma diretamente proporcional. Valores típicos encontram-se no intervalo de $0,05$ – menos movimentação – a $0,15$ – movimentação intensa (WOWZA, 2018). Assim, baixos valores de bpp para vídeo com muita movimentação resultam em baixa qualidade de exibição. O primeiro vídeo listado na Tabela 2, por exemplo, consiste de imagens aceleradas de câmera em diferentes lugares do mundo e foi disponibilizado com uma alta qualidade, exigindo uma elevada taxa de bits. O terceiro vídeo, de modo contrário, possui menos movimentação e privilegiou uma menor taxa de transmissão.

A Tabela 3 exibe os parâmetros escolhidos na pesquisa para geração das versões disponibilizadas pelo serviço DASH. Apesar das características diferentes em relação ao número de movimentos das cenas, todos os vídeos apresentados na Tabela 2 foram codificados de modo idêntico, de acordo com esses parâmetros, dando origem a cinco arquivos para cada um deles, cada arquivo com uma versão de um componente, conforme citado na Seção 3.1.2.

O motivo do uso do mesmo bpp para versões semelhantes de vídeos diferentes foi simplificar o processo de codificação e os testes preliminares, os quais tiveram como objetivo encontrar os parâmetros de vídeo e de carga adequados ao ambiente de teste implementado, evitando deixá-lo continuamente saturado ou ocioso. Por isso, também foram desenvolvidos *scripts* em Shell Script para automatizar todo o processo de codificação,

Tabela 3 – Parâmetros usados na codificação dos arquivos de mídia.

Componente	**Parâmetros**					
	codec	**resolução**	**fps**	**GOP**	**kbps**	**buffer(k)**
Vídeo	H.264	426 x 240	18	72	280	140
		854 x 480	24	96	980	490
		1280 x 720	30	120	2080	1040
Áudio	**codec**		**canais**		**kbps**	
	AAC		2		128 64	

que precisou ser refeito em várias ocasiões. Os valores do bpp correspondentes às três resoluções escolhidas são, da menor para a maior, $0,152$, $0,100$ e $0,075$.

A taxa de quadros por segundo também variou entre as versões do componente de vídeo, pois esse foi um dos parâmetros de QoS investigados (detalhado na Seção 3.3). Como foi mantido o alinhamento dos GOPs entre elas, com um GOP por segmento, essa decisão não comprometeu a capacidade de transição entre as mesmas. Com relação ao tamanho do *buffer*, este foi configurado para corresponder a 50% da taxa de bits para possibilitar as transições entre as versões do vídeo (DESGRANGE, 2017).

A ferramenta utilizada para a codificação foi o Ffmpeg. As Figuras 12 e 13 apresentam, respectivamente, um exemplo do código usado para geração dos componentes de áudio e vídeo de um dos arquivos originais, com as devidas explicações sobre cada um dos parâmetros. A diferença para o utilizado nos demais vídeos é apenas nos nomes dos arquivos de entrada e nos de saída.

```
ffmpeg -y -i einstein.mp4 -vn -c:a aac -ac 2 \
  -b:a 32k einstein_audio_32k.mp4

ffmpeg -y -i einstein.mp4 -vn -c:a aac -ac 2 \
  -b:a 128k einstein_audio_128k.mp4
```

Geram os arquivos

`einstein_audio_32k.mp4` e
`einstein_audio_128k.mp4`,

onde:

`-y` = sobrepõe o arquivo de saída (nome ao final do comando) sem perguntar;
`-i` = arquivo de entrada;
`-vn` = desconsidera o vídeo;
`-c:a` = codec de aúdio;
`-ac` = número de canais;
`-b:a` = taxa de transmissão de bits de áudio (*bitrate*).

Figura 12 – Exemplo de código usado na geração dos componentes de áudio.

```
ffmpeg -y -i einstein.mp4 -an -r 18 -c:v libx264 \
  -x264opts'keyint=72:min-keyint=72:no-scenecut' \
  -b:v 280k -maxrate 280k -bufsize 140k \
  -vf 'scale=426:240' einstein_426x240_18_280k_buf50.mp4

ffmpeg -y -i einstein.mp4 -an -r 24 -c:v libx264 \
  -x264opts 'keyint=96:min-keyint=96:no-scenecut' \
  -b:v 980k -maxrate 980k -bufsize 490k \
  -vf 'scale=854:480' einstein_854x480_24_980k_buf50.mp4

ffmpeg -y -i einstein.mp4 -an -r 30 -c:v libx264 \
  -x264opts 'keyint=120:min-keyint=120:no-scenecut' \
  -b:v 2080k -maxrate 2080k -bufsize 1040k \
  -vf 'scale=1280:720' einstein_1280x720_30_2080k_buf50.mp4
```

Geram os arquivos

`einstein_426x240_18_280k_buf50.mp4`,
`einstein_854x480_24_980k_buf50.mp4` e
`einstein_1280x720_30_2080k_buf50.mp4`,

onde:

`-y` = sobrepõe o arquivo de saída (nome ao final do comando) sem perguntar;
`-i` = arquivo de entrada;
`-an` = desconsidera o áudio;
`-r` = taxa de quadros/segundo (fps);
`-c:v` = codec de vídeo;
`-x264opts` = configura parâmetros x264 como uma lista, separados por dois pontos (:);
- `keyint` = tamanho máximo do GOP;
- `min-keyint` = tamanho mínimo do GOP;
- `no-scenecut` = desabilita completamente decisão adaptativa de quadros I-frame;
- `keyint=X:min-keyint=X:no-scenecut` = força o codificador a usar um GOP de tamanho X constante;

`-b:v` = taxa de transmissão de bits de vídeo (*bitrate*);
`-maxrate` = taxa de bits máxima em qualquer ponto do vídeo (requer que o tamanho do *buffer* seja configurado);
`-bufsize` = tamanho do *buffer* em bits;
`-vf` = filtro de vídeo;
`scale` = resolução do vídeo.

Figura 13 – Exemplo de código usado na geração dos componentes de vídeo.

As versões criadas com o Ffmpeg, entretanto, ainda não correspondem às finais. Elas são usadas como entrada para o MP4Box, aplicação empregada para a criação dos arquivos de manifesto e das versões segmentadas dos componentes. São os arquivos gerados com o MP4Box que devem ser disponibilizados no servidor para a exibição do vídeo. As versões intermediárias não são necessárias.

Os principais parâmetros envolvidos nesse processo de formatação das versões para um formato aceitável pelo padrão DASH são: a duração do segmento; e o perfil do padrão MPEG-DASH aplicado. A decisão sobre o primeiro deles envolveu alguns testes preliminares, onde foram testados segmentos de 1, 2, 4 e 8s, optando-se por realizar os

experimentos com vídeos com segmentos de 4s de duração. Esse valor está de acordo com o recomendado por soluções comerciais (LEDERER, 2015). Quanto ao perfil, esta foi uma escolha direta por se tratar de vídeo sob demanda e não transmissão ao vivo.

O comando MP4Box usado para a geração do arquivo de manifesto e das versões segmentadas de um dos vídeos originais, com a descrição dos parâmetros envolvidos, é reproduzido na Figura 14. Pode-se notar que os nomes dos arquivos de saída com as versões de componentes segmentadas não aparecem no comando, dando a impressão de que as versões informadas seriam sobrescritas. Todavia, como subentendido da figura, para os perfis `onDemand`, por padrão, o MP4Box cria novos arquivos, acrescentando o termo `_dashinit` antes da extensão. Assim, se o nome do arquivo de entrada é `file.mp4`, o de saída será `file_dashinit.mp4`.

```
MP4Box -dash 4000 -rap -frag-rap -profile dashavc264:onDemand \
  -out einstein.mpd \
  einstein_426x240_18_280k_buf50.mp4 \
  einstein_854x480_24_980k_buf50.mp4 \
  einstein_1280x720_30_2080k_buf50.mp4 \
  einstein_audio_32k.mp4 \
  einstein_audio_128k.mp4
```

Utiliza como entrada

`einstein_426x240_18_280k_buf50.mp4`,
`einstein_854x480_24_980k_buf50.mp4`,
`einstein_1280x720_30_2080k_buf50.mp4`,
`einstein_audio_32k.mp4` e
`einstein_audio_128k.mp4`.

Gera os arquivos

`einstein.mpd`,
`einstein_426x240_18_280k_buf50_dashinit.mp4`,
`einstein_854x480_24_980k_buf50_dashinit.mp4`,
`einstein_1280x720_30_2080k_buf50_dashinit.mp4`,
`einstein_audio_32k_dashinit.mp4` e
`einstein_audio_128k_dashinit.mp4`.

Onde:

`-dash` = duração do segmento em milissegundos;
`-rap` = força o segmento a começar com um ponto de acesso aleatórios;
`-frag-rap` = todos os fragmentos começarão com pontos de acesso aleatório;
`-profile` = perfil MPEG-DASH (configura valores opcionais padrões para o perfil desejado);
`-out` = arquivo de saída (.mpd).

Figura 14 – Exemplo de comando MP4Box usado na criação de arquivos MPD e de versões dos vídeos em formato aceitável pelo padrão DASH.

Embora não apresentadas neste documento, as instalações das ferramentas de codificação estão descritas de forma didática em Lacerda (2018), um tutorial desenvolvido

para documentar a criação de um serviço DASH para teste. O tutorial utiliza o VirtualBox como aplicativo de virtualização e o serviço pode ser acessado do navegador da própria máquina hospedeira. Ele também detalha todo o procedimento de codificação para obtenção de uma versão final de vídeo em formato compatível com o DASH.

3.3 Metodologia Aplicada

A metodologia adotada nesta pesquisa foi apresentada em Pasquini e Stadler (2017) e consiste no emprego de métodos de aprendizagem de máquina supervisionada para estimar métricas de QoS utilizando como atributos de entrada dados estatísticos de tráfego agregado dos dispositivos de rede. Mais especificamente, busca-se entender quanto de conhecimento sobre a qualidade de serviço obtida nos clientes pode ser extraído da rede.

O estudo baseia-se nos resultados obtidos por esses autores, os quais indicaram ser possível atingir erro médio inferior a 10% para algumas métricas, como a taxa de quadros por segundo em um serviço de vídeo sob demanda e tempo de resposta de leitura em uma aplicação *key-value store*. A intenção da pesquisa aqui descrita foi verificar quais seriam os resultados para uma terceira aplicação, um serviço de vídeo com taxa de bits dinamicamente adaptável sobre HTTP.

De modo genérico, o problema pode ser definido como segue:

Problema geral. *O objetivo é estimar o conjunto de métricas contínuas de serviço $\hat{Y}_t$, no instante t, em um cliente, com base no conhecimento extraído do conjunto de estatísticas de infraestrutura $X_t = [X_{1t}, ..., X_{dt}]$ de d dispositivos. Ou seja, mapear*

$$M : X_t \rightarrow \hat{Y}_t,$$

com $\hat{Y}_t$ muito próximo de Y_t, os valores reais observados, para um dado X_t, caracterizando-se um problema de regressão em aprendizagem de máquina supervisionada.

Para o serviço em análise, as métricas Y de QoS são as mesmas analisadas para a aplicação de vídeo sob demanda: a taxa de quadros de vídeo (quadros/seg ou fps) e a taxa de *buffer* de áudio (*buffers*/seg). Como já citado, essas métricas são obtidas configurando-se a aplicação cliente para que as registre em arquivos de *log*, em formato texto, a cada segundo.

O conjunto de características de infraestrutura X é composto por X_{port}, o qual representa os dados coletados por porta em todos os três comutadores OpenFlow que compõem a rede entre o cliente e o servidor, consistindo dos números totais de bytes e de pacotes recebidos e enviados por porta. Esses dados são obtidos por meio de módulo de monitoramento no controlador OpenFlow.

Assim, particularizando para a pesquisa:

Problema específico. *Mapear, para um instante t,*

$$M : X_t \rightarrow \hat{Y}_t,$$

com $\hat{Y}_t$ muito próximo de Y_t para um dado X_t, onde:

$Y_t = \{y_{fps_t}, y_{buffers/s_t}\}$;
$\hat{Y}_t = \{\hat{y}_{fps_t}, \hat{y}_{buffers/s_t}\}$;
$X_t = X_{port_t}$
$X_{port_t} = X_{bytesin_t} \cup X_{bytesout_t} \cup X_{packetsin_t} \cup X_{packetsout_t}$;

É importante ressaltar que o conjunto X corresponde aos dados de todas as portas dos três dispositivos presentes na rede OpenFlow da Figura 9 que estejam relacionadas ao serviço DASH. Considerando que há sete portas envolvidas, duas nos roteadores TP1 e TP2 e três no TP3, o subconjunto X_{port} tem 28 atributos. Consequentemente, o conjunto X é composto de 28 atributos.

Para coleta dessas informações, optou-se por realizar duas execuções do serviço com duas horas de duração cada (7200s) e níveis de carga diferentes. Cada uma delas gerou *traces* (séries temporais) distintos. A primeira dessas execuções ocorreu com carga nula, denominada nível 0 (N_0), cuja finalidade foi observar o comportamento do serviço quando não ocorrem interferências e nortear a análise do outro nível de carga. Este, por sua vez, chamado nível 1 (N_1), utiliza o padrão de carga citado na Seção 3.1.2, com parâmetros conforme Tabela 4. Como pode ser observado nessa tabela, o valor inicial P_S corresponde a seis instâncias do cliente VLC, com amplitude P_A de até três instâncias do cliente VLC e período da função senoidal P_T em 30 minutos.

Tabela 4 – Parâmetros usados no gerador de carga.

Nível	P_A	P_S	P_T
N_1	3	6	30

As Figuras 15 e 16 apresentam os resultados obtidos para a taxa de quadros por segundo. Observando-se os gráficos, é perceptível que o efeito da carga corresponde ao padrão de carga utilizado. Também é perceptível, pela Figura 15, que, mesmo quando não há carga empregada, há pequena variação de versão do vídeo. Uma possível explicação para isso é que uma pequena alteração na rede tenha levado o mecanismo de adaptação a requisitar uma versão de menor qualidade por um pequeno espaço de tempo.

Para a métrica *buffers* por segundo, os gráficos correspondentes ao tráfego coletado com o gerador de carga inativo e quando o mesmo é ativado com carga conforme padrão

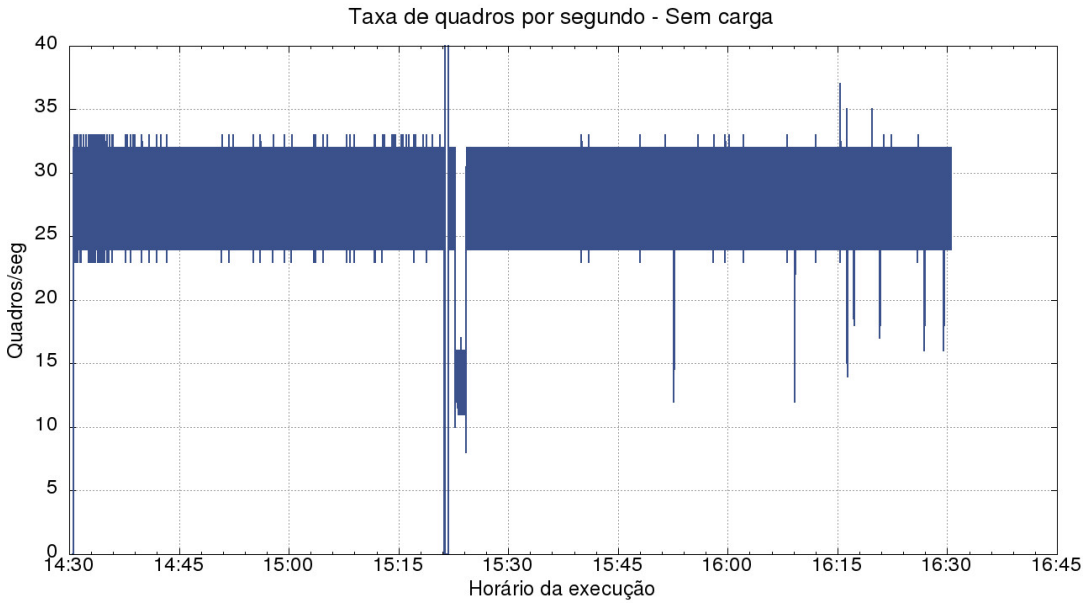

Figura 15 – Resultado da execução realizada sem carga para a taxa de quadros por segundo.

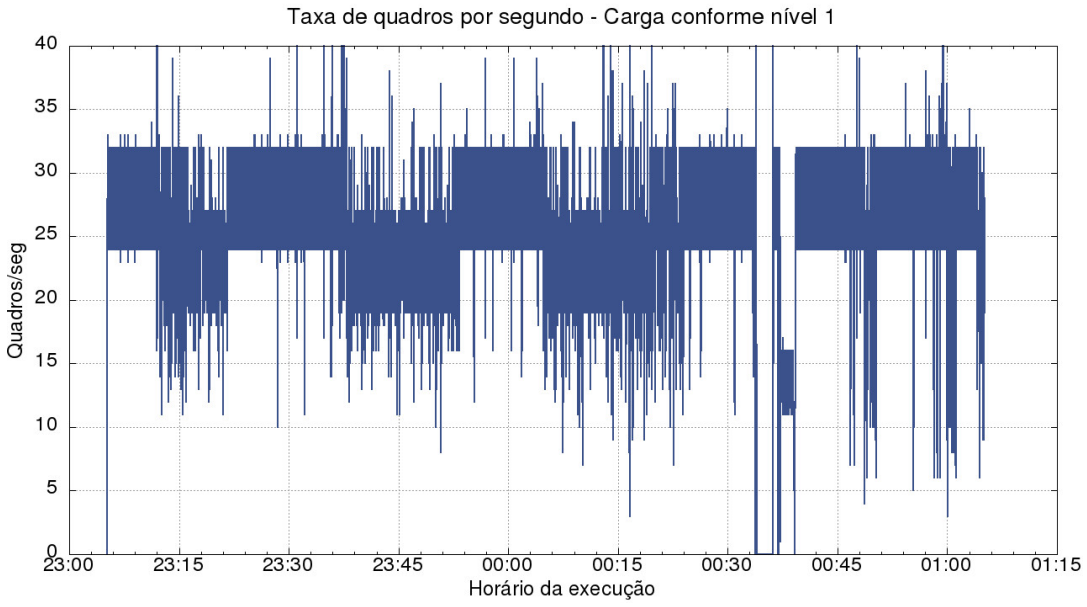

Figura 16 – Resultado da execução realizada com a carga no nível N_1 para a taxa de quadros por segundo.

de carga N_1 são exibidos, respectivamente, nas Figuras 17 e 18. Embora as alterações causadas nessa métrica de QoS para o padrão de carga N_1 sejam menores que na de quadros por segundo, verifica-se que ela também sofre oscilações devido à carga. Como os componentes de vídeo demandam mais taxa de transmissão do que os de áudio, optou-se por utilizar a métrica de vídeo como referência para escolha do padrão de carga a ser empregado.

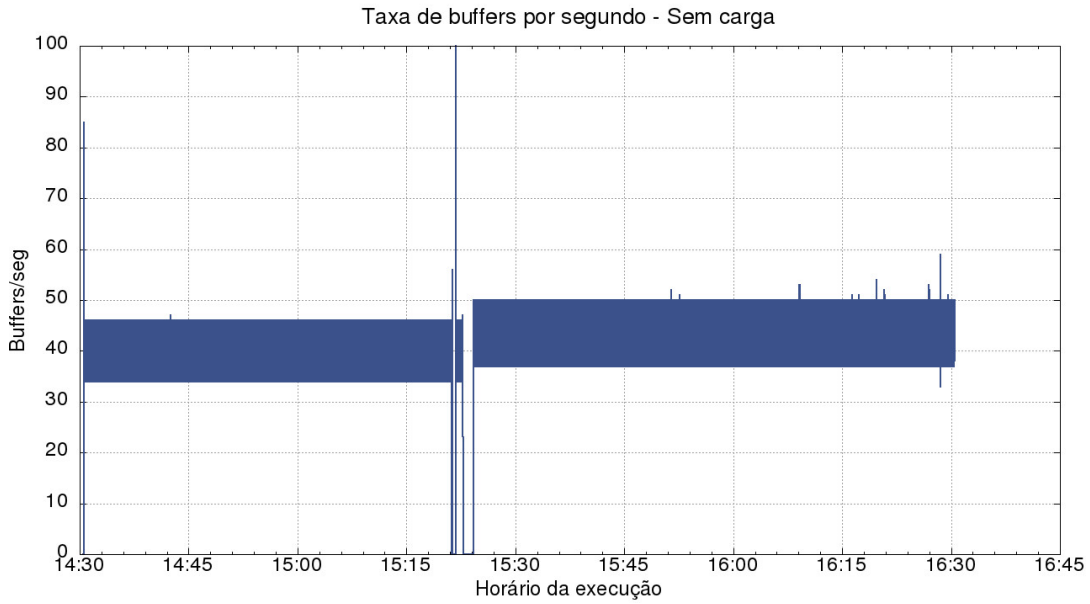

Figura 17 – Resultado da execução realizada sem carga para *buffers* por segundo.

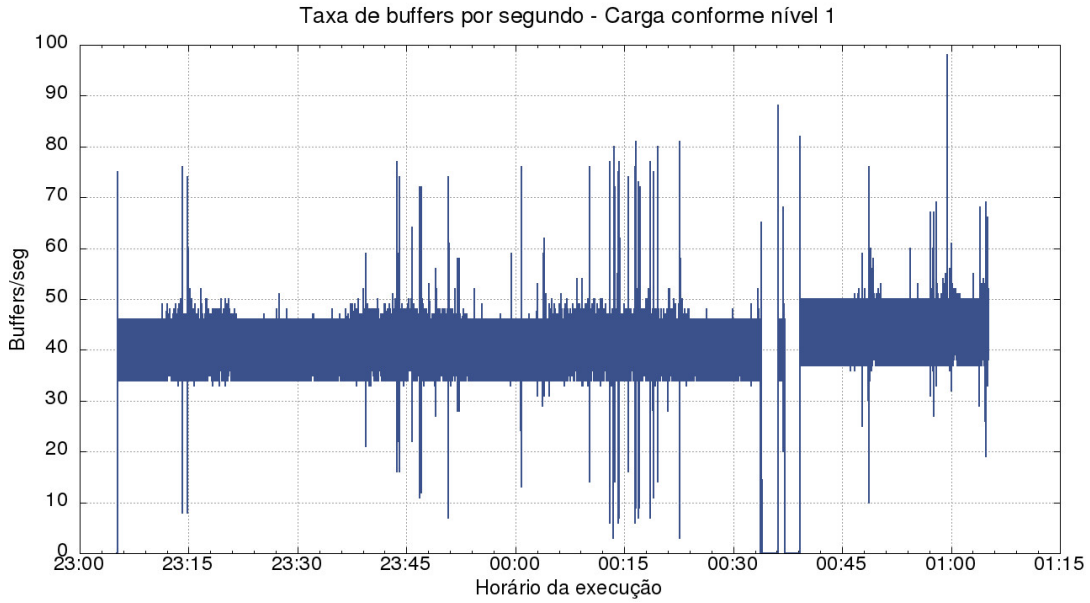

Figura 18 – Resultado da execução realizada com a carga no nível N_1 para *buffers* por segundo.

Conceitualmente, a série temporal gerada corresponde à junção, com base no instante em que foram observadas, das estatísticas de rede com as métricas de QoS, sendo referenciadas por $\{(X_t, Y_t)\}$. Ou seja, trata-se do conjunto de pares X e Y observados no mesmo instante t. Em termos práticos, foram usados dois arquivos para o nível de carga N_1, sendo um para o conjunto X e outro para o conjunto Y. Isso porque os métodos de aprendizagem de máquina empregados recebem as informações de atributos e rótulos

separadamente, não sendo necessário agrupá-los em um único arquivo.

O arquivo referente ao conjunto de atributos contém, em cada uma de suas linhas, uma entrada com as estatísticas por porta dos três roteadores, em sequência, devidamente identificadas. Esse arquivo foi gerado com o auxílio de *scripts* desenvolvidos em Shell Script, pois os dados são registrados pelo controlador em arquivos distintos para cada roteador e porta. Essa formatação incluiu ainda filtrar os instantes em que houve registros para todas as portas e também no cliente. No total, dentre 7.200 amostras esperadas, 6.962 se confirmaram válidas – ou seja, possuíam todas as informações registradas.

Já o arquivo correspondente ao conjunto Y, possui a mesma forma do registrado em *log* pelo cliente DASH, à exceção de terem sido filtradas as entradas para os instantes nos quais houve registro em todas as portas, permitindo a correspondência direta entre atributos e rótulos. Dessa forma, é um arquivo com o mesmo número de amostras do arquivo de atributos, 6.962. A seleção da métrica a ser utilizada é realizada no código Python de geração do modelo, não sendo obrigatório gerar um arquivo para cada uma delas.

De posse da série, pôde-se realizar os experimentos, isto é, gerar as estimativas sobre as métricas de QoS por meio do emprego dos métodos de aprendizagem de máquina árvore de regressão e floresta aleatória. Os experimentos também envolveram o emprego de método de redução de atributos com a finalidade de encontrar o conjunto mínimo. Finalizados os experimentos, os resultados obtidos em cada um deles foram avaliados por meio das medidas de avaliação de desempenho $NMAE$ e tempo de treinamento, as quais foram descritas na Seção 2.3. Pôde-se então analisar a viabilidade ou não dos modelos gerados na extração de conhecimento a partir de estatísticas de tráfego agregado da rede para o serviço e métricas de QoS investigadas.

Como pôde ser confirmado na descrição da metodologia, a vantagem da mesma é ser independente do serviço. Nela, as características são selecionadas dinamicamente na criação dos modelos, refletindo a influência que cada uma delas exercem na aplicação em análise. No contexto das redes de computadores e computação em nuvem, isso significa poder identificar quais equipamentos mais interferem na qualidade de serviço (os do núcleo, da borda, os de tráfego mais intenso, etc.) e como o monitoramento deve ser atacado (estatísticas por fluxos, por porta, de processamento, etc.). Essa percepção será ainda maior no próximo capítulo, onde são detalhados os experimentos selecionados, as ferramentas utilizadas para execução dos mesmos, bem como os resultados obtidos.

Capítulo 4

Experimentos e Análise dos Resultados

Os experimentos realizados com a série temporal gerada a partir dos dados coletados na execução do serviço DASH com gerador de carga ativado são apresentados neste capítulo. Conforme detalhado na Seção 3.3, a série, que na verdade é composta de dois arquivos separados, corresponde a duas horas de coleta e possui 6.962 amostras. Na Seção 4.1, são detalhados os parâmetros utilizados em cada um dos experimentos e, na Seção 4.2, são analisados os resultados obtidos.

4.1 Experimentos

Os experimentos efetuados consistiram no uso da série temporal descrita na Seção 3.3 para geração de modelos de predição das métricas taxa de quadros de vídeo por segundo e *buffers* de áudio por segundo. Foram utilizados como método de aprendizagem de máquina supervisionada os algoritmos de árvore de regressão e floresta aleatória. Eles foram implementados usando-se, respectivamente, as classes `DecisionTreeRegressor` e `RandomForestRegressor` da biblioteca `sklearn` da linguagem Python (PEDREGOSA et al., 2011).

Os parâmetros utilizados nos experimentos são apresentados na Tabela 5. Optou-se por também configurar o parâmetro randômico (`random_state`) para obter maior fidelidade na comparação do uso do conjunto completo de atributos X e do conjunto de atributos mínimos, X_{min}. Para o algoritmo de árvore de regressão, os parâmetros foram configurados após algumas tentativas, empregando-se valores aleatórios e considerando as alterações nas medidas de avaliação como limites de parada.

Como pode ser verificado na Tabela 5, os experimentos envolvendo o algoritmo de floresta aleatória (`RandomForestRegressor`) variaram amplamente o número de árvores, possibilitando a análise quanto a influência do número de estimadores nos resultados. De modo semelhante ao que ocorreu para o método de árvore de decisão, os demais parâmetros foram definidos com base em testes preliminares, empregando-se quantidades de

Tabela 5 – Parâmetros dos experimentos realizados.

Métrica	**DecisionTreeRegressor**		**RandomForestRegressor**		
	max_depth	**random_state**	**max_depth**	**random_state**	**n_estimators**
quadros/seg	5	10	8	5	2-150
buffers/seg	3	10	3	5	2-150

estimadores fixas e tendo como objetivo melhorar o tempo de treinamento sem prejudicar o erro alcançado.

Os resultados obtidos pela execução dos algoritmos foram gravados em arquivo texto, com os dados dos experimentos (parâmetros e resultados) separados por vírgula. No caso do `RandomForestRegressor`, posteriormente o arquivo foi classificado de acordo com as métricas de avaliação NMAE e tempo de treinamento, consideradas de forma conjunta e nesta ordem. Dessa classificação, retirou-se os dez melhores em relação ao erro NMAE e ao tempo de treinamento, salvando-se essa informação em dois novos arquivos. Essa última classificação fez-se necessária para eliminar resultados iguais para uma dada medida de avaliação, mas com baixo desempenho na outra, sendo o critério de desempate o menor número de árvores. Esse critério foi utilizado, porque, como será apresentado na Seção 4.2, variando-se o número de árvores utilizadas, os melhores resultados variam na ordem de décimos – ou menos – de percentual de erro NMAE, enquanto há uma maior variação no tempo de treinamento.

A intenção dessas listagens intermediárias foi encontrar uma lista final contendo no máximo dois valores satisfatórios para ambas as métricas, o que foi conseguido a partir do cruzamento desses arquivos. De posse dessa lista final, foi possível realizar novos experimentos que investigassem o conjunto de atributos mínimos apenas para os números de árvores selecionados. Ou seja, para cada uma das duas quantidades de árvores selecionadas, o algoritmo `RandomForestRegressor` foi executado analisando-se os a melhores atributos, com a variando de 1 a 28 (quantidade total de atributos).

O algoritmo de redução de atributos empregado foi o de seleção univariada, descrito na Seção 2.3.3. Esse mesmo procedimento de identificação do conjunto mínimo foi realizado para algoritmo de árvore de regressão (`DecisionTreeRegressor`), porém, obviamente, sem a necessidade de seleção do número de árvores, já que não se trata de um método de agregação como o de floresta aleatória.

Os resultados obtidos pela execução do experimentos descritos são apresentados e analisados na próxima seção. No restante deste documento, o algoritmo de árvore de regressão será referenciado nas tabelas como a entrada cujo número de estimadores é igual a 1.

4.2 Avaliação dos Resultados

As Figuras 19 e 20 sintetizam, respectivamente, os erros NMAE e o tempo de treinamento encontrados com o emprego do algoritmo floresta aleatória para estimar a métrica de vídeo quadros por segundo, de acordo com o número de estimadores (árvores) empregado. É possível perceber pelas figuras que, para a série temporal em análise, o tempo de treinamento é muito mais sensível à variação do número de estimadores do que o NMAE. Isso também é verdadeiro para a métrica de áudio *buffers* por segundo, conforme pode ser verificado nas Figuras 21 e 22.

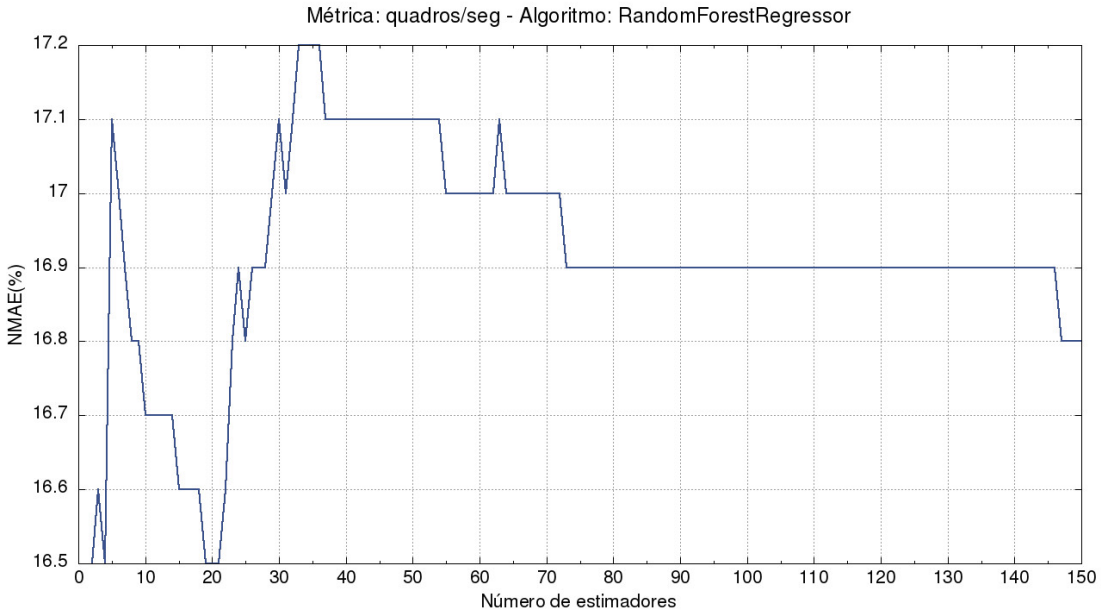

Figura 19 – NMAE obtidos para a métrica quadros/seg com o emprego do algoritmo floresta aleatória.

A análise dos gráficos para ambas as métricas de QoS demonstra também que, diferente do sugerido pelo trabalho que propôs a metodologia (PASQUINI; STADLER, 2017), no qual empregou-se o algoritmo floresta aleatória com 120 árvores, usar uma quantidade maior de estimadores para os dados investigados não se mostrou adequado. Como apresentado na Tabela 6, o cruzamento dos melhores resultados de ambas as medidas de avaliação revelou ser mais interessante ter poucos estimadores. Os dois melhores resultados considerando-se conjuntamente o erro NMAE e o tempo de treinamento ocorreram com o uso de 2 e 3 árvores. Isso provavelmente está relacionado ao fato de serem utilizados neste trabalho menos atributos no conjunto X.

Além disso, observa-se ainda que, para a métrica de QoS de vídeo, os erros obtidos para o serviço DASH foram maiores do que os encontrados para a aplicação de vídeo sob demanda tradicional na análise desenvolvida no trabalho anteriormente citado. Naquela ocasião, obteve-se erros inferiores a 10%. Na verdade, esse aumento no erro não

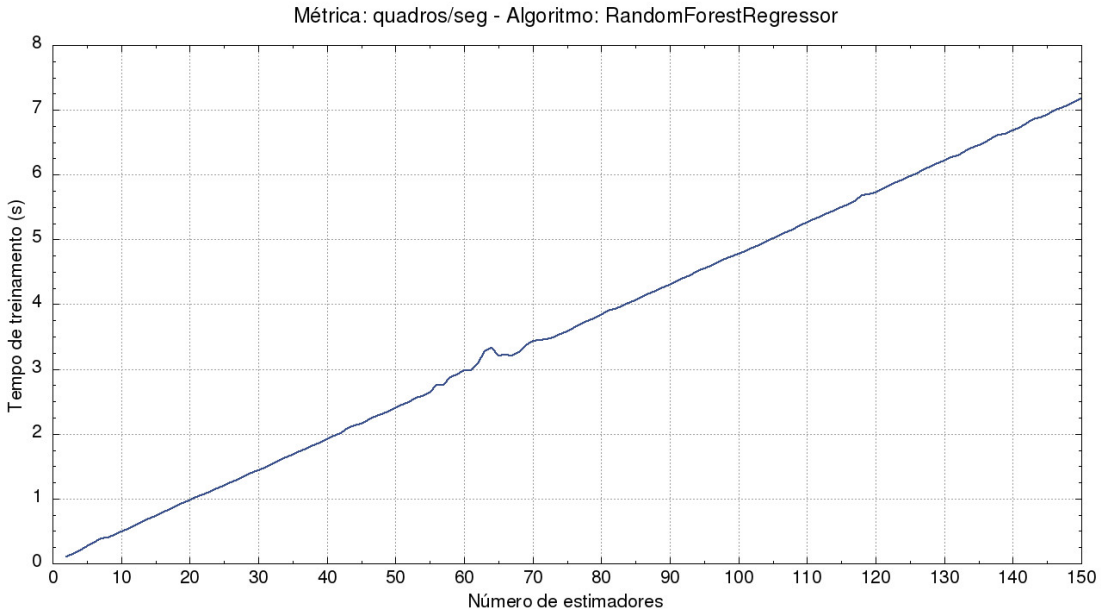

Figura 20 – Tempos de treinamentos obtidos para a métrica quadros/seg com o emprego do algoritmo floresta aleatória.

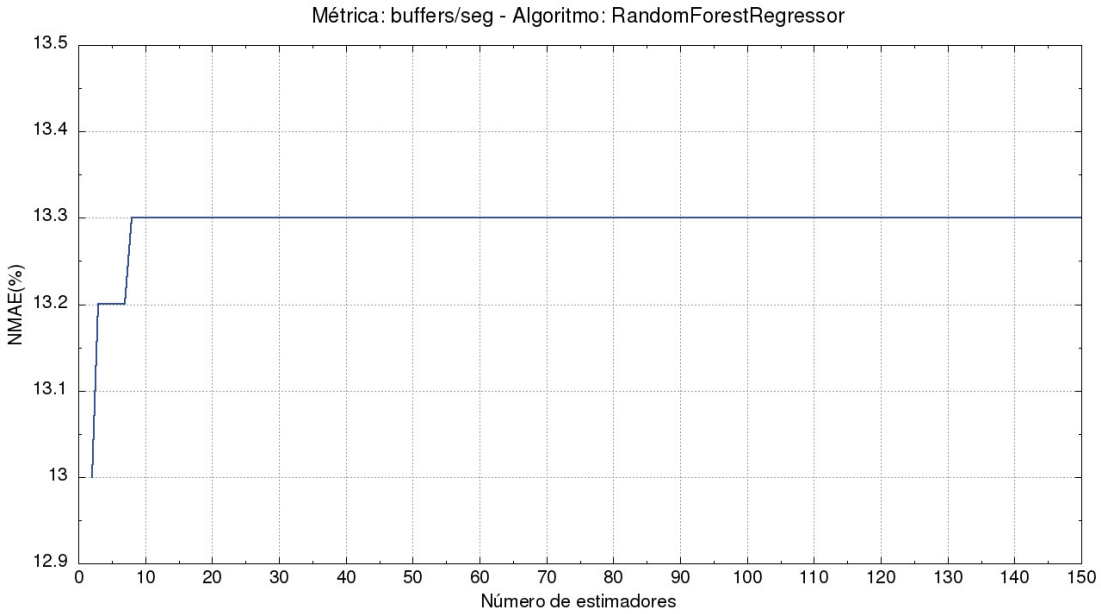

Figura 21 – NMAE obtidos para a métrica *buffers*/seg com o emprego do algoritmo floresta aleatória.

surpreende, pois o serviço DASH é um serviço menos estável no que se refere às taxas de transmissão, com muitas oscilações, possibilitando naturalmente maior margem de erro. Como o componente de áudio é menos sensível a estas oscilações, foi possível obter para a métrica *buffers* por segundo um resultado melhor do que para o taxa de quadros por segundo, sendo, inclusive, melhor até do que a conseguida com a aplicação de vídeo sob

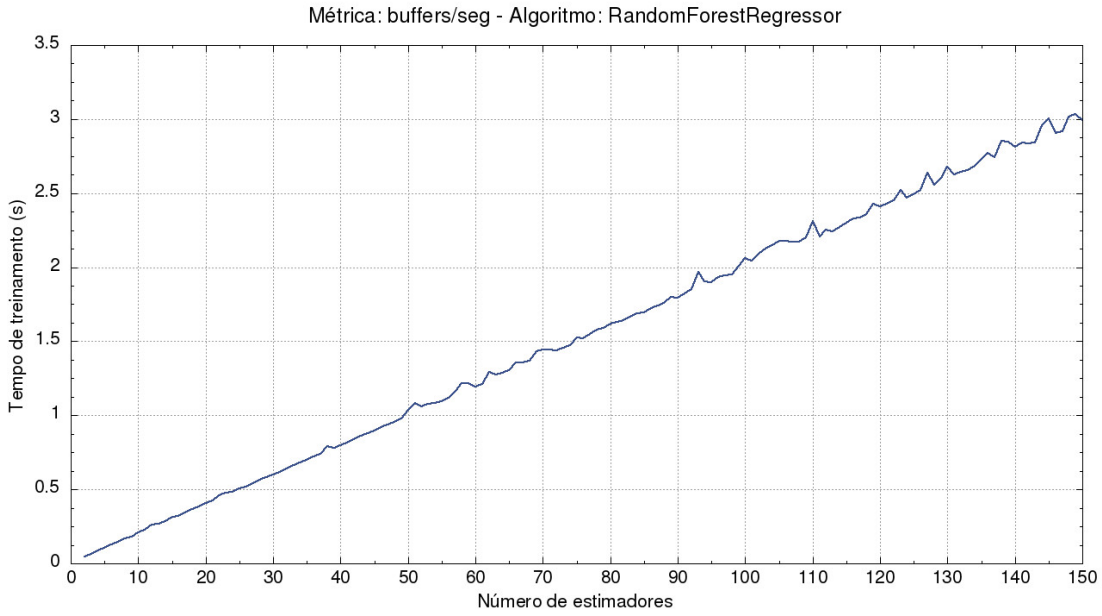

Figura 22 – Tempos de treinamentos obtidos para a métrica *buffers*/seg com o emprego do algoritmo floresta aleatória.

Tabela 6 – Melhores resultados considerando-se ambas medidas e todos os atributos.

Número de estimadores	**Quadros/seg**		***buffers*/seg**	
	NMAE (%)	**Tempo de Treino (s)**	**NMAE (%)**	**Tempo de Treino (s)**
1	16,6	0,050	13,3	0,031
2	16,5	0,109	13,0	0,047
3	16,6	0,150	13,2	0,066

demanda, que ficou em torno de 20% para essa métrica.

Contudo, para além das diferenças entre os serviços, como o tamanho das séries temporais (número de amostras) analisadas nos dois trabalhos foram diferentes, sendo menor na pesquisa aqui descrita, é possível questionar se talvez uma série temporal com mais amostras gere melhores resultados, uma vez que o conjunto de treinamento será maior. Porém, embora essa investigação seja válida e será realmente efetuada num futuro próximo, ressalta-se que o conjunto de teste também terá mais amostras, pois eles são proporcionais ao conjunto total de dados, e há sempre a possibilidade de um dado algoritmo de aprendizagem de máquina não ser adequado a um determinado problema, de modo que mais amostras de testes sendo preditas com erro prejudiquem ainda mais a acurácia. Neste último caso, outros algoritmos de aprendizagem poderão ser investigados para verificar se uma taxa de erro menor do que 10%, sem prejuízo do tempo de treinamento, pode ser conseguida para essas métricas para o serviço DASH.

O que é notável pelos dados da Tabela 6, considerando que ela contém os melhores resultados num conjunto de testes que analisou de 2 a 150 estimadores no algoritmo flo-

resta aleatória, é que não há benefício no emprego deste método de agregação para a série temporal estudada. O algoritmo árvore de regressão apresenta valores NMAE aproximados aos dois melhores erros obtidos pelo algoritmo de floresta aleatória e ainda possui melhor tempo de treinamento para as duas métricas de QoS investigadas. É importante destacar aqui que, apesar de as duas métricas apresentarem melhores resultados para o mesmo número de estimadores quando empregado o algoritmo floresta aleatória, os resultados foram obtidos de forma independente, conforme descrito na Seção 4.1, sendo uma particularidade da conjunto série temporal, serviço e métricas de QoS analisados.

Como detalhado na Seção 4.1, além da análise completa do conjunto X, os experimentos englobaram a definição do conjunto mínimo de atributos, X_{min}, para cada uma das métricas de QoS investigadas. Para isso, foram empregados os mesmos parâmetros dos experimentos do conjunto completo. Os resultados desses experimentos demonstraram que, para a métrica de áudio *buffers* por segundo, o atributo que mais influenciou na predição foi o número de bytes transmitidos na porta número 2 do roteador TP3, conforme dados apresentados na Tabela 7. Essa porta corresponde à interface em que o cliente se conecta à rede OpenFlow.

Tabela 7 – Conjunto mínimo de atributos para a métrica *buffers*/seg.

Número de estimadores	**Tamanho do conjunto**	**NMAE (%)**	**Tempo de Treino (s)**	**Atributos**
1	1	13,3	0,004	`3_2_TxBytes`
2	1	13,1	0,009	`3_2_TxBytes`
3	1	13,2	0,012	`3_2_TxBytes`

Desse modo, os resultados indicam que, para essa métrica e algoritmos analisados, é suficiente utilizar a informação de bytes transmitidos do servidor para o cliente para desenvolver modelos de predição com o mesmo nível de erro de quando são empregados todos os atributos, obtendo ainda a vantagem de ter o tempo de treinamento do modelo reduzido. Tem-se aqui o cuidado em citar apenas o tráfego do servidor para o cliente, pois não foram investigadas situações em que tráfego com outra origem é transmitido pela interface na qual o cliente se conecta à rede.

Conseguir conjuntos mínimos de tamanho 1 com o mesmo atributo explica porque o uso do método de agregação não trouxe benefícios para a métrica em questão. Na verdade, como pode ser visto na Tabela 8, que apresenta os resultados para a métrica de QoS quadros por segundo, mesmo conjuntos mínimos de maior tamanho, mas com os mesmos atributos, podem gerar esse efeito na comparação da árvore de regressão com o algoritmo floresta aleatória.

Na definição do conjunto mínimo para a métrica de vídeo, conseguiu-se obter um erro semelhante ao do conjunto completo, com uma redução significativa do número de atributos – na ordem de 75%. Como o objetivo é também reduzir o tempo de treinamento,

Tabela 8 – Conjunto mínimo de atributos para a métrica quadros/seg.

Número de estimadores	**Tamanho do conjunto**	**NMAE (%)**	**Tempo de Treino (s)**	**Atributos**
1	7	16,6	0,018	2_3_RxPacktes, 3_2_RxPacktes, 3_2_TxPacktes, 3_2_RxBytes, 3_2_TxBytes, 3_3_TxPacktes e 3_3_TxBytes
2	7	16,6	0,046	2_3_RxPacktes, 3_2_RxPacktes, 3_2_TxPacktes, 3_2_RxBytes, 3_2_TxBytes, 3_3_TxPacktes e 3_3_TxBytes
3	7	16,6	0,066	2_3_RxPacktes, 3_2_RxPacktes, 3_2_TxPacktes, 3_2_RxBytes, 3_2_TxBytes, 3_3_TxPacktes e 3_3_TxBytes

foram selecionados os que melhor satisfizeram essas duas medidas de avaliação. Conforme pode ser verificado nos atributos listados na Tabela 8, a interface do cliente continua exercendo papel chave para extração do conhecimento, porém, neste caso, as estatísticas de tráfego nos dois sentidos (Tx e Rx) e nos dois níveis de granularidade (pacotes e bytes) se fazem necessárias. Além dessas estatísticas, chama a atenção o fato de serem relevantes o número de bytes e pacotes transmitidos na porta 3 do comutador TP3 para o comutador TP2, e, como atributo principal, o número de pacotes recebidos nesta porta do comutador TP2 proveniente do comutador TP3. Ou seja, os dados com requisições para o servidor tanto do cliente quanto do gerador de carga também influenciam na extração do conhecimento.

Capítulo 5

Conclusão

O objetivo da pesquisa aqui apresentada foi investigar como métricas de qualidade de serviço em um cliente DASH se relacionam com estatísticas de tráfego agregado de rede em uma rede OpenFlow. Para isso, foi empregada a metodologia proposta em Pasquini e Stadler (2017), a qual encontra-se detalhada na Seção 3.3. Basicamente, essa metodologia consiste em obter dados estatísticos do tráfego agregado da rede e utilizá-los como entradas para métodos de aprendizagem de máquina supervisionada, os quais geram modelos que estimam métricas de QoS observadas no cliente de um dado serviço.

Como a abordagem de aprendizagem de máquina usada é a supervisionada, valores reais para as métricas de QoS devem ser coletados no cliente para criação dos modelos. Na pesquisa desenvolvida, as métricas de QoS analisadas para o serviço DASH foram: a taxa de quadros por segundo, para o componente de vídeo; e a taxa de *buffers* por segundo, para o componente de áudio. Essas métricas foram obtidas empregando-se a modificação no cliente VLC desenvolvida em Matos (2019).

As estatísticas de tráfego agregado usadas foram o número de bytes e de pacotes enviados e recebidos nas interfaces de redes que integram os caminhos entre os clientes (cliente e gerador de carga) e o servidor. Elas foram registradas através de uma extensão do controlador Floodlight elaborada por Rezende (2016).

Durante a obtenção desses dados, tomou-se o cuidado de garantir a sincronização dos relógios das máquinas envolvidas no processo, de modo que pudesse haver uma relação inequívoca entre o conjunto de atributos (correspondente às estatísticas de rede) e os rótulos (métricas observadas no cliente) num dado instante de tempo t. O intervalo de coleta de dados utilizado foi de 1 segundo.

Uma das etapas do projeto contemplou a implementação do ambiente de testes, cujos principais componentes são a rede OpenFlow e o serviço DASH, conforme apresentado na Figura 9. Fisicamente, foram utilizados três *notebooks*, três roteadores com suporte ao protocolo OpenFlow, dois comutadores não gerenciáveis e quatro adaptadores USB com interface RJ45 para expansão do quantidade de interfaces de rede cabeada de dois dos *notebooks* utilizados. A rede OpenFlow foi composta por um controlador virtualizado e os

três roteadores físicos citados, sendo responsável pela interligação do servidor DASH com o cliente DASH e com o gerador de carga (todos virtualizados). Também foi utilizado um servidor virtual auxiliar fornecendo serviço NTP para garantir a sincronização de relógio.

Como muitos dos componentes do ambiente de testes são virtuais, eles poderão ser usados em outros cenários onde se pretenda analisar um serviço DASH, sem que sejam necessárias grandes modificações. Embora não tenha sido possível, por questões de prazo, automatizar o processo de início de coleta de dados de monitoramento, outros pontos do projeto, incluindo a codificação de vídeo para o formato DASH, a formatação da série temporal e os vários experimentos realizados com a mesma foram realizados de forma total ou parcialmente automatizada por meio de *scripts*.

Para os experimentos, foram coletados dados de execução do serviço durante duas horas, com o gerador de carga ativado com padrão de carga periódica, o que resultou na criação de uma série temporal com 6.962 amostras. Apesar dos dados serem coletados a cada segundo, o número menor de amostras se deu pelo fato de ser necessário cruzar as informações de todos os pontos de coletas, retirando os instantes em que houve falha em algum deles. Essa série foi então empregada para geração dos modelos de predição, o que foi realizado utilizando-se a linguagem de programação Python.

Os algoritmos de aprendizagem de máquina empregados para a criação do modelos de predição foram a árvore de regressão e o algoritmo de agregação floresta aleatória. Este último foi amplamente avaliado considerando-se o número de estimadores usados. A avaliação do desempenho dos modelos gerados foi efetuada por meio das medidas de avaliação erro absoluto médio normalizado (NMAE) e tempo de treinamento do modelo.

Os resultados para as duas métricas avaliadas demonstraram que não houve benefício no emprego de mais estimadores, sendo que os melhores resultados podem ser alcançados pelo algoritmo de árvore de regressão ou pelo floresta aleatória quando este é configurado com dois ou três estimadores. Também foram encontrados conjuntos mínimos de atributos com redução considerável no número de atributos, o que favorece ainda mais o tempo de treinamento dos modelos.

Para a métrica *buffers* por segundo, o erro ficou em torno de 13%, sendo identificada uma influência maior dos dados obtidos na interface que o cliente se conecta à rede, mais especificamente, aos pacotes transmitidos do servidor para o cliente. De fato, o conjunto mínimo de atributos obtido foi composto por essa única métrica. Isso fez também com que a profundidade da árvore criada não influenciasse muito nos resultados, permitindo o uso de uma árvore de menor profundidade (profundidade máxima = 3), o que reduziu o tempo de treinamento, atingindo a ordem de milissegundos.

A influência dos dados obtidos no equipamento de borda que conecta o cliente e o gerador de carga à rede foi observada também para o métrica quadros por segundo. Neste caso, o erro ficou em torno de 16,6%, sendo que o conjunto mínimo de atributos obtido incluiu sete atributos: quatro deles da porta que conecta o cliente à rede (dois sentidos

do tráfego e dois graus de granularidade); dois deles correspondentes às taxas de bytes e pacotes transmitidos deste roteador de borda que conecta cliente e gerador de carga ao roteador de núcleo; e um deles à taxa de bytes recebidos pelo roteador de núcleo deste roteador de borda.

No que se refere a comparação inicialmente proposta entre os resultados obtidos em Pasquini e Stadler (2017) e em Stadler, Pasquini e Fodor (2017) com o serviço de video sob demanda e os aqui encontrados com serviço DASH, os primeiros foram melhores em relação ao erro (NMAE) para a métrica de quadros por segundo, alcançando resultados abaixo de 10%. Porém, para a métrica de áudio *buffers* por segundo, os resultados encontrados neste trabalho foram melhores. Aqui, os melhores resultados não ultrapassaram os 13,5%, enquanto que no trabalho citado, para o mesmo padrão de carga, eles foram superiores a 20%. Contudo, cabe destacar que o trabalho sobre o vídeo sob demanda só analisou essa métrica utilizando todo o conjunto de atributos.

Considerando o tempo de treinamento, os resultados aqui obtidos foram mais satisfatórios, ficando na ordem de milissegundos para ambas as métricas de QoS. Alguns fatores que podem ter influenciado neste resultado foram o menor número de atributos disponíveis e a quantidade menor de amostras no conjunto de treinamento. Quanto à influência de ponto na rede onde os atributos são obtidos, para o serviço DASH, as estatísticas que mais influenciaram os resultados foram coletadas no comutador de acesso ao cliente enquanto que, para o serviço de vídeo sob demanda, foi identificado que dados do comutador de acesso ao servidor seriam os mais significativos.

5.1 Principais Contribuições

A principal contribuição do trabalho é ter demonstrado ser possível utilizar os dados de tráfego agregado da rede para extrair conhecimento sobre qualidade de serviço em um serviço de vídeo sob demanda dinamicamente adaptável. Os resultados obtidos demonstraram que há uma indicação de o ponto onde o cliente se conecta a rede ser o local que contenha as melhores informações sobre as métricas investigadas.

Além disso, os resultados alcançados com relação ao tempo de treinamento dos modelos de predição indicam ser possível o desenvolvimento de mecanismos que gerem informações sobre a qualidade desse serviço sem a necessidade constante de extrair dados dos clientes. Os dados de referência seriam apenas necessários para criar os modelos de predição, os quais poderiam ser atualizados periodicamente.

5.2 Trabalhos Futuros

Após a avaliação dos resultados, confirmou-se que alguns pontos da pesquisa podem ser melhor esclarecidos com a utilização de séries temporais com maior número de amostras.

O uso de mais dados na geração dos modelos de predição pode esclarecer tanto se o erro pode ser melhorado, como se o tempo de treinamento continuará satisfatório, inclusive investigando qual a quantidade de amostras ideal a ser empregada para geração do modelo de predição de uma dada métrica de QoS utilizando-se um determinado algoritmo.

Outra questão identificada é o emprego de mais estatísticas de tráfego agregado disponibilizadas pelo protocolo OpenFlow, como as estatísticas por fluxo. Isso porque novos atributos podem fornecer mais dados que influenciem na acurácia do modelo de predição criado. Também devem ser investigados outros algoritmos de aprendizagem de máquina para verificar se há outros algoritmos que modelem melhor o comportamento do serviço DASH, fornecendo resultados ainda melhores.

Além disso, devem ser investigados outros padrões de carga para ampliar o conhecimento a respeito do próprio comportamento do serviço DASH, inclusive avaliando o seu comportamento quando as estatísticas de tráfego agregado incluem informações de outros serviços além dele. Já a coleta de dados em mais de um cliente simultaneamente poderia esclarecer quais locais da rede contém informações mais genéricas sobre a qualidade do serviço, considerando inclusive a relação da acurácia com o congestionamento de tráfego.

Finalmente, deve ser citado que o ambiente de testes proposto na pesquisa é uma implementação do serviço com escopo simplificado, de modo que devem ser investigadas futuramente situações mais próximas do real. Isso inclui considerar um número maior de servidores e clientes, maior número de roteadores na rede, diferentes pontos de entrada de carga na rede, comunicação via rede sem fio, além dos já citados nos parágrafos anteriores relacionados à questão.

Referências

ABAR, T.; LETAIFA, A. B.; ELASMI, S. Enhancing qoe based on machine learning and dash in sdn networks. In: **2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)**. [S.l.: s.n.], 2018. p. 258–263. <https://doi.org/10.1109/WAINA.2018.00095>.

ADOBE INC. **Adobe HTTP Dynamic Streaming (HDS) Technology Center**. 2019. [Online]. Disponível em: <https://www.adobe.com/devnet/hds.html>. Acesso em: 05/12/2019.

ALTHOS. Mpeg group of pictures - gop. In: _____. **IPTV Dictionary**. Discover-Net Publishing, 2009. [Online]. Disponível em: <http://www.iptvdictionary.com/iptv_dictionary_MPEG_GOP_definition.html>. Acesso em: 20/01/2020.

APPLE INC. **HTTP Live Streaming**. 2019. [Online]. Disponível em: <https://developer.apple.com/documentation/http_live_streaming>. Acesso em: 05/05/2019.

AROUSSI, S.; MELLOUK, A. Statistical evaluation for quality of experience prediction based on quality of service parameters. In: **2016 23rd International Conference on Telecommunications (ICT)**. [S.l.: s.n.], 2016. p. 1–5. <https://doi.org/10.1109/ICT.2016.7500364>.

BATAEV, A. V. Overview of the global e-learning systems market. In: **2017 International Conference "Quality Management,Transport and Information Security, Information Technologies"(IT QM IS)**. [S.l.: s.n.], 2017. p. 640–644. <https://doi.org/10.1109/ITMQIS.2017.8085905>.

BBC BRASIL. **5 formas simples de ganhar dinheiro com o YouTube**. 2017. [Online]. Disponível em: <http://www.bbc.com/portuguese/geral-42465923>. Acesso em: 05/03/2018.

BENTALEB, A.; BEGEN, A. C.; ZIMMERMANN, R. SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking. In: **Proceedings of the 2016 ACM on Multimedia Conference**. New York, NY, USA: ACM, 2016. (MM '16), p. 1296–1305. ISBN 978-1-4503-3603-1. <http://doi.org/10.1145/2964284.2964332>.

BENTALEB, A. et al. SDNHAS: An SDN-enabled architecture to optimize QoE in HTTP adaptive streaming. **IEEE Transactions on Multimedia**, v. 19, n. 10, p. 2136–2151, Oct 2017. ISSN 1520-9210. <https://doi.org/10.1109/TMM.2017.2733344>.

BOYD, T. **Adaptive Streaming**. Bitmovin Inc., 2016. [Online]. Disponível em: <https://bitmovin.com/adaptive-streaming/>. Acesso em: 15/10/2018.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, Oct 2001. ISSN 1573-0565. <http://doi.org/10.1023/A:1010933404324>.

BUCZAK, A. L.; GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. **IEEE Communications Surveys Tutorials**, v. 18, n. 2, p. 1153–1176, Segundo Trimestre 2016. ISSN 1553-877X. <https://doi.org/10.1109/COMST.2015.2494502>.

CANONICAL LTD. **Network Time Protocol daemon and utility programs**. 2019. <https://packages.ubuntu.com/bionic/ntp>. [Online].

CASADO, M.; FOSTER, N.; GUHA, A. Abstractions for software-defined networks. **Commun. ACM**, ACM, New York, NY, USA, v. 57, n. 10, p. 86–95, set. 2014. ISSN 0001-0782. <http://doi.org/10.1145/2661061.2661063>.

CISCO SYSTEMS, INC. **Cisco Visual Networking Index: Forecast and Methodology, 2016–2021**. 2018. [Online]. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html>. Acesso em: 05/03/2018.

COELHO, M. d. S.; MELO, C. A. V. Estratégia de adaptação de fluxo de vídeo baseado em fatores de qoe. In: **XXXIV O Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**. [S.l.: s.n.], 2016. v. 1, p. 1–11.

DESGRANGE, L. **Encode videos for Dynamic Adaptive Streaming over HTTP**. 2017. [Online]. Disponível em: <https://blog.desgrange.net/post/2017/04/17/encode-videos-dynamic-adaptive-streaming-http.html>. Acesso em: 10/11/2018.

FARHADY, H.; LEE, H.; NAKAO, A. Software-defined networking: A survey. **Computer Networks**, v. 81, p. 79 – 95, 2015. ISSN 1389-1286. <https://doi.org/10.1016/j.comnet.2015.02.014>.

FFMPEG. **A complete, cross-platform solution to record, convert and stream audio and video.** 2019. <https://www.ffmpeg.org/>. [Online].

FLOODLIGHT PROJECT. **Floodlight Controller**. 2020. <https://http://www.projectfloodlight.org/floodlight/>. [Online].

GADALETA, M. et al. D-dash: A deep q-learning framework for dash video streaming. **IEEE Transactions on Cognitive Communications and Networking**, v. 3, n. 4, p. 703–718, Dec 2017. ISSN 2372-2045. <https://doi.org/10.1109/TCCN.2017.2755007>.

GEORGOPOULOS, P. et al. Using software defined networking to enhance the delivery of video-on-demand. **Computer Communications**, v. 69, p. 79 – 87, 2015. ISSN 0140-3664. <https://doi.org/10.1016/j.comcom.2015.06.015>.

GPAC. **MP4Box.** 2019. <https://gpac.wp.imt.fr/mp4box/>. [Online].

HANDURUKANDE, S. et al. Magneto approach to qos monitoring. In: **12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops**. [S.l.: s.n.], 2011. p. 209–216. ISSN 1573-0077. <https://doi.org/10.1109/INM.2011.5990693>.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. 2nd. ed. New York, NY, USA: Springer New York Inc., 2009. (Springer Series in Statistics). Disponível em: <http://web.stanford.edu/~hastie/ElemStatLearn/>.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). **ISO/IEC 23009-1:2014 Dynamic Adaptive Streaming over HTTP (DASH)**. 2014.

JAMES, G. et al. **An introduction to statistical learning: with applications in R**. New York, NY: Springer, 2013. (Springer Texts in Statistics).

JARRAYA, Y.; MADI, T.; DEBBABI, M. A survey and a layered taxonomy of software-defined networking. **IEEE Communications Surveys Tutorials**, v. 16, n. 4, p. 1955–1980, Quarto trimestre 2014. ISSN 1553-877X. <http://doi.org/10.1109/COMST.2014.2320094>.

JI, K. et al. Measuring and predicting quality of experience of dash-based video streaming over lte. In: **2016 19th International Symposium on Wireless Personal Multimedia Communications (WPMC)**. [S.l.: s.n.], 2016. p. 102–107. ISSN 1882-5621.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015. ISSN 0036-8075. <https://doi.org/10.1126/science.aaa8415>.

KARAKUS, M.; DURRESI, A. Quality of service (QoS) in software defined networking (SDN): A survey. **Journal of Network and Computer Applications**, v. 80, p. 200 – 218, 2017. ISSN 1084-8045. <https://doi.org/10.1016/j.jnca.2016.12.019>.

KARHOFF, A. **Social Media Video Content is About to Explode**. American Marketing Association, 2017. [Online]. Disponível em: <https://www.ama.org/publications/MarketingNews/Pages/video-content-about-to-explode.aspx>. Acesso em: 05/03/2018.

KATSARAKIS, M. et al. Towards a causal analysis of video qoe from network and application qos. In: **Proceedings of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks**. New York, NY, USA: ACM, 2016. (Internet-QoE '16), p. 31–36. ISBN 978-1-4503-4425-8.

KAZAKOV, V. Mpeg-dash: Gop. In: _____. **KVS Software: Technical blogging and my projects**. [s.n.], 2015. [Online]. Disponível em: <https://kvssoft.wordpress.com/2015/01/28/mpeg-dash-gop/>. Acesso em: 20/01/2020.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219. <https://doi.org/10.1109/JPROC.2014.2371999>.

KUA, J.; ARMITAGE, G.; BRANCH, P. A survey of rate adaptation techniques for dynamic adaptive streaming over http. **IEEE Communications Surveys Tutorials**, v. 19, n. 3, p. 1842–1866, Terceiro trimestre 2017. ISSN 2373-745X. <https://doi.org/10.1109/COMST.2017.2685630>.

LACERDA, M. C. C. **Configuração de serviço DASH para teste**. 2018. <https://github.com/mcalasans/dash_necos_ufu/wiki>. [Online].

LEDERER, S. **Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length**. Bitmovin Inc., 2015. [Online]. Disponível em: <https://bitmovin.com/mpeg-dash-hls-segment-length/>. Acesso em: 24/01/2020.

LIAW, A.; WIENER, M. Classification and Regression by randomForest. **R News**, v. 2, n. 3, p. 18–22, 2002. Disponível em: <http://CRAN.R-project.org/doc/Rnews/>.

MANOEL, E. T. M. **Codificação de vídeo H.264: estudo de codificação mista de macroblocos.** 2007. Dissertação(Mestrado em Engenharia Elétrica), Universidade Federal de Santa Catarina.

MARSLAND, S. **Machine Learning - An Algorithmic Perspective.** 1st. ed. [S.l.]: CRC Press, 2009. (Chapman and Hall / CRC machine learning and pattern recognition series).

MASOUDI, R.; GHAFFARI, A. Software defined networks: A survey. **Journal of Network and Computer Applications**, v. 67, p. 1 – 25, 2016. ISSN 1084-8045. <https://doi.org/10.1016/j.jnca.2016.03.016>.

MATOS, J. R. F. **Construção de um serviço conteinerizado de vídeo sob demanda baseado em DASH para experimentação e coleta de métricas de desempenho**. 2019. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação), Universidade Federal de Uberlândia, Uberlândia.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. <http://doi.org/10.1145/1355734.1355746>.

MICHALSKI, R.; CARBONELL, J.; MITCHELL, T. An overview of machine learning. In: **Machine Learning: An Artificial Intelligence Approach**. [S.l.]: Springer Berlin Heidelberg, 1983, (Symbolic Computation). cap. 2. ISBN 9783662124055. <https://doi.org/10.1007/978-3-662-12405-5>.

MICROSOFT CORPORATION. [**MS-SSTR**] **Smooth Streaming Protocol**. 2019. [Online]. Disponível em: <https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sstr/8383f27f-7efe-4c60-832a-387274457251>. Acesso em: 05/12/2019.

MILLS, D. et al. **Network Time Protocol Version 4: Protocol and Algorithms Specification**. [S.l.], 2010. Disponível em: <https://www.rfc-editor.org/rfc/rfc5905>.

MORADI, F.; STADLER, R.; JOHNSSON, A. Performance prediction in dynamic clouds using transfer learning. In: **2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2019. p. 242–250. ISSN 1573-0077.

MORI, S.; BANDAI, M. Qoe-aware quality selection method for adaptive video streaming with scalable video coding. In: **2018 IEEE International Conference on Consumer Electronics (ICCE)**. [S.l.: s.n.], 2018. p. 1–4. ISSN 2158-4001. <https://doi.org/10.1109/ICCE.2018.8326093>.

MUSHTAQ, M. S.; AUGUSTIN, B.; MELLOUK, A. Empirical study based on machine learning approach to assess the qos/qoe correlation. In: **2012 17th European Conference on Networks and Optical Communications**. [S.l.: s.n.], 2012. p. 1–7. <https://doi.org/10.1109/NOC.2012.6249939>.

OPEN NETWORKING FOUNDATION (ONF). **Software-Defined Networking: The New Norm for Networks**. 2012. White Paper. [Online]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf/>. Acesso em: 03/02/2019.

_____. **SDN Architecture, version 1.0**. 2014. [Online]. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf>. Acesso em: 03/02/2019.

_____. **OpenFlow Switch Specification, version 1.3.5**. 2015. White Paper. [Online]. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf>. Acesso em: 03/02/2019.

_____. **SDN Architecture, version 1.1**. 2016. [Online]. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf>. Acesso em: 03/02/2019.

_____. 2019. <https://www.opennetworking.org/>.

OPENWRT PROJECT. **OpenWrt Wireless Freedom**. 2020. <https://openwrt.org/>. [Online].

ORACLE. **VirtualBox**. 2020. <https://www.virtualbox.org/>. [Online].

OZER, J. **The Secret to Encoding High Quality Web Video: Tutorial**. [S.l.]: Tubular Insights & Tubular Labs, Inc., 2011. <https://tubularinsights.com/secret-encoding-web-video/>. [Online]. Acesso em: 16/01/2020.

PASQUINI, R.; STADLER, R. Learning end-to-end application QoS from OpenFlow switch statistics. In: **2017 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2017. p. 1–9. <https://doi.org/10.1109/NETSOFT.2017.8004198>.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PETRANGELI, S. et al. A machine learning-based framework for preventing video freezes in HTTP adaptive streaming. **Journal of Network and Computer Applications**, v. 94, p. 78 – 92, 2017. ISSN 1084-8045. <https://doi.org/10.1016/j.jnca.2017.07.009>.

REZENDE, P. H. A. **Extensões na arquitetura SDN para o provisionamento de QoS através do monitoramento e uso de múltiplos caminhos**. 2016. Dissertação(Mestrado em Ciência da Computação), Universidade Federal de Uberlândia. Disponível em: <http://doi.org/10.14393/ufu.di.2016.59>.

SALGADO, E. **Como a revolução do streaming mudou as TVs e Hollywood**. Revista Exame, 2017. Disponível em: <https://exame.abril.com.br/revista-exame/como-a-revolucao-do-streaming-mudou-os-negocios-das-tvs/>. Acesso em: 05/03/2018.

SAMANI, F. S.; STADLER, R. Predicting distributions of service metrics using neural networks. In: **2018 14th International Conference on Network and Service Management (CNSM)**. [S.l.: s.n.], 2018. p. 45–53. ISSN 2165-9605.

SAMMUT, C.; WEBB, G. I. **Encyclopedia of Machine Learning**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. ISBN 0387307680, 9780387307688.

SANAEI, M.; MOSTAFAVI, S. Multimedia delivery techniques over software-defined networks: A survey. In: **2019 5th International Conference on Web Research (ICWR)**. [S.l.: s.n.], 2019. p. 105–110. <https://doi.org/10.1109/ICWR.2019.8765278>.

SDNCENTRAL, L. **OpenFlow and SDN – State of the Union**. 2016. Special Report. [Online]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/special-reports/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf>. Acesso em: 03/02/2019.

SEUFERT, M. et al. A survey on quality of experience of http adaptive streaming. **IEEE Communications Surveys Tutorials**, v. 17, n. 1, p. 469–492, Primeiro trimestre 2015. ISSN 2373-745X.

SHAFIQ, M. et al. Network traffic classification techniques and comparative analysis using machine learning algorithms. In: **2016 2nd IEEE International Conference on Computer and Communications (ICCC)**. [S.l.: s.n.], 2016. p. 2451–2455. <https://doi.org/10.1109/CompComm.2016.7925139>.

SODAGAR, I. The MPEG-DASH standard for multimedia streaming over the internet. **IEEE MultiMedia**, v. 18, n. 4, p. 62–67, April 2011. ISSN 1070-986X. <https://doi.org/10.1109/MMUL.2011.71>.

SON, J.; BUYYA, R. A taxonomy of software-defined networking (SDN)-enabled cloud computing. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 51, n. 3, p. 59:1–59:36, maio 2018. ISSN 0360-0300. <http://doi.org/10.1145/3190617>.

STADLER, R.; PASQUINI, R.; FODOR, V. Learning from network device statistics. **Journal of Network and Systems Management**, v. 25, n. 4, p. 672–698, Oct 2017. ISSN 1573-7705. <https://doi.org/10.1007/s10922-017-9426-z>.

STOCKHAMMER, T. Dynamic adaptive streaming over http –: Standards and design principles. In: **Proceedings of the Second Annual ACM Conference on Multimedia Systems**. New York, NY, USA: ACM, 2011. (MMSys '11), p. 133–144. ISBN 978-1-4503-0518-1. <https://doi.org/10.1145/1943552.1943572>.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introdução à Mineração de dados**. [S.l.]: Ciência Moderna, 2009.

THE APACHE SOFTWARE FOUNDATION. **Apache HTTP Server Project**. 2020. <https://httpd.apache.org/>. [Online]. Acesso em: 27/01/2020.

The Linux Foundation. **Open vSwitch Project (OVS).** 2019. <http://www.openvswitch.org/>. [Online].

TP-LINK. **Roteador TP-Link Modelo TL-WR1043ND**. 2019. <https://www.tp-link.com/br/products/details/cat-9_TL-WR1043ND.html>. [Online].

UZAKGIDER, T.; CETINKAYA, C.; SAYIT, M. Learning-based approach for layered adaptive video streaming over SDN. **Computer Networks**, v. 92, p. 357 – 368, 2015. ISSN 1389-1286. <https://doi.org/10.1016/j.comnet.2015.09.027>.

VASILEV, V. et al. Predicting qoe factors with machine learning. In: **2018 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2018. p. 1–6. ISSN 1938-1883.

VEGA, M. T. et al. A review of predictive quality of experience management in video streaming services. **IEEE Transactions on Broadcasting**, v. 64, n. 2, p. 432–445, June 2018. ISSN 1557-9611. <https://doi.org/10.1109/TBC.2018.2822869>.

VIDEOLAN. Modules - demux - adaptive. In: **VLC Media Player Repository on GitHub**. 2019. [Online]. Disponível em: <https://github.com/videolan/vlc/tree/master/modules/demux/adaptive/logic>. Acesso em: 24/01/2020.

_____. VLC Command-line Help. In: **VideoLAN's Wiki**. 2019. [Online]. Disponível em: <https://wiki.videolan.org/VLC_command-line_help/>. Acesso em: 18/12/2019.

_____. **VLC Media Player**. 2019. <https://www.videolan.org/index.pt-BR.html>. [Online].

WANG, M. et al. Machine learning for networking: Workflow, advances and opportunities. **IEEE Network**, PP, n. 99, p. 1–8, 2017. ISSN 0890-8044. <https://doi.org/10.1109/MNET.2017.1700200>.

WIKIMEDIA. Content delivery network. In: _____. **Wikipédia: a enciclopédia livre**. Wikimedia, 2019. [Online]. Disponível em: <https://en.wikipedia.org/w/index.php?title=Content_delivery_network&oldid=930535477>. Acesso em: 05/12/2019.

_____. Group of pictures. In: _____. **Wikipédia: a enciclopédia livre**. Wikimedia, 2019. [Online]. Disponível em: <https://en.wikipedia.org/wiki/Group_of_pictures>. Acesso em: 15/01/2020.

_____. Vídeo compression picture types. In: _____. **Wikipédia: a enciclopédia livre**. Wikimedia, 2019. [Online]. Disponível em: <https://en.wikipedia.org/wiki/Video_compression_picture_types>. Acesso em: 15/01/2020.

WOWZA MEDIA SYSTEMS, LLC. **Optimize the video bitrate for Wowza Streaming Engine**. 2018. <https://www.wowza.com/docs/how-to-optimize-the-video-bitrate>. [Online]. Acesso em: 16/11/2018.

XIA, W. et al. A survey on software-defined networking. **IEEE Communications Surveys Tutorials**, v. 17, n. 1, p. 27–51, Primeiro trimestre 2015. ISSN 1553-877X. <https://doi.org/10.1109/COMST.2014.2330903>.

YANGGRATOKE, R. et al. Predicting service metrics for cluster-based services using real-time analytics. In: **2015 11th International Conference on Network and Service Management (CNSM)**. [S.l.: s.n.], 2015. p. 135–143. <http://doi.org/10.1109/CNSM.2015.7367349>.

YOUTUBE, LLC. **YouTube: Broadcasting Yourself**. 2018. <https://www.youtube.com>. [Online].

ZHAO, Y. et al. A survey of networking applications applying the software defined networking concept based on machine learning. **IEEE Access**, v. 7, p. 95397–95417, 2019. <https://doi.org/10.1109/ACCESS.2019.2928564>.