Classificação de Tráfego para Gerenciamento de Largura de Banda em Redes Definidas por Software

Nilton J. Mocelin Jr.[1], **Rafael S. Parpinelli**[1], **Adriano Fiorese**[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCAP)
Departamento de Ciência da Computação (DCC)
Universidade do Estado de Santa Catarina (UDESC)
CEP: 89.219-710 – Joinville / SC – Brasil

nilton.junior@edu.udesc.br

{rafael.parpinelli, adriano.fiorse}@udesc.br

Abstract. *Applications have different requirements in terms of bandwidth and delay to deliver the expected quality of service. For this and other reasons, new paradigms have emerged, such as software-defined networks, which allow the development of new applications for dynamic programming of routing devices in the network through a device called controller. In this work, five machine learning models for classifying network traffic are tested, which when implemented in the controller make it possible to discover the types of applications and their quality of service requirements. The obtained results demonstrate that the Random Forest model has an excellent performance in the classification of network traffic using few packets and achieves very high accuracy margins. On the other hand, it was noted that machine learning models based on streaming data are very sensitive to the database and the amount of packets used in the classification.*

Resumo. *As aplicações têm requisitos diferentes em termos de largura de banda e atraso para entregar a qualidade de serviço esperada. Por esse e outros motivos, surgiram novos paradigmas como as redes definidas por software, que permitem o desenvolvimento de novas aplicações para programação dinâmica de dispositivos de roteamento na rede por meio de um dispositivo denominado controlador. Neste trabalho, são testados cinco modelos de machine learning para classificação do tráfego de rede, que quando implementados no controlador possibilitem descobrir os tipos de aplicações e seus requisitos de qualidade de serviço. Os resultados obtidos demonstram que o modelo Random Forest tem um excelente desempenho na classificação do tráfego de rede usando poucos pacotes e alcança margens de precisão muito altas. Por outro lado, notou-se que os modelos de aprendizado de máquina baseados em streaming de dados são muito sensíveis ao banco de dados e à quantidade de pacotes utilizados na classificação.*

1. Introdução

O termo qualidade de serviço (QoS) pode ser definido pelas características de um serviço de telecomunicação que garantem a capacidade de satisfazer as necessidades do usuário

do serviço, tanto dos requisitos anunciados quanto dos implícitos [ITU 2001]. Quando esses recursos não são administrados, não se pode garantir a sua distribuição apropriada, fazendo com que alguns serviços sejam prejudicados [Karakus and Durresi 2017].

Prover gerenciamento dos recursos de rede, em especial a largura de banda, também traz diversos benefícios para administradores de rede [Huawei 2022]. Pode se garantir que os principais serviços não são afetados quando a rede está sobrecarregada. Além disso, o gerenciamento de largura de banda ajuda os administradores de rede a alocar adequadamente os recursos de rede disponíveis e evitar gastos com infraestrutura de forma desnecessária.

As redes convencionais não implementam tratamento para garantir os requisitos de qualidade de serviço dos fluxos de aplicações, causando a disputa dos recursos disponíveis na rede. A infraestrutura de rede da Internet atual não se compromete com garantias de entrega priorizada nem de reserva de recursos, apenas implementa um sistema de entrega *best-effort*. A Internet é formada por grupos de sistemas independentes [ITU 2001]. Cada sistema autônomo é responsável pelo gerenciamento de tráfego e uso de protocolos de roteamento dentro de seu domínio. Por este motivo, garantir QoS é uma tarefa complexa em ambientes baseados na Internet. No entanto, o interesse em aplicações cada vez mais modernas e o surgimento de novos paradigmas de rede, como SDN, buscam mudar esse panorama.

Assim, para lidar com o provisionamento dinâmico de recursos e as necessidades de entrega prioritária de aplicações diversas, existem diversos *frameworks* como o *Flowing with Priority in Software Defined Network (SDN)* (FLOWPRI-SDN) [Mocelin Júnior and Fiorese 2023]. Este controlador consiste em uma implementação do controlador SDN Ryu que busca prover reserva hierárquica de recursos e priorização de tráfego para fluxos que declarem seus requisitos por meio de um contrato de QoS. Deste modo, o *framework* é capaz de criar regras de encaminhamento específicas para tratar os fluxos com QoS dentro de seu domínio, além de encontrar outros *frameworks* em outros domínios e distribuir os contratos de QoS, sendo compatível com redes ou domínios que não o suportam. No entanto, o FLOWPRI-SDN não é capaz de identificar os fluxos e seus requisitos dentro de seu domínio sem que o *host* emissor anuncie essas informações.

Deste modo, este artigo busca levantar uma comparação entre modelos de *machine learning* para classificação de tráfego de rede em classes de serviços. Modelos de aprendizado como *Random Forest* são baseados em árvores de decisão e são utilizados para classificar blocos de dados. Por outro lado, modelos *Hoeffding Tree* (também uma variação de árvores de decisão) possuem foco em dados do tipo *streaming* (contínuo). Os dados de pacotes de fluxos de aplicações na rede podem ser interpretados em blocos ou em *streaming*. Além disso, outro ponto a ser levado em consideração é sobre quantos pacotes são necessários para que os modelos de aprendizado alcancem taxas altas de precisão na classificação, focando em quantias de 10 e 30 pacotes. O Objetivo é analisar modelos de classificação para que estes possam ser incluídos no FLOWPRI-SDN no processo de classificação de tráfego de rede e identificação de requisitos de QoS.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta trabalhos relacionados e abordagens utilizadas na classificação de tráfego de rede. A metodologia da obtenção e tratamento das bases de dados é apresentada na Seção 3, os

modelos de *machine learning* utilizados são brevemente apresentados. Os experimentos e resultados são apresentados na Seção 4, finalizando com as conclusões do trabalho na Seção 5.

2. Trabalhos Relacionados

A classificação baseada em porta foi uma das técnicas mais usadas no passado, porque muitos aplicativos usavam números de porta fixos atribuídos pela *Internet Assigned Numbers Authority* (IANA). No entanto, com o tempo, algumas limitações dessa abordagem começaram a aparecer. Inúmeros aplicativos não possuem números de porta registrados, e muitos deles usam mecanismos dinâmicos de negociação de porta para se esconder de *firewalls* e ferramentas de segurança de rede. Além disso, o uso de criptografia de camada IP acabaram impossibilitando a descoberta dos números de porta originais [Amaral et al. 2016a].

O trabalho apresentado em [Raikar et al. 2020] avalia os modelos *Support Vector Machine* (SVM), *GaussianNB* e *Nearest Centroid* para identificar aplicações do tipo HTTP, e-mail, *streaming* e VLC em redes SDN. No entanto, utilizam todo o fluxo para classificação e possuem objetivo similar de adaptar em um *framework* de controle.

O trabalho apresentado em [Amaral et al. 2016b] também busca classificar tráfego utilizando *machine learning* baseado em *features*. Dentre as classes de tráfego estão HTTP, *YouTube*, Skype e *Facebook*. Os modelos utilizados foram *Random Forest*, *Stochastic Gradient Boosting Accuracy* e *Extreme Gradient Boosting Accuracy*, onde *Random Forest* teve os melhores resultados. O objetivo também era utilizar em um controlador SDN.

Os trabalhos relacionados apresentam um pouco do contexto de classificação de tráfego. Otimizar um modelo e implementar sobre um controlador SDN é o direcionamento seguido por muitos, no entanto, poucos definem seus objetivos e utilizam métodos de geração de bases de dados ajustadas em sub fluxos (fluxos válidos) ou definem estratégias de uso para seus modelos. Além disso, os trabalhos utilizam todo o fluxo ou um excesso de pacotes para a classificação, o que é um impeditivo para identificar fluxos de curta duração, além de degradar o desempenho da rede. Da mesma forma, poucos trabalhos comparam métodos de classificação para dados do tipo *batch* com métodos voltados para dados do tipo *streaming*, uma vez que os pacotes na rede podem ser tratados das duas formas.

3. Metodologia de Desenvolvimento

Em uma rede baseada no protocolo TCP/IP, os pacotes são responsáveis por realizar a comunicação entre dois dispositivos [Zhou et al. 2018]. Um dispositivo envia pacotes para outro que incluem endereços de origem e destino, bem como portas de origem e destino que servem para identificá-los na rede. Quando o dispositivo de destino adquire os pacotes, esses são decodificados e agregados para obter os dados encaminhados. Desta forma, uma definição de fluxos de dados pode ser dada pela quíntupla: endereço IP de origem e destino, portas de origem e destine tipo de protocolo. Assim, a coleta de fluxos de rede consiste em obter pacotes que possuem as mesmas características que passam por um caminho monitorado por um período específico de tempo [Zhou et al. 2018].

As bases de dados para classificação de tráfego de rede geralmente podem ser encontradas na Internet de forma processada, quando uma tabela com *features* sobre informações estatísticas e comportamentais de um ou vários pacotes são rotuladas em uma classe específica (*label*), ou na forma bruta de captura de pacotes (arquivo *.pcap*). As bases já processadas, possuem geralmente dados que agrupam um fluxo completo (do primeiro pacote ao último) e neste caso podem não representar um cenário de rede simulado, no qual, apenas alguns primeiros pacotes de um fluxo podem ser capturados, caso contrário a sobrecarga associada pode diminuir o desempenho da rede.

A base de dados ISCX-VPN-NonVPN-2016 consiste em capturas de pacotes de rede de tráfego real com muita diversidade de tipos de aplicações rotuladas [UNB 2016]. Essa base foi gerada a partir da captura de tráfego de dois computadores, chamados de Bob e Alice, que foram usados para acessarem diversos serviços como Skype, Facebook e entre outros, além de tráfego sobre redes virtuais privadas (*Virtual Private Network* - VPN). A Figura 1 representa como os pacotes são armazenados em formato de captura por meio das ferramentas *Wireshark* e *tcpdump*.

Figura 1. Inspeção de pacotes capturados utilizando a ferramenta *wireshark*.

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	131.202.240.87	64.12.24.167	TLSv1.1	171	Application Data
2	0.054789	64.12.24.167	131.202.240.87	TCP	60	443 → 13385 [ACK] Seq=1 Ack=118 Win=16384 Len=0
3	0.054832	131.202.240.87	64.12.24.167	TLSv1.1	155	Application Data
4	0.109924	64.12.24.167	131.202.240.87	TLSv1.1	315	Application Data
5	0.110444	64.12.104.85	131.202.240.87	TLSv1.2	331	Application Data
6	0.110472	64.12.24.167	131.202.240.87	TLSv1.1	155	Application Data
7	0.110497	131.202.240.87	64.12.24.167	TCP	54	13385 → 443 [ACK] Seq=219 Ack=363 Win=63729 Len=0
8	0.126831	64.12.24.167	131.202.240.87	TLSv1.1	373	Application Data, Application Data, Application Data
9	0.129959	131.202.240.87	64.12.104.85	TCP	54	1254 → 443 [ACK] Seq=1 Ack=278 Win=64055 Len=0
10	0.151426	131.202.240.87	64.12.24.167	TCP	54	13385 → 443 [ACK] Seq=219 Ack=682 Win=64800 Len=0
11	0.279966	131.202.240.87	131.202.244.3	DNS	75	Standard query 0x0034 A au.pool.ntp.org
12	0.432986	131.202.244.3	131.202.240.87	DNS	553	Standard query response 0x0034 A au.pool.ntp.org A 20
13	0.433385	131.202.240.87	203.23.237.200	NTP	90	NTP Version 3, client
14	0.711308	203.23.237.200	131.202.240.87	NTP	90	NTP Version 3, server
15	6.598172	131.202.240.87	178.237.19.228	SSL	60	Continuation Data
16	6.599033	131.202.240.87	178.237.19.103	SSL	60	Continuation Data

Fonte: Próprio Autor, 2023.

Neste trabalho, apenas uma porção da base de dados gerada foi utilizada. A Tabela 1, apresenta os arquivos de captura utilizados e a quantidade arredondada de pacotes identificada para cada tipo de aplicação. No caso da informação referente a classe de serviço, essa foi extraída por meio do rótulo associado no nome do arquivo e da natureza da aplicação mediante a pesquisa em seus *web sites*.

A Tabela 2 sintetiza um estudo das características de largura de banda das classes de serviços. A maioria das aplicações não fornece de forma clara seus requisitos, no entanto, existem outros meios de identificar essas informações. Uma das formas é analisando o tráfego de rede de cada aplicação. Outra forma é se basear em estudos genéricos de classes de aplicações, assim como apresentado em [Yan Chen 2004]. Por isso, os rótulos definidos para cada classe representam a maior largura de banda especificada pelos serviços que compõe a base de dados.

3.1. Processamento de Dados

Para montar a base de dados desejada a partir dos arquivos *.pcap* foi utilizada uma biblioteca de manipulação de arquivos chamada Scapy [Biondi 2023]. Essa biblioteca permite implementar códigos na linguagem *python* de modo para extrair pacotes, medir e armazenar em outro formato se necessário.

Tabela 1. Divisão dos fluxos e pacotes na base de dados.

Classe de Serviço	Aplicações	Fluxos	Pkts
Áudio Tempo-Real	VoipBuster,Skype, Hangouts	5588	796.964
Áudio Estático	Spotify	176	26.812
Vídeo Tempo-Real	Skype,Hangouts	580	978.291
Vídeo Estático	YouTube,Netflix	657	370.356
Chat	Facebook,Hangouts, ICQ,AIM,Skype, GmailChat	4363	189.869
Dados Download	Bittorrent,ftps,sftp, scp	758	589.583
Dados Upload	ftps,sftp	135	200.425
E-mail	Gmail	1400	96.660

Fonte: Próprio Autor, 2023.

Utilizando a ferramenta, filtramos dos arquivos de captura de cada aplicação os fluxos de dados individuais, ou seja, o conjunto de pacotes em sequência que possuem a mesma quíntupla de endereços IP de origem e destino, porta de origem, porta de destino e protocolo, armazenando em um novo arquivo $.pcap$. Isso é necessário para identificar um fluxo em uma classe. Durante a filtragem dos fluxos foram identificados alguns fluxos de protocolos não relacionados com a qualidade das aplicações desejadas, como DNS, LMNS e ARP, que foram removidos para não influenciarem nas medidas das *features*. Esses tipos de fluxos são tratados como fluxos sem requisitos de QoS pelo FLOWPRI-SDN.

Após extrair os fluxos de aplicações em arquivos separados, identificamos os sub fluxos dentro desses fluxos que são considerados ativos em um cenário gerenciado pelo controlador FLOWPRI-SDN, ou seja, o tempo entre chegada dos pacotes não ultrapassa 10s. Para contextualizar, fluxos de dados que entram em um domínio gerenciado por um controlador SDN, são encaminhados por dispositivos de rede para tratamento no controlador (quando ocorre a coleta de pacotes). O FLOWPRI-SDN trata esses fluxos definindo regras de encaminhamento com QoS, de modo que as regras expirem caso um fluxo fique inativo por mais de 10s, necessitando uma nova aceitação do fluxo por parte do controlador e criação de novas regras.

As regras de fluxo do FLOWPRI-SDN, gerenciam a largura de banda das portas dos *switches* do domínio, realizando alocação e reserva de largura de banda para os fluxos identificados. A identificação desses fluxos deve ser automatizada por um método de aprendizado. A Figura 2, mostra um domínio gerenciado pelo controlador FLOWPRI-SDN, onde os *switches* encaminham os pacotes dos fluxos conforme as regras criadas por ele. Deste modo, para extrair os sub fluxos válidos neste cenário são extraídos os pacotes dos fluxos que possuem tempo entre chegadas (tempo decorrido entre um pacote e outro)

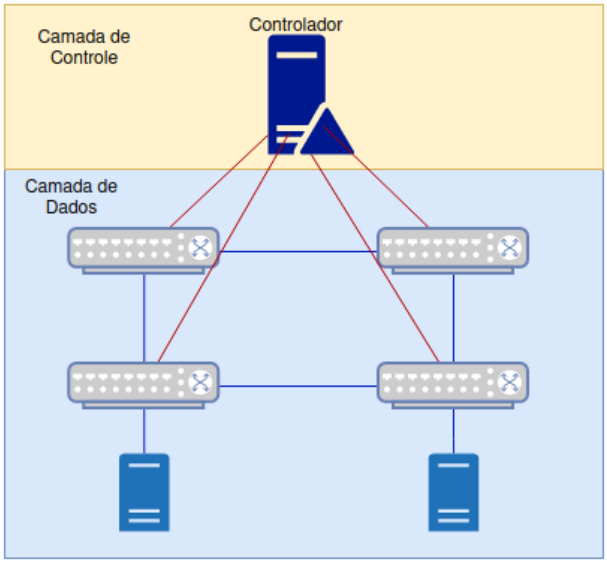
Tabela 2. Largura de banda das aplicações e rótulo definidos.

Classe de Serviço	Aplicação	Largura de Banda	Rótulo
Áudio Tempo-Real	VoipBuster, Skype, Hangouts	55-90kbps (codec) 100kbps (skype)	ar-100kbps
Áudio Estático	Spotify	1-2Mbps	ae-2Mbps
Vídeo Tempo-Real	Skype, Hangouts	HD 1.5Mbps (skype) SD 300-500kbps (skype)	vr-1.5Mbps
Vídeo Estático	YouTube, Netflix	(Youtube) 1080p,30fps = 5 Mbps 720p,30fps = 2.5 Mbps 360p,30fps = 700kbps	ve-5Mbps
Chat	Facebook,Hangouts, ICQ,AIM,Skype, GmailChat	chat em geral 24-128kbps	chat-128kbps
Dados Download	Bittorrent, ftps, scp	Máximo (10Mbps)	down-max
Dados Upload	ftps, sftp	Máximo (10Mbps)	up-max
E-mail	Gmail	10kbps	email-10kbps

Fonte: [Yan Chen 2004, Microsoft 2023, Google 2023].

menor que 10s e armazenados em outro arquivo *.pcap* dos sub fluxos. Essa técnica é chamada de *Sub-flow packet sampling* [Zander et al. 2012].

Figura 2. Domínio gerenciado pelo *framework* FLOWPRI-SDN.



Fonte: Próprio Autor, 2023.

3.2. Extração das *Features*

A partir da separação dos sub fluxos válidos, temos os arquivos *.pcap* ajustados de cada aplicação e é possível montar as bases de *features* para treinar nos modelos de *machine learning* extraindo as informações dos pacotes por meio da biblioteca *scapy*. Buscando avaliar a influência da quantidade de pacotes utilizados e os modelos de classificação, foi

proposto criar quatro bases de dados de *features*, duas para dados do tipo *batch* e duas para dados do tipo *streaming*.

As bases do tipo *batch* foram criadas a partir da extração das *features* de cada 10 pacotes de um sub fluxo, armazenando cada resultado em uma linha da tabela de *features* de cada aplicação, o mesmo para a base de 30 pacotes. Já para criar as bases para dados do tipo *streaming*, cada pacote é avaliado dois-a-dois, resultando em 10 linhas na tabela quando avaliando 10 pacotes (ou 30 linhas para 30 pacotes). Após a geração inicial das bases de *features* de cada aplicação, as bases são agrupadas por classe de serviço, balanceadas e os blocos com menos de 10 pacotes (ou 30) são descartados. Para bases de *streaming*, ainda os blocos de *features* do fluxo são randomizados respeitando a ordem de precedência das entradas.

A Tabela 3 apresenta as *features* calculadas em cada linha da base de dados para o *batch* ou *streaming* de pacotes. Dentre as informações calculadas estão o tamanho do maior pacote em *bytes*, a duração do bloco, soma dos tamanhos dos pacotes dos blocos, tempo entre chegadas dos pacotes (IAT), tamanho de *payload* (carga de dados) dos pacotes) e outras informações relacionadas aos campos de cabeçalho do protocolo TCP.

Tabela 3. Tabela de *features* calculadas.

proto	duracao	pkt_soma_tam
pkt_maior_tam	pkt_menor_tam	pkts/seg
1quartil_pkt_tam	3quartil_pkt_tam	pkt_tam_media
pkt_tam_mediana	pkt_tam_std	pkts_tam_menor_media
pkts_tam_maior_media	payload_soma	payload_menor
payload_maior	payload_media	payload_std
payload_menor_128	payload_entre128_1024	payload_maior_1024
1quartil_payload_tam	3quartil_payload_tam	IAT_menor
IAT_maior	IAT_soma	IAT_media
IAT_maiores_media	IAT_menores_media	IAT_std
1quartil_IAT	3quartil_IAT	tcp_push_flags
tcp_urg_flags	tcp_syn_flags	tcp_fin_flags
tcp_rst_flags	tcp_ack_flags	tcp_flags
tcp_windowsize_soma	tcp_windowsize_media	tcp_windowsize_std
header_tam_soma	header_tam_media	header_tam_std

Fonte: Próprio Autor, 2023.

Devido à diferença das escalas dos valores coletados das *features*, as *features* com valores maiores possuem maior influência na classificação em determinados modelos. Assim, cada *feature* foi normalizada em um tamanho máximo calculado. Da mesma forma, para evitar tendências nos modelos em que se beneficie de uma classe, as bases de *features* foram balanceadas, respeitando a classe de menor quantidade de dados de *features*:

- *Batch* 10pkts: 1770 dados por classe (total: 14160).
- *Batch* 30pkts: 555 dados por classe (total: 4440).
- *Streams* 10pkts: 18035 dados por classe (total: 144280).
- *Streams* 30pkts: 17763 dados por classe (total: 142110).

Assim, a Figura 3 mostra um exemplo de base de *features* gerada para dados do tipo *batch* de 10 pacotes. Na Figura 4, é apresentado um exemplo de base de dados do tipo *streaming* para 10 pacotes. Como se pode perceber, a base de dados para o tipo *streaming* leva em consideração a ordem de chegada dos pacotes dos fluxos. Desta forma, se assemelha com o comportamento dos fluxos de pacotes em um cenário simulado.

Figura 3. Exemplo base de *features* - *batch* 10pkts.

pkts	class	proto	dport	lbanda	duracao	pkt_soma_tam	pkt_maior_tam	pkt_menor_tam	pkts_por_seg	1stq_pkt_tam
10	ae-2Mbps	1	0.001220721752	0.07956600362	0.00000553	0.044	0.044	0.044	1.808318264	0.044
10	up-max	1	0.0003356984817	0.0009310944188	0.00043533	0.04053333333	0.08133333333	0.036	0.02297107941	0.036
10	ae-2Mbps	1	0.001220721752	0.0352	0.0000125	0.044	0.044	0.044	0.8	0.044
10	chat-128kbps	1	0.08574044404	0.0002174232273	0.01745597	0.3795333333	0.9393333333	0.04	0.0005728699121	0.054
10	chat-128kbps	1	0.0067597467	0.0001849079807	0.03174552	0.587	1.138	0.036	0.0003150050779	0.036
10	ve-5Mbps	1	0.916395819	3.225531915	0.00000705	2.274	3.612	0.936	1.418439716	0.936
10	down-max	1	0.9024032959	0.3815937149	0.00002376	0.9066666667	0.9066666667	0.9066666667	0.4208754209	0.9066666667
10	email-10kbps	1	0.779064622	0.1092501105	0.00008299	0.9066666667	0.9066666667	0.9066666667	0.1204964454	0.9066666667
10	email-10kbps	1	0.01515220874	7.56E-06	0.06684026	0.05053333333	0.06333333333	0.03466666667	0.0001496104294	0.03466666667
10	ar-100kbps	1	0.7127641718	5.49E-05	0.01205272	0.0662	0.236	0.03466666667	0.0008296882363	0.03466666667
10	ve-5Mbps	1	0.916395819	0.6054985094	0.00003019	1.828	3.612	0.936	0.3312355084	0.936
10	down-max	1	0.9024032959	0.03928365107	0.0002308	0.9066666667	0.9066666667	0.9066666667	0.04332755633	0.9066666667
10	email-10kbps	1	0.779064622	0.000179254666	0.00908577	0.1628666667	0.3966666667	0.03466666667	0.001100622182	0.03466666667
10	down-max	1	0.6642252232	0.9717756342	0.00000933	0.9066666667	0.9066666667	0.9066666667	1.071811361	0.9066666667
10	ve-5Mbps	1	0.916395819	0.3888576252	0.00003554	1.382	2.72	0.936	0.2813731007	0.936

Fonte: Próprio Autor, 2023.

Figura 4. Exemplo base de *features* - *streaming* 10pkts.

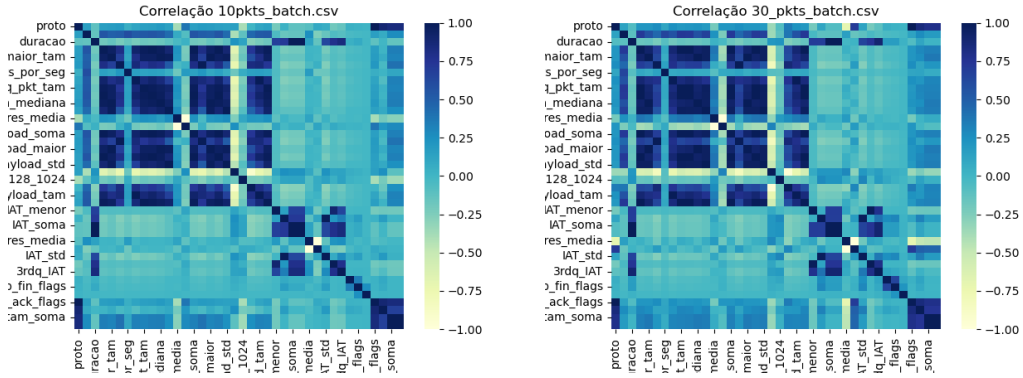
cont	pkts	class	proto	dport	lbanda	duracao	pkt_soma_tam	pkt_maior_tam	pkt_menor_tam	pkts_por_seg
0	1	ae-2Mbps	1	0.001220721752	0	0	0.2686666667	0.2686666667	0.2686666667	0
3600	1	ar-100kbps	0	0.6145265888	0	0	0.09	0.09	0.09	0
14470	1	up-max	1	0.0003356984817	0	0	0.036	0.036	0.036	0
12260	1	ve-5Mbps	1	0.916395819	0	0	0.936	0.936	0.936	0
590	1	down-max	1	0.9024032959	0	0	0.9066666667	0.9066666667	0.9066666667	0
1380	1	email-10kbps	1	0.779064622	0	0	0.9066666667	0.9066666667	0.9066666667	0
330	1	chat-128kbps	1	0.08574044404	0	0	0.05466666667	0.05466666667	0.05466666667	0
16960	1	vr-1.5Mbps	0	0.9357137407	0	0	0.7733333333	0.7733333333	0.7733333333	0
1730	1	chat-128kbps	1	0.08574044404	0	0	0.6346666667	0.6346666667	0.6346666667	0
12760	1	ar-100kbps	0	0.6145265888	0	0	0.08866666667	0.08866666667	0.08866666667	0
9680	1	ve-5Mbps	1	0.916395819	0	0	1.828	1.828	1.828	0
920	1	email-10kbps	1	0.01515220874	0	0	0.03466666667	0.03466666667	0.03466666667	0
70	1	chat-128kbps	1	0.1983672847	0	0	0.282	0.282	0.282	0
150	1	ve-5Mbps	1	0.916395819	0	0	0.936	0.936	0.936	0
3100	1	up-max	1	0.0003356984817	0	0	0.036	0.036	0.036	0
14310	1	vr-1.5Mbps	0	0.9357137407	0	0	0.1973333333	0.1973333333	0.1973333333	0
320	1	ae-2Mbps	1	0.6598001068	0	0	3.612	3.612	3.612	0
390	1	down-max	1	0.5160601205	0	0	0.9066666667	0.9066666667	0.9066666667	0
7030	1	up-max	1	0.0003356984817	0	0	0.036	0.036	0.036	0

Fonte: Próprio Autor, 2023.

As *features* obtidas nas bases de dados foram analisadas por meio de matrizes de correlação, que representam as ligações entre as variáveis. As Figuras 5 e 6 demonstram que algumas variáveis possuem forte correlação, enquanto outras aparentam ter correlação muito fraca. Outros métodos para seleção de *features* como o *Recursive Feature Elimination* (RFE) podem auxiliar a diminuir a dimensionalidade da base conforme a adaptação do modelo ML, no entanto, essas ferramentas não são facilmente implementadas em modelos de aprendizado baseados em *data streaming*. Dito isso, algumas *features* foram manualmente removidas:

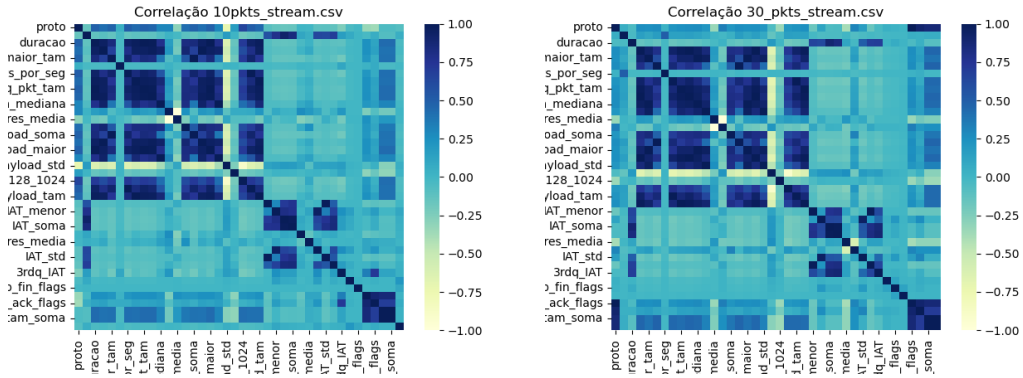
- tcp_urg_flags: nem um fluxo utilizou essa *flag*.
- tcp_windowsize_std, tcp_windowsize_media: são referentes ao controle de congestionamento da rede e não representam uma característica do fluxo exatamente.
- dport: foi removido por influenciar muito na classificação (por ser tráfego de apenas dois computadores).
- Restaram 41 *features*.

Figura 5. Matriz de correlação para as *features* - *batch*.



Fonte: Próprio Autor, 2023.

Figura 6. Matriz de correlação para as *features* - *streaming*.



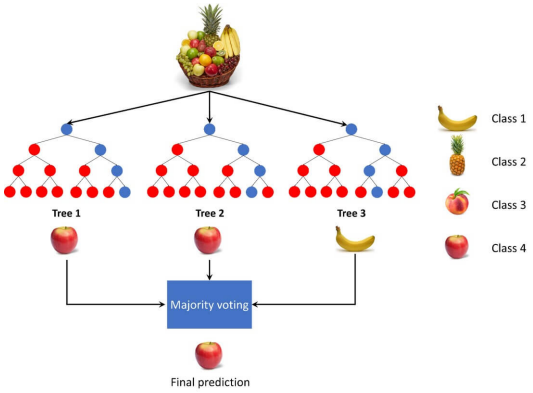
Fonte: Próprio Autor, 2023.

3.3. Modelos de classificação de *machine learning*

Os modelos de ML para classificação para tipo de dados supervisionados (rotulados) podem ser do tipo que processam dados em *batch* ou que processam dados em *streaming*. Modelos do tipo *batch* agrupam *features* dos dados em um bloco para realizar a classificação, necessitando armazenar o bloco de dados. Por outro lado, modelos que tratam dados em *streaming* classificam os dados conforme chegam, sem a necessidade de armazenar e aguardar dados. Dentre os modelos de aprendizagem mais comuns na literatura estão os algoritmos de árvores de decisão e suas variações, que geralmente apresentam altas taxas de precisão, e modelos clássicos como o SVM.

Uma Árvore de Decisão (*Decision Tree*) é um modelo de aprendizagem em formato de árvore que funciona como um fluxograma, onde cada nó representa uma *feature*, cada ramificação representa uma regra de decisão e cada folha representa uma saída (classificação) [Navlani 2023]. O modelo *Random Forest* é outro modelo de *machine learning* que pode ser utilizado para realizar classificações de dados em blocos. Este algoritmo cria de forma aleatória várias *Decision Trees* e combina o resultado de todas elas para chegar na classificação resultante. A Figura 7 representa como uma *Random Forest* se parece.

Figura 7. Representação Random Forest.



Fonte: [Zivkovic 2022].

O modelo SVM é um dos classificadores mais populares para aprendizagem supervisionada. O algoritmo SVM busca separar o espaço n-dimensional (n representa o número de *features*) em classes criando uma divisão chamada de hiperplano. Este hiperplano é construído baseado nos vetores de suporte, definidos pelos pontos mais próximos do separador. Assim, para separar duas classes de dados o hiperplano encontrado deve ser aquele cuja margem de distância entre os vetores de suporte é a maior [Gandhi 2018].

O modelo *Hoeffding Tree* é um modelo de classificação para fluxos de dados do tipo *streaming*, onde fluxos contínuos de dados são categorizados ao longo do tempo [Verma 2021]. Este modelo também se inspira em árvores de decisão. Ele utiliza um coeficiente para determinar quando um nó deve ser dividido para gerar uma nova árvore. As árvores são construídas de forma incremental e se assume que se os dados não mudarem com o tempo, os resultados obtidos são equivalentes ao de métodos em *batch*. O principal ponto destacado nesse modelo é que não precisa armazenar dados para seu funcionamento.

O modelo *Adaptive Random Forest* é uma versão modificada do *Random Forest* para lidar com dados do tipo *streaming*. Uma das alterações é que a base de aprendizagem são as árvores modificadas do tipo *Hoeffding Tree*. O algoritmo inicia um número definido de árvores que recebem os dados do *stream*. Quando é detectada uma perda de desempenho, árvores de *backup* são treinadas em paralelo e podem ser utilizadas para substituir uma árvore original [Alkazaz and Saado Kharouki 2020].

4. Experimentos e Resultados

Os modelos de classificação descritos foram avaliados nos experimentos utilizando a técnica de divisão da base de dados chamada *cross validation*. Esse método permite avaliar melhor o desempenho dos modelos de aprendizado de máquina, pois permite particionar a base de dados em K subconjuntos com quantidade de dados semelhantes. Isso permite realizar K iterações de treino e teste, iterando sobre quais conjuntos são utilizados para teste e quais são utilizados para treino [Rabello 2017].

Desta forma, definimos K como sendo 5 e repartimos as bases de dados em blocos de tamanhos iguais. Assim, em uma iteração o bloco 1 é utilizado para teste enquanto os demais são utilizados para treino, na próxima iteração o bloco 2 é utilizado para teste e

os demais são utilizados para treino, seguindo até que cada bloco tenha sido utilizado ao menos uma vez para teste.

As bases de dados para fluxos do tipo *streaming*, conforme explicado no processo de extração de *features*, foram randomizadas mantendo a ordem de precedência das *features* dos pacotes. Além disso, para melhorar o desempenho dos modelos de *streaming*, foi garantido que não existisse mais de 20 fluxos ativos ao mesmo tempo, ou seja, o modelo não recebe *features* misturadas de mais de 20 fluxos. Isso faz com que os modelos melhorem o poder de aprendizagem ao custo de limitar o tipo da entrada.

Para executar os testes, a máquina utilizada foi um computador com sistema operacional Arch Linux 64bits, processador AMD Ryzen 5 5600 e 20Gb de RAM, sendo utilizado apenas uma *thread* para os testes. Os modelos de aprendizado de máquina do tipo *batch* utilizados foram importados da biblioteca *scikit-learn*, já os modelos de aprendizado de máquina do tipo *streaming* foram importados da biblioteca *scikit-multiflow*, que implementa os modelos baseados na biblioteca *Massive Online Analisys* (MOA).

As métricas selecionadas para análise dos modelos foram a precisão, acurácia, *recall* e F1-*score*. A precisão remete quantos dados o classificador identificou corretamente, a acurácia mede a frequência que o classificador identifica um dado corretamente, o *recall* representa a taxa de quantos casos positivos reais o modelo foi capaz de prever corretamente e por fim o F1-*score* é uma média harmônica entre precisão e *recall* [Agrawal 2023].

Após executar as 5 iterações com cada base em nos modelos de classificação de dados do tipo *batch* e do tipo *streaming*, foi possível analisar as médias dos valores das métricas de desempenho obtidas. A Tabela 4, representa a comparação entre os modelos de aprendizado para as bases de dados de *features* dos sub fluxos de 10 pacotes. Como é possível identificar, o modelo *Random Forest* (*batch*) obteve os melhores indicadores e um desvio padrão baixo com relação às iterações do *cross validation*. O modelo *Adaptive Random Forest* obteve melhor desempenho geral que seu similar *Hoeffding Tree* nas bases de dados do tipo *streaming*.

Tabela 4. Tabela de comparação entre os modelos (10pkts).

Modelo ML	Precisão	Acurácia	Recall	F1-Score	Desvio Padrão Precisão
Decision Tree	0.97	0.97	0.97	0.97	0.0039
Random Forest	0.98	0.98	0.98	0.98	0.0045
SVM	0.9	0.9	0.9	0.9	0.0074
Hoeffding Tree	0.75	0.78	0.76	0.72	0.11
Adaptive Random Forest	0.88	0.85	0.86	0.85	0.024

Fonte: Próprio Autor, 2023.

A Tabela 5, compreende a comparação entre os modelos de *features* para sub

fluxos de 30 pacotes. Nesses cenários, as bases de dados do tipo *batch* encolhem $3x$, pois mais pacotes são utilizados para extrair as *features*, gerando menos entradas de dados, mas com valores mais precisos. Os modelos para dados do tipo *batch* sofreram uma leve redução no desempenho geral, nos casos de *Decision Tree* e SVM, enquanto o modelo *Random Forest* não teve alteração. Já nos modelos do tipo *streaming*, o tamanho da base pouco altera, no entanto, as sequências de pacotes de cada sub fluxo aumenta em $3x$, fazendo com que mais conjuntos de *features* de um mesmo fluxo sejam analisados. Com isso, os modelos melhoraram de desempenho geral, principalmente o modelo *Hoeffding Tree*, que aumentou a precisão em 16%.

Tabela 5. Tabela de comparação entre os modelos (30pkts).

Modelo ML	Precisão	Acurácia	Recall	F1-Score	Desvio Padrão Precisão
Decision Tree	0.95	0.95	0.95	0.95	0.013
Random Forest	0.98	0.98	0.97	0.98	0.0048
SVM	0.73	0.71	0.71	0.67	0.026
Hoeffding Tree	0.91	0.92	0.91	0.90	0.015
Adaptive Random Forest	0.91	0.92	0.92	0.91	0.025

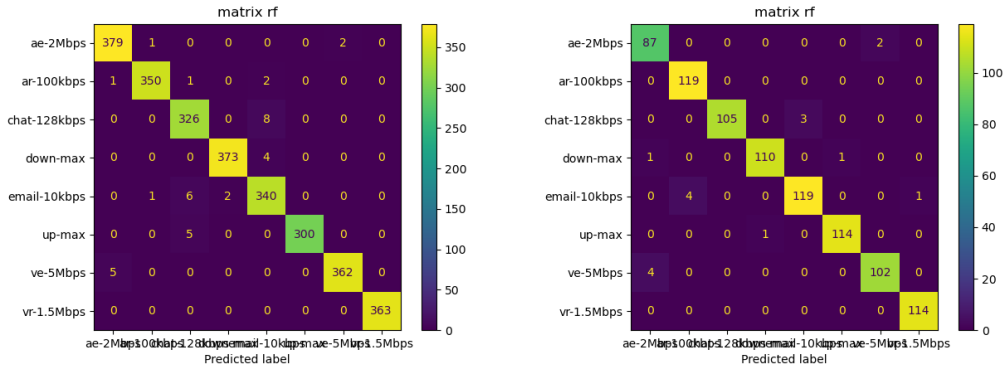
Fonte: Próprio Autor, 2023.

Os modelos para classificação de dados em *batch*, treinam e testam suas bases praticamente de forma instantânea (menor que $1s$), devido a utilizar apenas uma entrada da base para realizar a classificação, pois representa o agregado de 10 ou 30 pacotes. Por outro lado, os modelos de classificação para dados do tipo *streaming* precisam ler 10 ou 30 entradas da base para terem uma classificação final.

Para a base de sub fluxos de 10 pacotes, o modelo *Hoeffding Tree* consumiu em média $9s$ para treinar o modelo em cada iteração do *cross validation* e $4.17s$ para testar o modelo, nesta mesma base o *Adaptive Random Forest* consumiu em média $188.3s$ para treinamento e $22.9s$ para o processo de teste. A base de dados de 30 pacotes demandou em média $7.9s$ para o modelo *Hoeffding Tree* realizar o treinamento em cada iteração e $4.15s$ para o processo de teste, enquanto o modelo *Adaptive Random Forest* consumiu $176.3s$ para o treinamento e $23.56s$ para teste.

Desta forma, com base nas observações levantadas, se pode afirmar que o modelo *Random Forest* obteve o melhor desempenho na classificação de tráfego de rede. A Figura 8 apresenta a matriz de confusão para o melhor método avaliado (RF) na melhor iteração de *cross validation* obtida. Variando a quantidade de pacotes, os modelos baseados em dados *batch* não foram afetados, assim, é possível utilizar uma menor quantidade de amostras para fazer a classificação. Desta forma, coletando menos pacotes de fluxos de dados, se diminui o impacto no desempenho da rede e menos pacotes precisam se encaminhados ao controlador SDN.

Figura 8. Matriz de confusão 10 pkts (esq) e 30 pkts (dir) batch.



5. Conclusão

Nesse artigo, foi apresentado uma comparação entre cinco modelos de classificação baseados em *machine learning*. Foram analisados os modelos baseados em classificação de dados do tipo *streaming Hoeffding Tree* e *Adaptive Random Forest*, e os modelos baseados em classificação de dados do tipo *batch Random Forest*, *Decision Tree* e SVM.

A primeira etapa do trabalho foi a de extração de dados de pacotes de fluxos de dados categorizados e o processamento desses dados em uma base de *features* que pudesse ser utilizada para treinamento e teste dos modelos de aprendizado. Foram encontradas bases de dados de capturas de pacotes de tráfego de rede rotuladas por aplicações como *Facebook* áudio e *Skype* vídeo, a partir dos quais, os pacotes dessas aplicações foram processados em fluxos *one-way* e então em sub fluxos, para respeitarem a características dos fluxos em um cenário de rede gerenciada pelo *framework* FLOWPRI-SDN. Após as extrações dos sub fluxos, foram obtidas as *features* de conjuntos de 10 e 30 pacotes, gerando as bases de dados utilizadas nos modelos de classificação *machine learning*.

A partir dos treinamentos e testes realizados com os modelos por meio de *cross validation*, foram coletados valores para métricas de desempenho, que auxiliaram a identificar os modelos mais propícios a serem utilizados no FLOWPRI-SDN. Nos modelos para dados do tipo *streaming*, foram encontradas dificuldades para gerar uma base de dados que fossem compatíveis com as limitações do controlador. Devido a isso, as bases de dados extraídas de 10 e 30 pacotes demonstraram que os modelos de dados *streaming* tendem a se sair melhor com mais pacotes de amostra. No entanto, o *Random Forest* demonstrou que uma base extraída de 10 ou 30 pacotes não diminui seu desempenho de classificação. Deste modo, o uso de blocos de 10 pacotes pode implicar em menos sobrecarga no controlador e na rede em que se atua.

Agradecimentos

Este trabalho conta com o apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES - Brasil (PROAP/AUXPE).

Referências

[Agrawal 2023] Agrawal, S. K. (2023). Metrics to evaluate your classification model to take the right decisions. https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/.

[Alkazaz and Saado Kharouki 2020] Alkazaz, A. and Saado Kharouki, M. (2020). Evaluation of adaptive random forest algorithm for classification of evolving data stream.

[Amaral et al. 2016a] Amaral, P., Dinis, J., Pinto, P., Bernardo, L., Tavares, J., and Mamede, H. (2016a). Machine learning in software defined networks: Data collection and traffic classification. In *2016 IEEE 24th International Conference on Network Protocols*, pages 1–5.

[Amaral et al. 2016b] Amaral, P., Dinis, J., Pinto, P., Bernardo, L., Tavares, J., and Mamede, H. S. (2016b). Machine learning in software defined networks: Data collection and traffic classification. In *2016 IEEE 24th International conference on network protocols (ICNP)*, pages 1–5. IEEE.

[Biondi 2023] Biondi, P. (2023). Welcome to scapy's documentation! https://support.google.com/youtube/answer/78358?hl=en.

[Gandhi 2018] Gandhi, R. (2018). Support vector machine — introduction to machine learning algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[Google 2023] Google (2023). System requirements and supported devices for youtube. https://support.google.com/youtube/answer/78358?hl=en.

[Huawei 2022] Huawei (2022). Overview of bandwidth managementt. https://support.huawei.com/enterprise/en/doc/EDOC1000174073/1686bcc9/overview-of-bandwidth-management.

[ITU 2001] ITU (2001). *End-user multimedia QoS categories*. International Telecommunication Union (ITU).

[Karakus and Durresi 2017] Karakus, M. and Durresi, A. (2017). Quality of service (qos) in software defined networking (sdn): A survey. *Journal of Network and Computer Applications*, 80:200 – 218.

[Microsoft 2023] Microsoft (2023). How much bandwidth does skype need? https://support.skype.com/en/faq/fa1417/how-much-bandwidth-does-skype-need.

[Mocelin Júnior and Fiorese 2023] Mocelin Júnior, N. J. and Fiorese, A. (2023). Flowpri-sdn: A framework for bandwidth management for prioritary data flows applied to a smart city scenario. In *International Conference on Advanced Information Networking and Applications*, pages 346–357. Springer.

[Navlani 2023] Navlani, A. (2023). Decision tree classification in python tutorial. https://www.datacamp.com/tutorial/decision-tree-classification-python.

[Rabello 2017] Rabello, E. B. (2017). Cross validation: Avaliando seu modelo de machine learning. https://medium.com/@edubrazrabello/cross-validation-avaliando-seu-modelo-de-machine-learning-1fb70df15b78.

[Raikar et al. 2020] Raikar, M. M., Meena, S., Mulla, M. M., Shetti, N. S., and Karanandi, M. (2020). Data traffic classification in software defined networks (sdn) using supervised-learning. *Procedia Computer Science*, 171:2750–2759.

[UNB 2016] UNB (2016). Vpn-nonvpn dataset (iscxvpn2016). `https://www.unb.ca/cic/datasets/vpn.html`.

[Verma 2021] Verma, Y. (2021). A beginner's guide to hoeffding tree with python implementation. https://analyticsindiamag.com/a-beginners-guide-to-hoeffding-tree-with-python-implementation/.

[Yan Chen 2004] Yan Chen, Toni Farley, N. Y. (2004). Qos requirements of network applications on the internet. *Systems Management*, 4.

[Zander et al. 2012] Zander, S., Nguyen, T., and Armitage, G. (2012). Sub-flow packet sampling for scalable ml classification of interactive traffic. In *37th Annual IEEE Conference on Local Computer Networks*, pages 68–75.

[Zhou et al. 2018] Zhou, D., Yan, Z., Fu, Y., and Yao, Z. (2018). A survey on network data collection. *Journal of Network and Computer Applications*, 116:9–23.

[Zivkovic 2022] Zivkovic, S. (2022). Machine learning – introduction to random forest. https://datahacker.rs/012-machine-learning-introduction-to-random-forest/.