UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

AMAURY TEIXEIRA CASSOLA

K-Flix: Aplicação de Clustering para Classificação de Tráfego de Streaming de Vídeo em Planos de Dados Programáveis

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer Filho

Porto Alegre
2024

RESUMO

Streaming de vídeo vem se tornando parte cada vez mais significativa da Internet, com aplicações que oferecem este serviço compondo mais da metade do tráfego mundial. Dada a grande demanda que esta aplicação representa, surge a necessidade de políticas de *Quality of Service* (QoS) específicas para este nicho. A implementação de soluções de QoS depende, porém, da identificação em tempo hábil da presença deste tipo de tráfego na rede. Avanços recentes na tecnologia de Planos de Dados Programáveis (PDP) possibilitam o desenvolvimento de soluções eficientes para classificação de tráfego, delegando parte do processamento aos dispositivos que formam a infraestrutura da rede. Além da necessidade de processamento eficiente de modo a não afetar o funcionamento da rede, dispositivos de rede programáveis em geral não oferecem suporte a operações aritméticas complexas como divisão e logaritmo, havendo, então, a necessidade de soluções simples e eficientes. Este trabalho propõe o emprego de agrupamento, uma técnica de Aprendizado de Máquina (*Machine Learning* - ML), para classificação de tráfego referente à *streaming* de vídeo em planos de dados programáveis. Os agrupamentos são gerados e classificados *off-line*, de modo que distingam entre *streaming* de vídeo e outros tipos de tráfego. O processo de inferência, assim, consiste no cálculo da distância entre os atributos de uma entrada e cada um dos centróides dos agrupamentos gerados pelo modelo. Este cálculo é traduzido em uma série de comparações que, por sua vez, são integradas às *pipelines* de processamento de dispositivos programáveis. Desenvolveu-se um protótipo do sistema em um ambiente virtual e realizaram-se testes utilizando capturas de tráfego real. Os resultados obtidos com os testes demonstram o potencial da utilização de agrupamentos para a classificação de tráfego em planos de dados programáveis.

Palavras-chave: Classificação de tráfego. planos de dados programáveis. aprendizado de máquina. agrupamento. streaming de vídeo.

Application of Clustering for Video Streaming Traffic Classification in Programmable Data Planes

ABSTRACT

In recent years, video streaming has become a significant part of the Internet, with applications that offer this service accounting for more than half of global traffic. Given the high demand represented by this application class, specific Quality of Service (QoS) solutions become necessary. However, the implementation of such QoS solutions depends on the timely identification of such traffic in the network. Recent advances in Programmable Data Planes allow the development of efficient solutions for traffic classification, deploying part of the processing to the devices that form the network infrastructure. Besides the need for efficient processing so as to not impact the network operation, programmable network devices generally do not support complex arithmetic operations such as division and logarithms, and therefore require simple and efficient solutions. This work proposes the usage of clustering, a Machine Learning technique, for video streaming traffic classification in programmable data planes. The clusters are generated and classified off-line such that it is possible to distinguish between video streaming and other forms of traffic. The inference process thus consists in the computing of the distance between the attributes of an ingress packet flow and the centroids of each of the clusters generated by the algorithm. This computation is translated to a series of comparisons which are integrated to the processing pipelines of the programmable devices. A prototype of the proposed system was developed in a virtual environment and tests were executed using real traffic captures. The results observed demonstrate the potential of employing clustering for traffic classification in programmable data planes.

Keywords: Traffic classification, programmable data planes, machine learning, clustering, video streaming.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PISA	Protocol-Independent Switching Architecture
PDP	Planos de Dados Programáveis
QoS	Quality of Service
SDN	Software-Defined Network
FSM	Finite State Machine
DoS	Denial of Service
WSS	Within-Cluster Squared Sum of Errors
PPP	Payload Peak Points
RMT	Reconfigurable Match Tables
ML	Machine Learning
PCA	Principal Component Analysis

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem seu foco na classificação de fluxos de pacotes gerados por aplicações de *streaming* de vídeo. É aplicada uma técnica de aprendizado de máquina não-supervisionado como forma de criar um procedimento de inferência simples que pode ser executado eficientemente no plano de dados de um *switch* programável. Em específico, foi escolhido o algoritmo K-Means, um dos mais populares algoritmos de aprendizado não-supervisionado e que já mostrou bons resultados no contexto de classificação de tráfego em termos de acurácia e eficiência (ERMAN; ARLITT; MAHANTI, 2006). Na Seção 1.1, se encontra uma contextualização mais profunda sobre os assuntos abordados neste trabalho, enquanto que as Seções 1.2 e 1.3 apresentam motivações e objetivos, respectivamente. Por fim, a Seção 1.4 apresenta a organização deste documento.

1.1 Contextualização

De acordo com o Sandvine Global Internet Phenomena Report (SANDVINE, 2023), as aplicações de *streaming* de vídeo representaram mais de 65% do tráfego mundial na Internet no ano de 2022. Além disso, o relatório cita a tendência entre diferentes aplicações, sobretudo mídias sociais, de integrar tecnologias de *streaming* de vídeo em suas plataformas como maneira de atrair público. Levando em conta a demanda altíssima e incomum deste nicho, seria útil para provedores de Internet ter a capacidade de implementar políticas de QoS específicas para tratar este tráfego, que representa mais da metade das atividades desempenhadas por seus clientes. A execução de políticas de QoS específicas para uma classe de aplicações, porém, depende da identificação eficiente da presença deste tipo de tráfego na rede (WU et al., 2023).

A comunidade de pesquisa propôs múltiplas soluções para o problema da classificação de tráfego na Internet ao longo dos anos, com as mais populares atualmente se baseando na análise de atributos estatísticos de fluxos de pacotes como duração, tamanho médio dos *payloads* e tempo médio entre pacotes (AZAB et al., 2022). Em especial, numerosos trabalhos foram desenvolvidos neste paradigma utilizando tecnologias de Aprendizado de Máquina para realizar o mapeamento entre padrões de comportamento e classes de aplicações (NGUYEN; ARMITAGE, 2008a). Embora bons resultados tenham sido observados nas propostas utilizando ML e atributos estatísticos, severas limitações também foram notadas, sobretudo quanto à eficiência do sistema. Em geral, as imple-

mentações implicam em uma sobrecarga significativa de comunicação na rede, dado que todos os pacotes necessitam ser duplicados e encaminhados para um servidor onde ocorre a classificação, o que dificulta a aplicação destas tecnologias.

Avanços recentes na tecnologia de Planos de Dados Programáveis abriram novas possibilidades para a aplicação de técnicas de ML na classificação de tráfego. Em especial, a linguagem P4 possibilita a programabilidade dos planos de dados de dispositivos de rede, de modo que novos processos possam ser implementados e testados sem necessidade de substituição de *hardware* (BOSSHART et al., 2014). É especialmente popular a injeção de modelos pré-treinados diretamente nos dispositivos programáveis, de modo que o estágio de inferência dos modelos aconteça com o mínimo de sobrecarga de comunicação. Diferentes trabalhos recentes exploram a aplicabilidade deste método e em geral obtém bons resultados com pouco ou nenhum impacto na performance da rede e alta eficiência de classificação (PARIZOTTO et al., 2023).

1.2 Motivação

Embora existam muitos trabalhos recentes investigando o uso de técnicas de Aprendizado de Máquina para classificação de tráfego em PDPs, há relativamente pouca pesquisa focada no paradigma de aprendizado não-supervisionado, sobretudo se comparado com modelos como Árvores de Decisão e Redes Neurais. Em (PARIZOTTO et al., 2023), uma *survey* extensiva sobre o emprego de ML em PDPs, apenas três trabalhos são citados que utilizam *clustering* em suas propostas. Sendo assim, se torna necessária uma maior investigação da aplicabilidade de aprendizado não-supervisionado para este problema. Sobretudo, o mapeamento do estágio de inferência para as *pipelines* dos dispositivos programáveis ainda não conta com um método ótimo, sendo implementado de maneiras diferentes nos trabalhos supracitados. Além disso, embora alguns trabalhos incluam esta classe de tráfego entre outras durante a avaliação dos modelos propostos, não há na literatura recente um estudo específico sobre a classificação de *streaming* de vídeo, mesmo estando entre as classes de tráfego mais pertinentes, o que demonstra outra lacuna que este trabalho objetiva preencher.

Assim, se pretende criar um sistema capaz de processar eficientemente fluxos de pacotes em uma rede de modo a classificar as aplicações que os geraram como *streaming* de vídeo ou não. Para tanto, é utilizada a linguagem P4 para mapear para a *pipeline* de um *switch* programável um processo de inferência baseado nos centróides de um modelo K-

Means treinado previamente. O *switch*, por sua vez, acumulará valores estatísticos sobre os fluxos de pacotes e utilizará o processo de inferência para classificá-los em tempo de execução.

1.3 Objetivos

O principal objetivo deste trabalho é propor, implementar e avaliar uma solução que combine técnicas de aprendizado de máquina não-supervisionado com *switches* programáveis que oferecem suporte à linguagem P4. Assim, se espera criar um sistema simples, porém robusto, que seja capaz de classificar fluxos de *streaming* de vídeo com precisão sem prejudicar significativamente a performance dos *switches*.

O método para alcançar este objetivo consiste nos seguintes sub-objetivos:

- O estudo dos trabalhos relacionados, sobretudo em implementações de algoritmos de aprendizado de máquina não-supervisionado e classificação de tráfego em PDPs;
- O treinamento de um modelo K-Means com *datasets* apropriados;
- A execução de classificação de tráfego no plano de dados utilizando os centróides gerados pelo modelo;
- Por fim, a avaliação da implementação quanto à qualidade da classificação utilizando métricas como acurácia, precisão, *recall* e *F1 score*.t

1.4 Organização deste Trabalho

O restante do presente trabalho organiza-se da seguinte forma. No Capítulo 2, é apresentado o referencial teórico, discorrendo sobre a tecnologia de programabilidade de rede, em especial sobre planos de dados programáveis, e o problema da classificação de tráfego, apresentando K-Means, o modelo de aprendizado de máquina não-supervisionado utilizado no sistema proposto. O Capítulo 3 apresenta outros trabalhos que implementam aprendizado de máquina em dispositivos de redes programáveis. No Capítulo 4, o sistema proposto é descrito em detalhes. O Capítulo 5 detalha a implementação de um protótipo do sistema e os testes realizados, além de analisar os resultados obtidos. Por fim, no Capítulo 6, é apresentada uma conclusão a este trabalho, incluindo um resumo das contribuições e a discussão das possibilidades de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo tem por objetivo apresentar os principais conceitos necessários para a compreensão deste trabalho. Primeiramente, a Seção 2.1 discute conceitos e definições fundamentais sobre a programabilidade de redes com enfoque em Planos de Dados Programáveis. A seguir, a Seção 2.2 apresenta uma definição do problema de classificação de tráfego de Internet e discorre sobre as soluções propostas, sobretudo com emprego de técnicas de Aprendizado de Máquina.

2.1 Programabilidade de Rede

A programabilidade de um produto pode ser identificada em termos da sua capacidade de ter seu comportamento definido por algoritmos externos (HAUSER et al., 2023). Equipamentos de redes convencionais, como *switches* e *Network Interface Cards*, têm sua funcionalidade definida por algoritmos e protocolos de rede implementados em código proprietário e fechado (KALJIC et al., 2019). Estes dispositivos oferecem pouca customização em seu comportamento, em geral permitindo apenas a configuração de parâmetros ou protocolos, sem modificações diretas aos algoritmos subjacentes, que se mantêm imutáveis. Neste contexto, a inovação se torna complexa e custosa, dado que o suporte a novas funcionalidades se mostra dependente da iniciativa dos fabricantes dos produtos (MICHEL et al., 2021). Além disso, a manutenção e a gerência dos equipamentos de rede também são afetados diretamente, vista a complexidade de se construir infraestruturas otimizadas para casos de uso específicos ou adaptar equipamentos existentes para demandas emergentes. Sendo assim, a programabilidade destes dispositivos de rede significaria maior autonomia por parte tanto de operadores de rede quanto de pesquisadores e simplificaria o desenvolvimento de novas soluções em diversos casos de uso importantes. Ademais, fabricantes destes dispositivos também se beneficiariam, podendo facilmente construir produtos diversos sobre um mesmo circuito.

O funcionamento de um dispositivo de rede pode ser separado logicamente entre um plano de controle, responsável por estabelecer políticas de processamento de pacotes, e um plano de dados, que se encarrega principalmente de executar estas políticas. Em redes tradicionais, os planos de controle dos dispositivos executam algoritmos distribuídos baseados em trocas de mensagens para popular suas tabelas de encaminhamento e sincronizar seu funcionamento. No início dos anos 2000, objetivando um maior grau de

programabilidade de rede, surge o paradigma de *Software-Defined Networks* (SDNs) (FEAMSTER; REXFORD; ZEGURA, 2014). Em SDNs, o plano de controle é parcialmente separado dos dispositivos da infraestrutura da rede, sendo transportado para um controlador central, de modo que um único programa de controle possa controlar múltiplos nodos do plano de dados. Com o advento das SDNs, se tornou possível a implementação de algoritmos de plano de controle centralizados, possibilitando mais simplicidade e maior eficiência em múltiplos casos de uso importantes. Os sistemas que implementam este paradigma oferecem programabilidade do plano de controle, de modo que usuários possam introduzir seus próprios algoritmos independentemente dos comportamentos definidos pelos fabricantes. Estes algoritmos, então, podem suprir informações de processamento e encaminhamento através de APIs previamente definidas para o plano de dados, que segue distribuído entre os vários nodos da rede e não programável. Uma das mais importantes APIs desenvolvidas para SDNs é OpenFlow (MCKEOWN et al., 2008), uma tecnologia de código aberto que visa padronizar a API de mensagens de controle do plano de dados, assim possibilitando a reconfiguração de protocolos da rede pelas aplicações SDN. Porém, OpenFlow presume uma funcionalidade do plano de dados específica pouco customizável (HAUSER et al., 2023). Sendo assim, embora OpenFlow ofereça programabilidade do plano de controle, os algoritmos do plano de dados, diretamente responsáveis pelo processamento dos pacotes que viajam pela rede, prosseguem fixos.

2.1.1 Planos de Dados Programáveis

Com os planos de controle programáveis, tornou-se possível implementar novas tecnologias na infraestrutura da rede, incluindo o emprego de técnicas de Aprendizado de Máquina (AMARAL et al., 2016). Por outro lado, embora o comportamento em alto nível da rede pudesse ser modificado, os equipamentos da infraestrutura da rede prosseguiram com seu funcionamento fixo, o que manteve presentes vários dos problemas relacionados à ausência de programabilidade, sobretudo a dificuldade em desenvolver novas funcionalidades e adaptar equipamentos existentes para novas demandas. Implementar programabilidade no plano de dados, então, significaria possibilitar uma maior exploração das capacidades da infraestrutura da rede como um todo e simplificaria significativamente o processo de inovação.

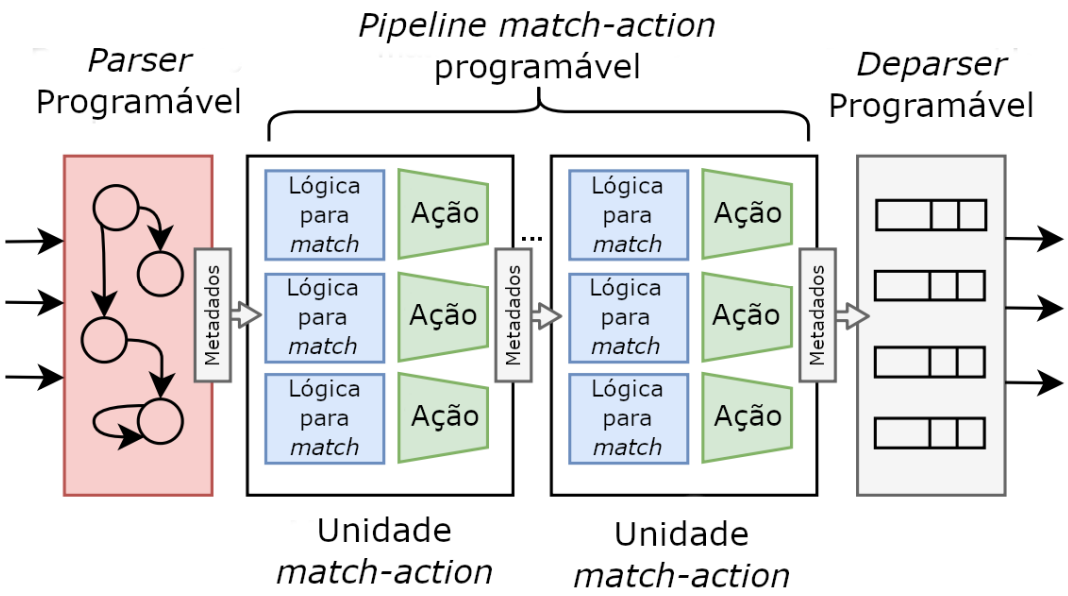
A nível de *hardware*, dispositivos de rede baseiam-se em Circuitos Integrados de Aplicação Específica (*Application-specific Integrated Circuits* - ASICs) de processamento

de pacotes dada a demanda por processamento em alta velocidade. Os ASICs empregados em dispositivos tradicionais oferecem a vazão necessária ao funcionamento da rede, mas comumente são circuitos de função fixa, ou seja, cujo funcionamento não é programável (HAUSER et al., 2023). Com a utilização de *hardware* especializado nos dispositivos, a programação do plano de dados não poderia ser realizada por linguagens de programação usuais, visto que os programas resultantes não poderiam ser mapeados diretamente para o equipamento. Assim, os esforços para alcançar a programabilidade do plano de dados passariam necessariamente pela definição de modelos de abstração do *hardware*, nos quais poderiam basear-se as linguagens de programação específicas para PDPs. Desta forma, os algoritmos para planos de dados poderiam ser expressados em linguagens de alto-nível e posteriormente compilados para execução nos equipamentos que oferecessem suporte ao modelo utilizado. Diferentes modelos de abstração foram desenvolvidos, sendo *Protocol-Independent Switching Architecture* (PISA) um dos mais relevantes.

2.1.2 Arquitetura PISA

Um dos principais modelos de abstração do *hardware* do plano de dados, PISA é uma arquitetura baseada em uma *pipeline* programável de tabelas *match-action* reconfiguráveis (*Reconfigurable Match Tables* - RMTs) (BOSSHART et al., 2013). Neste modelo, os cabeçalhos e o *payload* de um pacote são transportados separadamente, com o último não sendo considerado pelo processamento. Além dos campos presentes nos cabeçalhos, também são considerados metadados do pacote implementados em *hardware* tais como *timestamps*, tamanho em *bytes* e número de porta de entrada. A arquitetura do modelo se divide em três estágios programáveis: um *parser* que extrai os *headers* dos pacotes que chegam ao dispositivo; uma *pipeline* de processamento que utiliza tabelas *match-action* para processar e modificar os dados de um pacote e, por fim, um *deparser* responsável por concatenar os cabeçalhos de um pacote a seu *payload* antes de enviá-lo para a fila de saída. A Figura 2.1 apresenta um diagrama dos componentes da arquitetura PISA.

Figura 2.1: Modelo Abstrato PISA



Fonte: Adaptado de Hauser et al. (HAUSER et al., 2023)

Parser. Executa a desserialização dos *headers* do pacote, convertendo a *bytestream* em estruturas de dados definidas pelo usuário. A extração dos cabeçalhos de um pacote é realizada através de uma Máquina de Estados Finita (*Finite State Machine* - FSM) definida pelo programador que estabelece quais tipos de cabeçalhos serão esperados e em qual ordem, determinando quais protocolos são aceitos pelo equipamento.

Pipeline match-action. Consiste de múltiplas tabelas *match-action* e é onde a parte principal de um algoritmo de processamento de pacotes é definida. Neste estágio, é possível modificar, remover ou adicionar cabeçalhos ao pacote, além de aplicar transformações sobre os metadados. Cada entrada em uma tabela *match-action* é composta por três informações principais: uma regra de *matching*, uma ação e dados para esta ação. A regra define uma comparação entre algum dado do pacote e um valor ou faixa específicos de modo que a ação correspondente só será aplicada no caso de um resultado positivo. As ações em uma tabela são operações lógicas e aritméticas e são definidas pelo desenvolvedor de forma similar a funções em linguagens de programação tradicionais. Os dados, por fim, funcionam como parâmetros adicionais às ações e são guardados na memória interna do dispositivo. Em tempo de execução, as tabelas são preenchidas por um plano de controle, dinamicamente modificando a lógica de *matching* e possibilitando adaptação do plano de dados ao longo do tempo.

Deparser. Realiza a serialização do pacote, unindo os cabeçalhos ao *payload* antes de enviá-lo para uma porta de saída. Neste estágio, o pacote é reconstruído com os cabeçalhos que foram modificados, removidos ou adicionados ao longo do processamento. Além disso, o processo de saída em si é definido pelos metadados que indicam se um pacote deve ser encaminhado, descartado, duplicado ou outras ações específicas do equipamento.

2.1.3 Linguagem P4

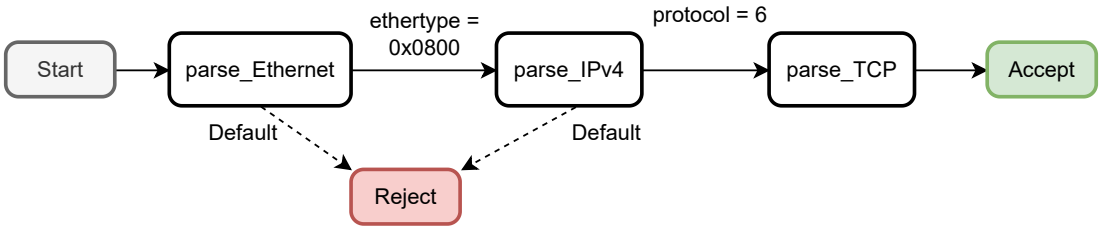
A linguagem P4 (BOSSHART et al., 2014), é atualmente a linguagem mais utilizada para descrever algoritmos de planos de dados (HAUSER et al., 2023). P4 apresenta um modelo de arquitetura de encaminhamento de pacotes baseado em PISA, com a *pipeline match-action* dividida em dois blocos de processamento: entrada e saída. O bloco de entrada contém a maior parte do processamento, enquanto que o bloco de saída é responsável por realizar modificações necessárias em instâncias de pacotes no caso de replicação. Como P4 é uma linguagem independente de *hardware*, os modelos específicos dos produtos, incluindo a estrutura das *pipelines* de processamento e a disponibilidade de componentes não-programáveis, são supridos pelos fabricantes, que também são responsáveis pelos compiladores que serão utilizados para instalar programas P4 em seus equipamentos.

O modelo proposto por P4 é controlado por dois tipos de operações, ambas podendo ser realizadas pelo plano de controle: Configuração e População. As operações de configuração consistem principalmente na implantação de um programa P4 compilado que defina os campos dos *headers* a serem extraídos, a ordem de extração, os protocolos suportados e a ordem dos estágios de *match-action*. As operações de população, por outro lado, realizam modificações no conteúdo das RMTs, criando, atualizando ou removendo regras. Sendo assim, a operação de configuração define o fluxo de controle a ser executado no dispositivo, enquanto que a operação de população define as políticas que serão aplicadas em um dado momento.

Dada a necessidade de operar em *line-rate*, a linguagem P4 apresenta uma série de restrições quando comparada com linguagens de uso geral. Em especial, não há suporte para alocação dinâmica de memória, recursividade e operações de ponto flutuante como divisão e logaritmo (BUDIU; DODD, 2017). Além disso, não há suporte para a construção de laços nos programas, com a única exceção de laços definidos na Máquina de Estados Finita do *parser*.

Programação do parser. O desenvolvedor define estruturas de dados representando os *headers* dos protocolos suportados. Tais estruturas, formadas por valores de tipos primitivos, são utilizadas na Máquina de Estados Finita, que é definida no mesmo programa P4. Além do estado inicial explícito *Start*, a FSM do *parser* programável conta com dois estados finais, *Accept* e *Reject*, que resultam respectivamente no envio dos metadados para o próximo estágio e no descarte do pacote. Além da possibilidade de implementar políticas de rejeição explícitas, a FSM também realiza transições implícitas para o estado *Reject* no caso de não encontrar uma transição apropriada, detectando a presença de um cabeçalho não suportado. O desenvolvedor pode definir os estados intermediários da FSM e as transições entre eles, condicionais ou não, representando quais protocolos serão suportados e em qual ordem. A Figura 2.2 mostra uma FSM de extração de cabeçalhos que espera somente pacotes TCP.

Figura 2.2: Exemplo de FSM para Extração de Cabeçalhos



Fonte: Autor

Tabelas match-action. A construção dos estágios de *match-action*, por outro lado, consiste na definição das estruturas das tabelas e do fluxo de controle das ações, que podem utilizar, além das construções próprias da linguagem e de sua biblioteca padrão, APIs e recursos específicos do *hardware*. A população das tabelas, como citado anteriormente, é realizada pelo plano de controle em tempo de execução e, portanto, não é especificada no código P4. Como exemplo, pode-se construir uma tabela para encaminhamento baseada em *largest prefix matching* do endereço IPv4 de deorstino de um pacote. Vinculam-se duas possíveis ações às entradas desta tabela: *ipv4_forward*, que realiza o encaminhamento para o próximo endereço MAC na rota do pacote, e *drop*, que marca o pacote para ser descartado. Estas ações são implementadas no mesmo código que define a estrutura da tabela em um formato similar a funções em outras linguagens de programação. A ação *ipv4_forward*, em especial, espera como parâmetro o endereço MAC correspondente ao prefixo identificado no pacote. A Tabela 2.1 mostra uma possível configuração para a tabela *ipv4_lpm*.

Tabela 2.1: Um Exemplo de Configuração para a Tabela *ipv4_lpm*

Valor para match	*Tipo de* match	*Ação*	*Parâmetros*
10.0.1.1	32	ipv4_forward	"08:00:00:00:01:11"
10.0.2.2	32	ipv4_forward	"08:00:00:00:02:22"
10.0.3.3	32	drop	

Fonte: Autor

Programação do deparser. Por fim, o *deparser* é implementado como um bloco de controle, similar às ações discutidas anteriormente, que define a maneira como os cabeçalhos de um pacote serão serializados. Neste estágio, um novo pacote é gerado implementando as modificações realizadas sobre os *headers* nos estágios anteriores. O *deparser* emite os cabeçalhos marcados como válidos e descarta os marcados como inválidos, o que possibilita adicionar ou descartar cabeçalhos conforme a necessidade da aplicação.

2.2 Classificação de Tráfego de Internet

A classificação de tráfego de Internet consiste na identificação das aplicações responsáveis por fluxos de pacotes em redes através da análise de diferentes atributos do tráfego. Para provedores de Internet, a utilização de técnicas de classificação de tráfego é essencial para diversas operações de gerência e segurança como a aplicação de políticas de QoS, detecção de intrusões e monitoramento de performance. Ademais, para operadores de redes institucionais ou corporativas, a classificação do tráfego pode ser útil para gerar relatórios estatísticos sobre os serviços utilizados na rede ou priorizar a alocação de recursos para aplicações importantes para o negócio (BOUTABA et al., 2018). Ainda, existem legislações que exigem a possibilidade de intercepção legal no caso de atividades maliciosas, que podem ser identificadas através de classificação de tráfego (AZAB et al., 2022). Múltiplas abordagens foram propostas para esta tarefa ao longo dos anos, podendo ser divididas de maneira simples entre as abordagens clássicas, que analisam um ou mais dados dos pacotes para classificar os fluxos, e as abordagens estatísticas, que utilizam de características do comportamento dos fluxos de pacotes.

2.2.1 Abordagens Clássicas

Dentre as abordagens nesta classe, se identificam três grupos distintos: as baseadas em números de portas (*port-based*), as baseadas em *payloads* (*payload-based*) e as baseadas no comportamento dos hospedeiros (*host behavior-based*) (BOUTABA et al., 2018).

Port-based. A solução mais antiga para o problema da classificação de tráfego baseia-se na análise dos números de porta presentes nos cabeçalhos de transporte dos pacotes. Associam-se números de porta registrados na *Internet Assigned Numbers Authority* às aplicações correspondentes. Embora seja uma abordagem de simples implementação e baixo custo computacional, esta técnica demonstra resultados insatisfatórios, sobretudo pela utilização comum de mecanismos de negociação de portas dinâmicas para mascarar a real aplicação ou evitar *firewalls* (BERNAILLE; TEIXEIRA, 2007).

Payload-based. Também conhecida como *Deep Packet Inspection*, esta abordagem consiste na identificação de assinaturas de aplicações específicas no conteúdo do *payload* dos pacotes. Por não depender dos números de porta dos pacotes, esta técnica contorna os desafios encontrados pelas técnicas *port-based* e oferece maior acurácia (MOORE; PAPAGIANNAKI, 2005). Embora ofereça classificações mais precisas que outras técnicas, a abordagem baseada em *payloads* conta com altos custos computacionais e de armazenamento (BOUTABA et al., 2018). A manutenção de um banco de assinaturas de aplicações, necessário para realizar a classificação, é uma tarefa bastante custosa dado o número crescente de aplicações na Internet. Além disso, com a popularização da criptografia a nível de aplicação e as preocupações crescentes com privacidade e segurança impedindo o acesso ao conteúdo de pacotes, a implementação desta técnica pode ser impraticável.

Host behavior-based. Nesta abordagem, a observação do tráfego se dá na borda da rede, examinando atributos comportamentais dos hospedeiros e baseando-se na noção de que diferentes aplicações geram diferentes padrões de comunicação. Os atributos do comportamento dos hospedeiros analisados incluem quantos *hosts* são contatados, quais protocolos são utilizados e quantas portas são usadas ao longo de uma sessão. Embora seja possível obter classificações relativamente acuradas utilizando esta técnica, a acurácia depende fortemente da localização do sistema de monitoramento e pode ser afetada negativamente por assimetrias de roteamento (BOUTABA et al., 2018).

2.2.2 Abordagens Estatísticas

Diferentemente das abordagens clássicas discutidas anteriormente, que analisam atributos de pacotes individuais ou dos hospedeiros conectados à rede, as abordagens estatísticas objetivam analisar fluxos de pacotes completa ou parcialmente de modo a realizar a classificação de tráfego. Em termos gerais, estas abordagens presumem que um fluxo pode ser classificado a partir da extração de atributos estatísticos de seu comportamento ao longo do seu ciclo de vida, tais como duração da sessão, tamanho médio dos *payloads* e tempo médio entre pacotes (AZAB et al., 2022). Nesta abordagem, a tarefa de classificação pode ser compreendida como a definição de um mapeamento entre um conjunto de atributos de fluxos e um conjunto de classes de aplicação (BOUTABA et al., 2018). Sendo assim, técnicas de Aprendizado de Máquina são especialmente apropriadas, já que oferecem métodos para encontrar este mapeamento automaticamente.

ML Supervisionado. Classificadores baseados em Aprendizado Supervisionado utilizam *datasets* de instâncias pré-classsificadas para encontrar relações entre os atributos das instâncias e as classes, de modo a serem capazes de classificar novas instâncias a partir destas relações. Embora diferentes modelos de Aprendizado Supervisionado tenham apresentado bons resultados na classificação de tráfego (BOUTABA et al., 2018), a necessidade de um conjunto pré-definido de classes de aplicação pode ser um problema, dada a variação nas aplicações da Internet ao longo do tempo. Além disso, o tempo necessário para a execução do processo de inferência de modelos de Aprendizado Supervisionado mais complexos pode ser proibitivo para casos de uso em que as decisões precisam ser feitas rapidamente. Entre os algoritmos de ML Supervisionado utilizados para classificação de tráfego figuram Florestas Aleatórias, Naïve Bayes e Support Vector Machines.

ML Não-Supervisionado. O Aprendizado de Máquina Não-Supervisionado consiste na identificação de padrões e estruturas em conjuntos de dados não-rotulados. Diferentemente do Aprendizado Supervisionado, em que se tem um conjunto de instâncias previamente rotuladas com as classes de interesse, o Aprendizado Não-Supervisionado visa descobrir as classes presentes nos dados através da identificação de padrões (FACELI et al., 2011). No contexto de classificação de tráfego, raramente se tem informações referentes a todas as aplicações presentes em uma rede, de modo que as técnicas de *clustering* podem ser utilizadas para encontrar padrões nos fluxos de pacotes e posteriormente identificar as aplicações correspondentes (BOUTABA et al., 2018). Um modelo de agrupamento poderia, por exemplo, ser capaz de identificar a emergência de novas aplicações na rede através

da detecção de novos agrupamentos, mas apresenta o problema de que os grupos identificados precisariam ser rotulados por um sistema externo. Além disso, os *clusters* em geral não mapeiam diretamente para classes de aplicações, sendo possível a existência de aplicações que ocupem múltiplos grupos sem dominar nenhum (NGUYEN; ARMITAGE, 2008b). Exemplos de algoritmos de Aprendizado Não-Supervisionado comumente utilizados para classificação de tráfego incluem K-Means, Maximização de Expectativa e Redes Neurais.

Métricas de avaliação. No contexto de classificação de tráfego, a avaliação de um modelo pode utilizar métricas específicas. Em especial, pode-se dividir a acurácia de um modelo entre acurácia por fluxo e acurácia por *byte* (NGUYEN; ARMITAGE, 2008a). A acurácia por fluxo se refere à quantidade de fluxos corretamente classificados pelo modelo dentre todos os fluxos do *dataset*. A acurácia por *bytes*, por outro lado, não distingue entre os fluxos específicos, mas mede a quantidade de *bytes* corretamente classificados. Se alguma das duas métricas é especialmente apropriada para um modelo dependerá do caso de uso específico, mas a acurácia por *bytes* pode ser fundamental para a classificação de tráfego, visto que os fluxos que formam a maior parte do tráfego são geralmente curtos e representam uma fração pequena dos *bytes* de fato trafegados sobre uma rede (ERMAN; MAHANTI; ARLITT, 2007). Se tratando de ML Não-Supervisionado, existem métricas específicas para avaliar a qualidade dos agrupamentos, tais como a Soma dos Erros Quadrados Intra-Cluster (WSS - *Within-Cluster Sum of Squared Errors*) e o Coeficiente de Silhueta. WSS consiste na distância total entre cada instância do *dataset* e o centróide do agrupamento vinculado a esta, de modo que um melhor agrupamento tende a minimizar este valor. O Coeficiente de Silhueta, por outro lado, visa expressar o quão coesos são os *clusters* e o quão bem eles dividem-se entre si. Para tanto, são calculados a coesão, que é a distância média entre um ponto e os outros pontos do mesmo agrupamento, e a dissimilaridade, que consiste na distância média entre um ponto e todos os pontos do *cluster* mais próximo. Com ambos os valores, é possível calcular o Coeficiente de Silhueta de modo que agrupamentos melhores tendem a maximizá-lo.

2.2.3 K-Means para Classificação de Tráfego

Dado o objetivo do presente trabalho de investigar a aplicação de Aprendizado de Máquina Não-Supervisionado para classificação de tráfego e K-Means estar entre as mais populares e eficientes técnicas neste contexto, a partir deste ponto é realizada uma

discussão mais detalhada do algoritmo.

K-Means é um algoritmo de Aprendizado de Máquina Não-Supervisionado de particionamento. Em ML Não-Supervisionado, particionamento é uma classe de algoritmos de *clustering* que executam a tarefa de decompôr um conjunto de instâncias em agrupamentos disjuntos baseados em alguma métrica de semelhança de seus atributos. Dadas n instâncias, o algoritmo particiona os dados em $k < n$ conjuntos com o objetivo de minimizar a similaridade inter-agrupamento e maximizar a similaridade intra-agrupamento (IKOTUN et al., 2023).

No contexto de classificação de tráfego, K-Means tende a demonstrar bons resultados (USAMA et al., 2019). Por ser um algoritmo de simples implementação e de baixo custo computacional, é bastante apropriado para casos de uso em que a utilização de recursos de processamento e armazenamento é uma preocupação importante, como é o caso em PDPs. Ademais, os centróides gerados pelo algoritmo podem ser combinados com o algoritmo *K Nearest Neighbors* com $k = 1$, resultando no Classificador de Centróide Mais Próximo, que classifica as instâncias de acordo com o rótulo do centróide mais próximo, um processo de inferência bastante simplificado.

Funcionamento. O algoritmo inicia criando um particionamento não ideal com k agrupamentos, sendo que as posições dos centróides dos *clusters* podem ser randômicas ou informadas como hiperparâmetros. Cada instância do *dataset* é então vinculada ao agrupamento cujo centróide esteja mais próximo, assim formando k agrupamentos. Após isso, cada centróide é recalculado como o ponto central entre as instâncias que formam o agrupamento correspondente e este processo será repetido iterativamente até que algum critério de convergência seja alcançado (JIN; HAN, 2010a). O Algoritmo 1 expressa o funcionamento de K-Means em psudeocódigo.

Algoritmo 1: Algoritmo K-Means

Entrada: K, número de clusters; D, um dataset com N pontos

Saída: Um conjunto de K clusters

início

    Inicialização dos clusters;
    repita
        para cada *ponto p em D* **faça**
            Encontre o centróide mais próximo e vincule p ao cluster correspondente;
        fim
        Atualize cada cluster calculando novos centróides usando a média de seus membros;
    até *Critério de fim de iteração satisfeito*;
    retorna *Resultado do agrupamento*

fim

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados trabalhos relacionados na área de classificação de tráfego no contexto de PDPs utilizando Aprendizado de Máquina. O principal objetivo deste capítulo é identificar as lacunas presentes na literatura que este trabalho visa preencher. Na Seção 3.1, são analisados trabalhos que realizam classificação de tráfego em PDP empregando técnicas de Aprendizado Supervisionado, enquanto que a Seção 3.2 discorre sobre o emprego de técnicas de *clustering* para o mesmo fim. Por fim, a Seção 3.3 oferece uma análise das lacunas identificadas.

3.1 Classificação de Tráfego em PDPs com ML Supervisionado

Xavier et al. (XAVIER et al., 2022) investigaram a viabilidade de mapear modelos de ML para PDPs utilizando a linguagem P4 para classificação de tráfego. O *framework* apresentado consiste em 3 componentes principais: um *Knowledge Plane*, que realiza a extração de *features* e o treinamento do modelo; um *Control Plane*, que realiza o mapeamento do modelo para um programa P4 e sua instalação no dispositivo físico e o *Data Plane*, no qual é executado o estágio de inferência. Além dos requerimentos usuais de Aprendizado de Máquina, o trabalho pretende construir um classificador que possa ser computado eficientemente em dispositivos programáveis, com pouco impacto na latência da rede. Portanto, o algoritmo utilizado é uma *Decision Tree*, que é mapeada na forma de um encadeamento de blocos *if-else*. Dois modelos são propostos, um pacote-a-pacote e um fluxo-a-fluxo, no qual informações sobre os fluxos são armazenadas em uma tabela de fluxos em memória. Para a validação do sistema, são utilizados dois cenários: detecção de intrusos e *Internet of Things*, no qual o objetivo é identificar o dispositivo responsável por um fluxo de pacotes. O sistema consegue bons resultados, mas apresenta uma série de limitações. Em especial, o mapeamento de Árvores de Decisão para cadeias *if-else* em código P4 que é posteriormente compilado e instalado na máquina implica que um novo programa precisará ser compilado e instalado quando o modelo precisar ser atualizado, o que pode não ser viável. Uma possível melhoria seria descrever o modelo como tabelas *match-action* que poderão ser atualizadas em tempo de execução, mas este trabalho não explora esta possibilidade.

Coelho e Schaeffer-Filho (COELHO; SCHAEFFER-FILHO, 2022) apresentaram a implementação de um sistema para identificação de fluxos referentes a diferentes classes

de ataques *Denial of Service* (DoS) utilizando Florestas Aleatórias em PDP. O modelo, previamente treinado *offline*, é mapeado como uma série de tabelas *match-action* com múltiplas tabelas por árvore, uma para cada nível de profundidade dos nodos. Cada linha da tabela corresponde a um nodo da árvore, com a ação vinculada estando na forma da comparação de um atributo e uma referência aos filhos deste nodo. A cada passo do processo de inferência, o atributo referente ao nodo é avaliado e a próxima tabela é invocada com o identificador do nodo apropriado. Para os nodos folha, a sua linha na tabela contém apenas a classificação entre "malicioso" e "legítimo" e sempre invoca a ação de classificar o fluxo. Cada uma das árvores é avaliada sequencialmente, até que todas tenham gerado uma classificação. Ao final do processo de inferência, os votos das árvores são contados e o fluxo é classificado com o rótulo mais votado. O sistema é avaliado utilizando diferentes números de árvores e *thresholds* de profundidade, demonstrando bons resultados em acurácia, precisão, *recall* e *F1 score*.

Lessa (LESSA, 2022) propôs o mapeamento de um modelo de rede neural para uma rede de *switches*, com cada dispositivo reproduzindo o processamento de um neurônio da rede, com o objetivo de realizar classificação de tráfego. O sistema utiliza a biblioteca Tensorflow para gerar modelos de Redes Neurais treinados que são mapeados para uma infraestrutura de *switches* programáveis na forma de programas P4. O mesmo programa é instalado em todos os *switches*, cujo funcionamento consiste no processamento dos valores de entrada e a difusão dos dados de saída para todos os *switches* da camada seguinte. Cada nodo conta com uma série de metadados que indicam o funcionamento do seu programa: um identificador, o número de estímulos esperados, a função de agregação, a função de ativação, *bias* e pesos. Além disso, são utilizados dois hospedeiros em ambas as extremidades da rede, com um conectado aos nodos de entrada e responsável por enviar pacotes com os dados de entrada e outro responsável por receber e decodificar a saída do modelo. O sistema apresenta em torno de 80% de acurácia de classificação, com a particularidade de que a acurácia é impactada proporcionalmente pela quantidade de bits utilizada na representação dos dados. Em especial, o trabalho consegue realizar classificações no ambiente de PDPs tão precisas quanto utilizando os mesmos modelos em ambientes tradicionais, demonstrando a viabilidade da implementação de redes neurais em PDP.

3.2 Classificação de Tráfego em PDPs com ML Não-Supervisionado

Embora a maior parte dos trabalhos realizados no contexto de aplicação de *Machine Learning* em Planos de Dados Programáveis debruce-se sobre o Aprendizado Supervisionado, a literatura ainda assim oferece exemplos de trabalhos que empregam *clustering* no mesmo contexto. Assim, dada a intenção deste trabalho de investigar a aplicação de Aprendizado Não-Supervisionado, é apresentada a seguir uma análise de trabalhos que utilizam técnicas deste paradigma.

Parizotto et al. (PARIZOTTO et al., 2023) apresentaram uma *survey* analisando trabalhos recentes que empregam Aprendizado de Máquina em Planos de Dados Programáveis. Foram analisados 36 trabalhos publicados entre 2014 e 2023 que utilizam modelos de Aprendizado de Máquina nos planos de dados de dispositivos de rede programáveis. Entre outras classificações, os trabalhos são divididos de acordo com a fase de ML que é relegada ao plano de dados programável, com 17 trabalhos tratando da fase de treinamento e 19 trabalhos utilizando o plano de dados para a fase de inferência. Da última categoria, apenas 3 utilizam aprendizado de máquina não-supervisionado, dos quais 2 utilizam K-Means: *Clustreams* (FRIEDMAN; GOAZ; ROTTENSTREICH, 2021) e *Do Switches Dream of Machine Learning?* (XIONG; ZILBERMAN, 2019), ambos discutidos a seguir. Embora o levantamento inclua um número notável de trabalhos que versam sobre classificação e análise de tráfego, não se encontram propostas investigando especificamente aplicações referentes a *streaming* de vídeo. Em geral, nota-se que os trabalhos analisados tendem a abordar classificação agnóstica quanto à aplicação ou, comumente, com foco em detecção de ataques.

Friedman, Goaz e Rottenstreich (FRIEDMAN; GOAZ; ROTTENSTREICH, 2021) apresentam o *Clustreams*, um sistema de agrupamento em planos de dados programáveis para classificação de tráfego agnóstica à aplicação. O sistema presume um modelo de agrupamento treinado *offline* que é carregado em *switches* programáveis. Interessantemente, a classificação utiliza como atributos apenas valores encontrados nos cabeçalhos dos pacotes, executando a classificação pacote-a-pacote, o que pode ser ineficiente para algumas aplicações. Para a etapa de inferência, o sistema utiliza *QuadTrees*, uma estrutura de dados para codificação de espaços através de divisão em quadrantes. Dados os centróides, o algoritmo considera um quadrante como pertencente a um *cluster* se o centróide deste *cluster* for o centróide mais próximo de todos os vértices do quadrante. Se não for o caso, o algoritmo subdivide o quadrante e repete o processamento recursi-

vamente até uma profundidade máxima pré-definida. É relevante o fato de que, dado o limite na profundidade do algoritmo, algumas regiões não são vinculadas a *cluster* algum, o que resulta em pacotes não classificados. Como forma de combater este problema, o sistema é capaz de encaminhar pacotes não identificados para o plano de controle onde poderão ser classificados utilizando técnicas tradicionais. Os resultados do trabalho indicam um impacto muito pequeno na latência da rede, enquanto que a razão de pacotes não classificados diminui significativamente com o aumento da profundidade das *QuadTrees*. Ademais, a solução proposta apresentou boa performance, com impacto muito pequeno na latência da rede.

Xiong e Zilberman (XIONG; ZILBERMAN, 2019) investigam o mapeamento de diferentes modelos de Aprendizado de Máquina para switches programáveis com o objetivo de realizar classificação de tráfego. O sistema é capaz de carregar diferentes algoritmos para o plano de dados para classificação, especificamente Árvores de Decisão, K-Means, Support Vector Machines e Naïve Bayes. O trabalho tem um interesse especial na utilização eficiente dos recursos dos *switches* na classificação, mostrando maior enfoque nestas métricas do que na acurácia das classificações. Assim como o trabalho anterior, este se debruça sobre classificação pacote-a-pacote, utilizando principalmente atributos extraídos dos cabeçalhos dos pacotes. O mapeamento dos modelos para a *pipeline* de processamento dos *switches* programáveis consiste na conversão destes em tabelas *match-action*, que definem regras de processamento para diferentes valores de atributo. Três métodos de mapeamento de K-Means são investigados: uma tabela por atributo, uma tabela por *cluster* e um terceiro mapeamento que cria tabelas para atributos e classes. O trabalho conclui que o mapeamento mais escalável para K-means é a criação de uma tabela por atributo, mas não apresenta uma análise mais profunda. Quanto à utilização de recursos, K-Means apresenta desempenho mediano, utilizando em média 5 tabelas e 44% da memória do switch, mas, novamente, não é apresentada uma discussão mais aprofundada.

Zhu e Zhang (ZHU; ZHANG, 2022) propuseram um modelo semi-supervisionado em que K-Means é utilizado para criar agrupamentos de fluxos de pacotes e uma Árvore de Decisão é aplicada para classificar as diferentes aplicações. Além disso, o trabalho propõe um algoritmo para armazenamento de atributos dos fluxos combinando *hash tables* e *Count-min sketch* com o objetivo de minimizar a utilização dos recursos do dispositivo. O *framework* proposto apresentou acurácia de classificação acima de 93%, com impacto mínimo na performance do dispositivo. O sistema foi avaliado utilizando diferentes clas-

ses de aplicações e demonstrou bons resultados especialmente para fluxos referentes a transferência de arquivos, email e *streaming*.

Mazzardo (MAZZARDO, 2019) propôs a utilização de Agrupamento Baseado em Limiar para classificação de tráfego em PDPs. O algoritmo utilizado é capaz de classificar fluxos à medida que chegam ao dispositivo, permitindo que o número de *clusters* aumente conforme o necessário em tempo de execução. O modelo é mapeado como uma série de tabelas *match-action* responsáveis pelo cálculo de características referentes a cada um dos fluxos e seu eventual agrupamento. Como a execução do algoritmo depende de laços, para os quais a linguagem P4 não oferece suporte, o sistema utiliza a tecnologia de reenvio de pacotes, enviando um pacote para o início da *pipeline* com *flags* ativadas de modo que possa continuar sendo processado. Esta técnica acaba tendo um impacto grande na latência da rede, com o tempo de processamento dos *switches* utilizando o modelo sendo duas vezes maior que seu tempo de processamento padrão. Foram utilizadas três classes de tráfego para a avaliação: DNS, VoIP e os *videogames* Quake 3 e Counter Strike. A taxa de acertos manteve-se acima de 70% na maioria dos casos, mas o sistema demonstra bastante dificuldade em classificar os fluxos mais complexos referentes aos *videogames*.

3.3 Análise dos Trabalhos Relacionados

Tabela 3.1: Comparação entre os Trabalhos Relacionados

Estudo	**Técnica**	**Aplicação**
Xavier et al. (2022)	Árvore de Decisão	Classificação de tráfego
Coelho e Schaeffer-Filho (2022)	Florestas Aleatórias	Identificação de ataques DoS
Lessa (2022)	Rede Neural	Classificação de tráfego
Friedman et al. (2021)	Sistema de *clustering* próprio	Agnóstico
Xiong e Zilberman (2019)	K-Means, entre outras	Agnóstico
Zhu e Zhang (2022)	K-Means semi-supervisionado	Classificação de tráfego
Mazzardo (2019)	Agrupamento Baseado em Limiar	Classificação de tráfego

Fonte: Autor

Duas relevantes lacunas se apresentam na literatura que se refere à classificação de tráfego utilizando Aprendizado de Máquina, sobretudo em PDPs: a exploração reduzida de técnicas de *clustering* e a carência de maiores investigações quanto à classificação de *streaming* de vídeo. A maior parte dos trabalhos utilizando ML fazem uso de modelos como Árvores de Decisão ou Redes Neurais, enquanto que poucos implementam soluções baseadas em agrupamento. Ademais, o algoritmo K-Means, que apenas ocasionalmente

figura na literatura, tende a apresentar resultados bons e é de baixa complexidade, se mostrando um bom candidato para a classificação de tráfego em PDPs. Embora a maior parte do tráfego na Internet atualmente seja referente a aplicações de *streaming* de vídeo (SANDVINE, 2023), poucos trabalhos consideram esta classe de aplicações, com a maior parte se dedicando a classificar ataques e invasões ou desenvolvendo sistemas de classificação agnósticos à aplicação. Embora existam casos em que classes referentes a aplicações de *streaming* de vídeo são utilizadas para validação do modelo, como em (ZHU; ZHANG, 2022), os sistemas de classificação são desenvolvidos de forma agnóstica. Sendo assim, há a ausência de trabalhos que se debrucem exclusivamente sobre a classificação de fluxos de *streaming* de vídeo.

Visto isso, a abordagem proposta é a única que emprega K-Means para a classificação específica de fluxos gerados por aplicações de *streaming* de vídeo. Neste trabalho, investiga-se a aplicabilidade desta técnica no contexto de Planos de Dados Programáveis. Através do desenvolvimento e validação do sistema K-Flix, pretende-se analisar a eficiência de modelos de *clustering* para a distinção de tráfego referente a *streaming*.

4 O SISTEMA K-FLIX

Neste capítulo, é apresentado o sistema proposto, K-Flix, descrevendo-se sua estrutura como um todo e detalhando-se as características de cada componente. Na Seção 4.1 é apresentada uma visão geral da abordagem proposta, discorrendo-se sobre seu funcionamento em linhas gerais e introduzindo cada módulo do sistema e seu funcionamento. Após, na Seção 4.2, é discutida a operação de K-Means em PDPs e o processo de mapeamento dos *clusters* para o plano de dados. A Seção 4.3 propõe um novo algoritmo para a aproximação da normalização de *features* em PDPs. Além disso, a Seção 4.4 apresenta as técnicas utilizadas para o processamento de múltiplos fluxos correspondentes e a aproximação das médias dos atributos. Ao fim, na Seção 4.5, a arquitetura é detalhada, descrevendo-se cada um dos componentes do sistema.

4.1 Visão Geral

K-Flix é um sistema para classificação de tráfego de internet com o principal objetivo de identificar a presença de tráfego correspondente a aplicações de *streaming* de vídeo. Dados os recursos limitados disponíveis para a computação em PDPs, surge a necessidade de um processo de inferência eficiente e escalável que possa ser empregado em infraestruturas de redes reais sem um grande impacto ao funcionamento da rede. Para este fim, o sistema objetiva utilizar os centróides gerados por um modelo K-Means para realizar a classificação dos fluxos de tráfego, o que configura um processo de inferência simplificado.

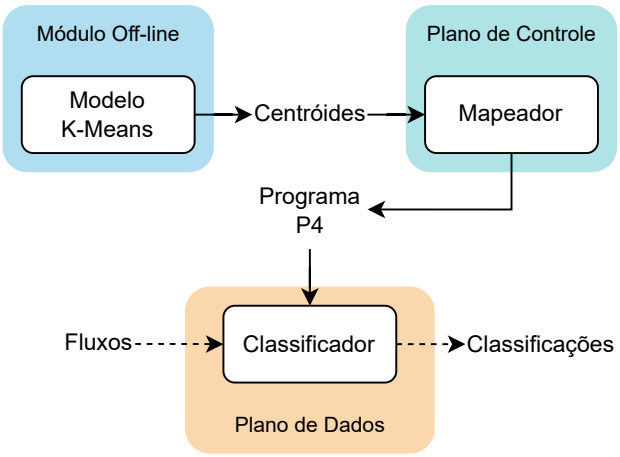
Dado o objetivo de minimizar-se a utilização de recursos, apenas o processo de inferência do classificador acontece na infraestrutura da rede. O processamento dos *datasets* e a geração dos *clusters* acontece *off-line*, de modo que o sistema receba apenas as coordenadas dos centróides gerados. Sendo assim, K-Flix pode ser facilmente atualizado com novos particionamentos ao longo do tempo ou até mesmo adaptado para empregar diferentes modelos de agrupamento.

É implementado um Classificador de Centrõide Mais Prõximo utilizando os *clusters* gerados pelo modelo K-Means, uma abordagem já explorada em trabalhos anteriores (ZHANG; SUN, 2010). Desta forma, o processo de inferência consiste meramente no cálculo das distâncias entre os centróides pré-computados e os atributos dos fluxos analisados. Este método para classificação baseia-se em operações aritméticas simples, o que

o torna especialmente apropriado para o emprego em PDPs.

Após o mapeamento do modelo para estruturas apropriadas para *switches* programáveis, ele é utilizado para distinguir fluxos unidirecionais de tráfego referentes a aplicações de *streaming* de vídeo. Um fluxo unidirecional de tráfego é definido como uma tupla de 5 componentes: um endereço IP de origem, um endereço IP de destino, uma porta de origem, uma porta de destino e um protocolo de transporte, tipicamente TCP ou UDP. Um fluxo, assim, é composto por todos os pacotes que, ao longo de uma conexão, partem de um hospedeiro de origem para outro de destino. Dado o emprego da classificação de tráfego baseada em atributos estatísticos, são calculadas e acumuladas métricas referentes a um fluxo a cada novo pacote analisado pertencente a este. Tais métricas são utilizadas no classificador para a identificação de fluxos referentes a *streaming* de vídeo com o objetivo de realizar uma classificação correta utilizando tão poucos pacotes quanto possível. Esta classificação, ainda, é atualizada ao longo do tempo de vida de uma conexão ao que novos pacotes chegam ao *switch* e os atributos do fluxo são recalculados.

Figura 4.1: Visão Geral do Sistema K-Flix



Fonte: Autor

O sistema proposto compõe-se de três subsistemas: um módulo *off-line*, um plano de controle e um plano de dados, ilustrados na Figura 4.1.

Módulo *off-line*. É responsável pela leitura de capturas de tráfego e a geração dos *datasets* correspondentes para o treinamento do modelo. Além disso, realiza a análise dos atributos para gerar os parâmetros necessários para a normalização. Neste módulo, o modelo K-Means é treinado a partir dos dados analisados de modo a gerar os centróides que serão utilizados pelos estágios posteriores do sistema.

Plano de Controle. Mapeia os valores gerados anteriormente para estruturas apropriadas para instalação em *switches* programáveis. Constrói um Classificador de Centróide Mais Próximo que é então mapeado para uma sequência de tabelas *match-action*, além de gerar o código P4 que será executado no plano de dados.

Plano de Dados. O último estágio do processo é responsável por receber o classificador gerado e realizar o processo de inferência em tempo de execução. Neste módulo, os fluxos capturados pelo *switch* programável são analisados com o objetivo de extrair os atributos estatísticos relevantes e executar a classificação.

4.2 K-Means em Planos de Dados Programáveis

Como discutido anteriormente, o principal estágio do processamento em *switches* programáveis acontece através de uma *pipeline* de tabelas *match-action* que mapeiam valores de entrada a ações definidas como blocos de código. Dadas as limitações da linguagem P4, sobretudo a impossibilidade de definição de laços e o conjunto pequeno de operações aritméticas suportadas, não é possível executar modelos de Aprendizado de Máquina diretamente no plano de dados. Sendo assim, se faz necessário um processo de mapeamento da lógica do modelo para um formato apropriado para o dispositivo programável.

Em um classificador baseado em K-Means, o algoritmo de inferência consiste no cálculo da similaridade entre a instância a ser classificada e cada um dos centróides das partições e a posterior escolha do agrupamento com o centróide mais próximo à entrada. Utilizando a Distância Euclidiana como medida de similaridade, o cálculo da distância entre uma entrada x e um centróide C^i é definido pela equação 4.1. Considerando k centróides, a inferência consiste em calcular cada uma das distâncias $\{D_1, D_2, ..., D_k\}$ e escolher o *cluster* cujo centróide resulte na menor distância.

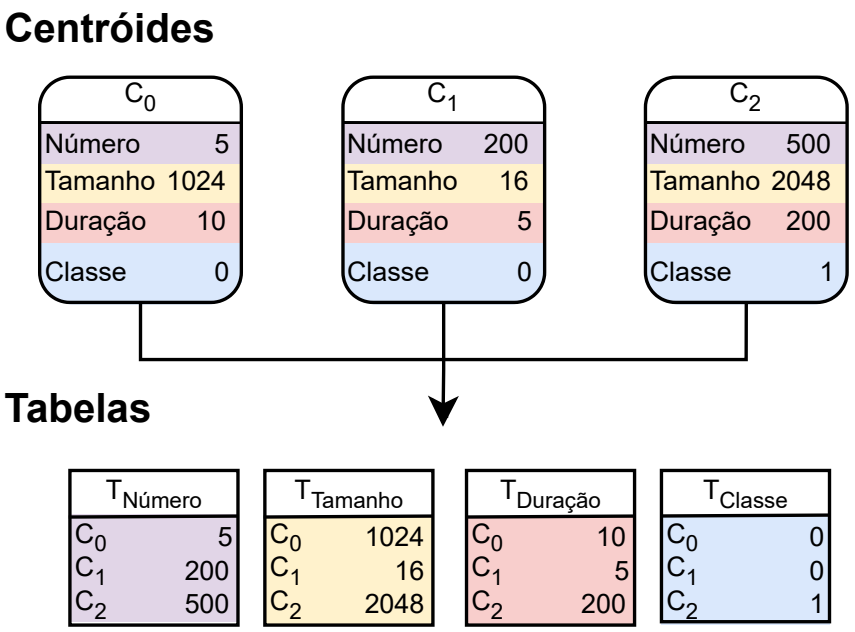
$$D_i = \sqrt{(x_1 - C_1^i)^2 + (x_2 - C_2^i)^2 + .. (x_n - C_n^i)^2} \tag{4.1}$$

Dado um conjunto de n atributos, K-Flix implementa um mapeamento baseado na geração de valores de distância intermediários ao longo de $n + 1$ estágios. A cada estágio do processamento, são calculadas as distâncias entre um dado atributo na entrada e cada valor do mesmo atributo nos centróides, gerando um vetor de distâncias unidimensional. No estágio seguinte, calculam-se as distâncias referentes ao atributo seguinte e

assim sucessivamente, acumulando os valores no vetor unidimensional de distâncias. É importante notar que, se tratando de valores e distâncias positivos e maiores que zero, a operação de radiciação da Distância Euclidiana pode ser ignorada, utilizando-se apenas as distâncias quadradas. No último estágio, a classificação consiste da escolha do agrupamento vinculado à menor distância calculada.

Baseando-se no *framework* proposto em (ZHENG et al., 2022), o algoritmo é implementado sob a forma de n tabelas *match-action* intermediárias, uma para cada atributo, e uma tabela final que executa a classificação. Cada tabela intermediária consiste em uma única entrada que se refere à função de cálculo das distâncias e inclui os valores do atributo correspondente em cada um dos centróides. Em tempo de execução, quando uma destas tabelas é aplicada, é calculada a distância entre o atributo correspondente no fluxo analisado e em cada centróide. Ao longo da *pipeline*, as distâncias são acumuladas em um registrador em memória. A última tabela recebe como entrada os rótulos correspondentes a cada centróide, compara todas as k distâncias entre si, encontra a menor e rotula o fluxo com a classe correspondente.

Figura 4.2: Mapeamento de K-Means para Tabelas *Match-action*

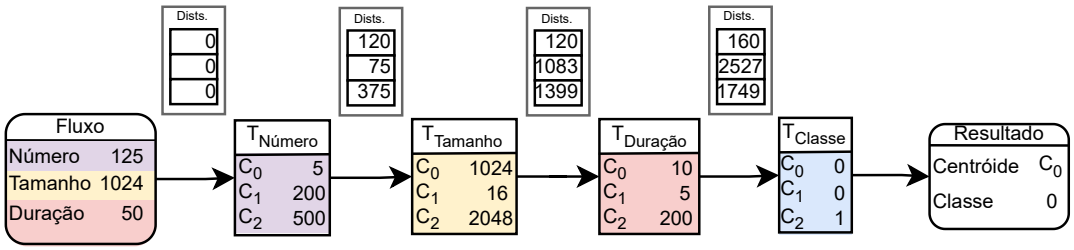


Fonte: Autor

Um mapeamento utilizando esta técnica está ilustrado na Figura 4.2. Neste exemplo, são considerados apenas três atributos: número de pacotes, tamanho médio de pacotes e duração do fluxo. Além disso, três centróides são definidos com apenas o último

rotulado com a classe de interesse. Assim, são geradas quatro tabelas ao total: uma correspondendo a cada atributo e uma tabela final que realiza a classificação. É importante notar que o aspecto de *matching* das tabelas é irrelevante nesta aplicação visto que sempre que uma tabela é aplicada, a ação correspondente deve ser invocada. Portanto, se omitem as chaves de *matching* e os nomes das ações das ilustrações de tabelas *match-action*. Na Figura 4.3, encontra-se a execução da inferência neste exemplo aplicada a um fluxo arbitrário. Como nota-se, o vetor de distâncias intermediárias é atualizado ao que cada tabela é aplicada, com as distâncias de cada centróide sendo somadas ao total. O processo inteiro se dá por sequências de subtrações e somas, além de comparações entre inteiros na etapa final, o que configura um processo de inferência bastante simples e apropriado para o contexto de PDPs.

Figura 4.3: Execução do Processo de Inferência



Fonte: Autor

4.3 Aproximação de Normalização em PDPs

Modelos de ML que utilizam a Distância Euclidiana são muito suscetíveis a atributos que variam em escalas diferentes (PATEL; MEHTA, 2011). Por exemplo, atributos referentes a tamanho de pacote e duração do fluxo estariam em unidades diferentes que apresentariam ordens de grandeza distintas. Esta situação se torna problemática ao passo de que o processo de inferência do modelo consiste no cálculo de distâncias referentes a todos os atributos em conjunto. Neste caso, as *features* que variam em escalas maiores teriam uma maior importância no processo de classificação do que as demais, o que impactaria gravemente a qualidade do modelo.

Em Aprendizado de Máquina, a normalização consiste no mapeamento de diversos atributos para uma mesma escala de modo que todos tenham a mesma importância na decisão final. Uma das técnicas mais simples de normalização é a Normalização Min-

Max, que faz o mapeamento dos atributos para uma escala definida a partir dos valores máximo e mínimo de cada um. O algoritmo consiste na aplicação da Equação 4.2 para cada valor de cada atributo X. A normalização deve ser aplicada tanto ao *dataset*, antes do treinamento do modelo, quanto à cada nova instância a ser classificada.

$$x_{i\_norm} = \frac{x_i - X_{min}}{X_{max} - X_{min}} \tag{4.2}$$

Dado que a normalização consiste necessariamente em uma divisão, a sua implementação em PDPs se torna complexa, visto a ausência desta operação. Como forma de solucionar este problema, é proposto um novo algoritmo para a aproximação da normalização. Embora a divisão não esteja disponível, é possível realizar *bit shifts* que são equivalentes a multiplicações ou divisões por potências de 2. O algoritmo baseia-se na ideia de aproximar a divisão da Normalização Min-Max através de uma série de divisões por potências de 2. É composto por duas etapas: a geração dos parâmetros de normalização a partir dos valores originais dos atributos e a aplicação da normalização através de *bit shifts* consecutivos. O primeiro estágio do algoritmo, que calcula as potências de 2 necessárias à aproximação da normalização, pode ser executado *offline* como um *bootstrap* de um sistema de ML em PDPs, sendo que somente a normalização dos valores deve acontecer no dispositivo programável. Assim, este algoritmo possibilita a aproximação da Normalização Min-Max em Planos de Dados Programáveis, visto que a operação de *bit shift* tende a ser bastante eficiente e é comumente implementada em *hardware*.

4.3.1 Geração dos Parâmetros

O funcionamento deste estágio encontra-se no Algoritmo 2. Ele recebe como entrada os valores máximo e mínimo do atributo, um valor representando o limite superior da normalização desejada e um valor *threshold* definindo o número máximo de potências de 2 que devem ser consideradas. Inicialmente, é calculado o divisor da Equação 4.2, que é então dividido pelo limite máximo da escala. Se utiliza o valor resultante para calcular o recíproco do divisor, ou seja, o resultado de 1 dividido por este. O algoritmo utiliza do fato de que a divisão por um número é equivalente à multiplicação por seu recíproco. No caso do recíproco incluir uma parte inteira em seu valor, esta parte representa uma multiplicação e será retornada como o MultFactor. O recíproco do divisor é então iterativamente aproximado através de recíprocos de potências de 2, com todas as potências

apropriadas para a aproximação guardadas em uma lista. Ao final, a lista de potências e o fator de multiplicação são retornados.

Algoritmo 2: Cálculo dos Parâmetros de Normalização

Entrada: FeatMax; FeatMin; ScaleMax, MaxPower

Saída: Potencias, potências de 2 que aproximam a divisão; MultFactor, fator para multiplicação

início

Divisor = FeatMax - FeatMin;
ScaledDiv = Divisor/ScaleMax;
DivReciproco = 1/ScaledDiv;
Multfactor = parte inteira de DivReciproco;
Decimal = parte decimal de DivReciproco;
Potencias = [];
PotenciaAtual = 1;
ValorAtual = 0;
repita
  Temp = ValorAtual + $1/(2^{\text{PotenciaAtual}})$;
  se *Temp <= Decimal* **então**
    Adicione PotenciaAtual a Potencias;
    ValorAtual = Temp;
  fim
  Incremente PotenciaAtual;
até *PotenciaAtual > MaxPower*;
retorna *Potencias, MultFactor*

fim

Com o objetivo de simplificar a compreensão do funcionamento da etapa de geração de parâmetros, é apresentada a computação de um exemplo na Tabela 4.1. Considera-se um atributo cuja Normalização MinMax na escala tradicional consista da divisão do valor do atributo por 25, de modo que recíproco do divisor será 0,04. Estes valores, embora bastante simples e não representantes de situações típicas encontradas em Aprendizado de Máquina, foram escolhidos com o objetivo de simplificar o entendimento do algoritmo. Em cada linha da tabela, são demonstrados os valores referentes a uma iteração do laço definido no Algoritmo 2.

Tabela 4.1: Valores Durante Geração de Parâmetros

Potência	*Reciproco*	*ValorAtual*	*Temp*	*Potencias*	*Temp <= Reciproco*
			...		
4	0,04	0	0,0625	$\emptyset$	Falso
5	0,04	0	0,03125	$\emptyset$	Verdadeiro
6	0,04	0,03125	0,046875	$\{5\}$	Falso
7	0,04	0,03125	0,0390625	$\{5\}$	Verdadeiro
8	0,04	0,0390625	0,04296875	$\{5, 7\}$	Falso
9	0,04	0,0390625	0,041015625	$\{5, 7\}$	Falso
10	0,04	0,0390625	0,0400390625	$\{5, 7\}$	Falso
			...		

Fonte: Autor

4.3.2 Aplicação da Normalização

O pseudo-código expressando o funcionamento deste estágio encontra-se no Algoritmo 3. Utilizando-se das saídas do passo anterior, é calculada a normalização aproximada de um valor do atributo. Inicialmente, calcula-se o dividendo da Equação 4.2. Para cada uma das potências calculadas no passo anterior, acumula-se o valor deste dividendo após um *bit shift* de um número de *bits* igual à potência, aproveitando-se do fato de que a multiplicação por um número X equivale a uma série de multiplicações por outros números cuja soma resulte em X. Ao fim, ainda acumula-se o valor da multiplicação do dividendo pelo MultFactor calculado anteriormente.

Algoritmo 3: Aproximação da Normalização Min-Max

```
Entrada: ValorFeature; MinFeature; Potencias; MultFactor
Saída: Valor do atributo normalizado
início
    Dividendo = ValorFeature - MinFeature;
    ValorNorm = 0;
    repita
        ValorNorm = ValorNorm + (Dividendo » potencia);
    até Para cada Potencia em Potencias;
    ValorNorm = ValorNorm + (Dividendo*MultFactor);
    retorna ValorNorm
fim
```

Apresenta-se uma comparação entre a normalização aproximada realizada com o algoritmo proposto e os resultados esperados obtidos com Normalização MinMax sobre diferentes atributos na Tabela 4.2. É considerado um *threshold* de 16 potências de 2 e uma escala de valores desejada entre 0 e 1024. O uso desta escala no contexto de PDPs é proposital: utilizar um limite superior maior que 1 é equivalente a multiplicar o dividendo da equação por este valor. Assim, utilizar uma potência de 2 suficientemente grande como limite superior equivale a *bit shifts* para a esquerda, o que evita que os *bits* mais significativos sejam perdidos no processo de divisão. Na tabela, os valores resultantes da aproximação são comparados com os retornados pela equação original multiplicada por 1024. Como nota-se, o algoritmo proposto retorna uma boa aproximação dos valores normalizados na grande maioria dos casos, apresentando uma média de erro bastante baixa. Com isso dito, dada a limitação de se utilizarem apenas valores inteiros, o algoritmo realiza uma discretização dos valores normalizados, originalmente contínuos. No exemplo apresentado, a normalização resulta em 1024 possíveis valores inteiros para cada atributo, um parâmetro que pode ser modificado de acordo com as características dos dados e as necessidades do sistema.

Tabela 4.2: Aplicação da Normalização Aproximada em Diferentes Atributos

Valor	*Mínimo*	*Máximo*	*Normalizado*	*Aproximação*	*Erro*
75	5	96	787.6923	787	0.6923
500	0	1000	512	510	2
500	300	8420	25.2217	25	0.2217
1024	0	65535	16.002	16	0.002
400	0	300	1365.3333	1363	2.3333
200	0	2000	102.4	101	1.4
0	0	1000	0	0	0

Fonte: Autor

4.4 Demais Técnicas Integradas ao Projeto

Nesta seção, são apresentados dois princípios de projeto propostos em trabalhos anteriores que foram integrados ao sistema K-Flix.

Aproximação de Médias. Em Aprendizado de Máquina, é comum que atributos

relevantes a um problema sejam médias de outros valores que precisam ser calculadas antes que o processo de inferência possa ser aplicado. Assim como a problemática da normalização discutida anteriormente, calcular a média é desafiador em um contexto em que não há suporte a operações de divisão. Baseando-se no algoritmo proposto no projeto BACKORDERS, é implementado um sistema de aproximação de médias utilizando *bit shifts* consecutivos (COELHO; SCHAEFFER-FILHO, 2022). O algoritmo consiste no cálculo de uma soma aproximada dos valores considerando um número de valores igual a uma potência de 2. Sendo assim, o cálculo da média, que consiste na divisão da soma dos valores pelo seu número, se torna uma divisão por potência de 2 e pode ser implementada através de *bit shifts*. O algoritmo apresenta ótimos resultados com pouco *overhead* computacional.

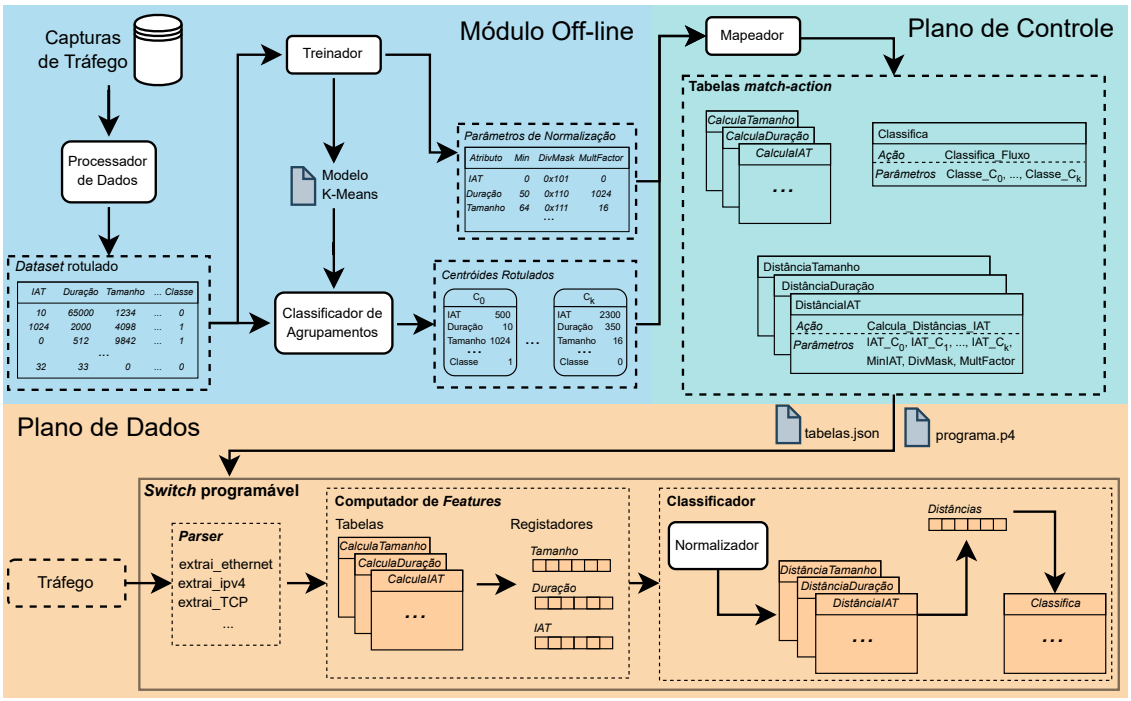
Classificação de Fluxos Concorrentes. *Switches* programáveis contam com *chips* de memória de capacidade de armazenamento relativamente pequena e, em geral, não mantém estado entre um pacote e outro. Esta limitação se torna um problema em aplicações que precisam monitorar múltiplos fluxos ao longo de seu tempo de vida. O trabalho Turboflow mitiga tal problema ao utilizar os registradores disponíveis nos *switches* programáveis para formar uma *hash table* (SONCHACK et al., 2018). Um registrador em P4 consiste em um *array* indexável mantido em memória. Utilizando-se de um algoritmo de *hash*, é possível mapear os atributos que representam um fluxo para uma posição de um registrador. Assim, todos os pacotes pertencentes a um mesmo fluxo são capazes de ler e escrever das mesmas posições de memória, o que possibilita a manutenção e o acúmulo de valores referentes a múltiplos fluxos concorrentes. K-Flix implementa esta funcionalidade utilizando os seguintes valores como identificadores de cada fluxo: endereços IP de origem e destino, números de portas de origem e destino e protocolo de transporte utilizado.

4.5 Arquitetura

O sistema K-Flix é dividido em uma série de componentes modulares, cada um com propósito e funcionamento específicos, organizados em uma *pipeline*. A Figura 4.4 apresenta a organização de tais componentes. Nesta seção, é discutida em detalhes a arquitetura do sistema. A Subseção 4.5.1 discorre sobre os componentes do módulo *off-line*, que tem como objetivo gerar os centróides que serão a base do classificador. Na Subseção 4.5.2, é apresentado o funcionamento do Mapeador, módulo presente no plano de con-

trole e responsável por programar o plano de dados a partir das informações geradas pelo módulo anterior. Por fim, a Subseção 4.5.3 explica o funcionamento dos componentes que se encontram no plano de dados e tem como objetivo principal realizar a classificação de tráfego em tempo de execução.

Figura 4.4: Arquitetura do Sistema K-Flix



Fonte: Autor

4.5.1 Módulo *Off-line*

A *pipeline* do sistema K-Flix se inicia em componentes externos à rede, acumulando a parte mais custosa do processamento de modo a minimizar a carga imposta à infraestrutura de rede.

Processador de Dados. Seu funcionamento baseia-se na análise de capturas de tráfego com o objetivo de extrair os atribuytos que serão utilizados para o treinamento do modelo K-Means posteriormente. O Processador de Dados recebe como entrada capturas de tráfego, identifica os fluxos presentes nestas e extrai os atributos referentes a cada um. Além disso, os fluxos são rotulados como *streaming* de vídeo ou não de acordo com uma classificação anterior parametrizável. Ao fim da execução, os dados analisaqdos são retornados sob a forma de um *dataset* rotulado aplicável a um modelo de Aprendizado de Máquina.

Treinador. Utilizando-se dos atributos gerados anteriormente, o componente Treinador desempenha duas principais funções: o treinamento de um modelo K-Means e o cálculo dos parâmetros necessários para o algoritmo de aproximação da normalização. Os atributos dos fluxos são normalizados e passam pelo processo de agrupamento do modelo K-Means. Os centróides dos agrupamentos são armazenados para uso posterior. Além disso, os valores máximo e mínimo de cada atributo são utilizados para calcular as potências que aproximam a normalização. A lista de tais potências é codificada como uma *string* binária em que um valor 1 representa a presença da potência correspondente na aproximação e um valor 0, a sua ausência. Também são coletados os valores mínimos de cada atributo, necessários para a normalização das instâncias. Por fim, os centróides e os parâmetros de normalização são enviados ao módulo seguinte

Classificador de Agrupamentos. K-Means é um algoritmo de ML Não-Supervisionado, o que indica que não há um processo de classificação automática. Desta forma, para que seja possível utilizar os centróides gerados pelo algoritmo como ferramenta de classificação, é necessário vincular um rótulo a cada um deles. A implementação escolhida para este fim foi a de votação, na qual o rótulo vinculado a um dado *cluster* é aquele que forma a maioria dentre todas as instâncias agrupadas nele. Sendo assim, o Classificador de Agrupamentos analisa as instâncias agrupadas anteriormente e rotula os centróides apropriadamente, resumindo umnúmero arbitrariamente grande de fluxos em outro consideravelmente menor de centróides. Uma outra possível implementação para esse módulo seria a utilização de um modelo de Aprendizado de Máquina Supervisionado para realizar a classificação dos centróides, mas a complexidade envolvida na implementação de tal solução foge ao escopo deste trabalho. Este componente produz um conjunto de centróides rotulados com cada um consistindo das coordenadas do centróide no espaço de atributos e a classe vinculada a este. Juntamente com os parâmetros de normalização gerados anteriormente, os centróides rotulados são enviados ao Mapeador no plano de controle.

4.5.2 Plano de controle

O plano de controle do sistema K-Flix consiste de um único componente, o Mapeador. Apropriadamente, sua função é mapear as informações geradas pelos componentes anteriores para estruturas apropriadas para o plano de dados. Utilizando-se dos centróides rotulados e dos parâmetros de normalização, esse componente constrói algoritmicamente duas estruturas distintas: o código do programa P4 e os comandos para população das

tabelas *match-action*. A partir da lista de atributos considerados, são geradas as tabelas de cálculo dos atributos que formarão o Computador de *Features* no plano de dados. Por fim, são utilizados os atributos dos centróides e os parâmetros de normalização para criar as tabelas de classificação de tráfego que formarão o componente Classificador.

4.5.3 Plano de Dados

No plano de dados se encontram os principais componentes do Sistema K-Flix. Tais componentes analisam o tráfego, acumulam dados sobre os fluxos e classificam eles enquanto *streaming* de vídeo ou não. A classificação é indicada pela adição de um cabeçalho extra nos pacotes, o Cabeçalho K-Flix. Em tal cabeçalho, localizado entre o Ethernet e o IP, são incluídos três campos com valores inteiros: IsVideo, com um valor binário indicando se o fluxo se refere a uma aplicação de vídeo, Cluster, com o número do agrupamento referente a esta classificação, principalmente para depuração do modelo, e Ethertype, um valor copiado do campo de mesmo nome do cabeçalho Ethernet que indica o tipo do cabeçalho seguinte. A classificação na forma de um cabeçalho extra se dá tanto pela simplicidade de implementação quanto pela oportunidade de classificação contínua dos fluxos ao longo de sua execução.

Parser. Como é típico em programas P4, é implementado um desserializador sob a forma de uma FSM. Este desserializador tem como função a extração dos dados presentes nos cabeçalhos do pacote, indicando quais protocolos são esperados e em qual ordem. Na versão atual do sistema, oferece-se suporte a pacotes IPv4 que contenham como protocolo de transporte TCP ou UDP.

Computador de *Features*. Ao que pacotes passam pelo *switch*, o Computador de *Features* calcula os atributos de cada fluxo e os acumula em registradores em memória indexados por *hashes* da tupla de identificadores de cada fluxo. Ao processar o primeiro pacote pertencente a um fluxo, são armazenados valores iniciais que serão recalculados a cada novo pacote. No caso de uma *hash collision*, indicando que um novo fluxo tomará o lugar de um anterior em memória, os dados acumulados são reiniciados e o fluxo atual é simplesmente considerado um fluxo novo. Dada a funcionalidade de classificação contínua do sistema, é razoável presumir que o fluxo anterior já terá sido classificado antes de ser substituído em memória por um novo.

A extração de atributos é implementada na forma de uma série de tabelas *match-action*, cada uma responsável pela computação de um atributo específico. *Features* como

os *inter-arrival times* e a duração dos fluxos são computadas a partir de metadados dos pacotes oferecidos pelo dispositivo. Outros atributos, como o tamanho da janela TCP ou o tamanho do pacote em *bytes* são recuperados de campos dos cabeçalhos. Ademais, atributos derivados destes valores, tais como máximos, mínimos e médias, são calculados e atualizados a cada nova extração de atributos. O conjunto concreto de *features* implementadas no protótipo atual é apresentado na Tabela 5.3.

Classificador. É implementado um Classificador de Centróide Mais Próximo baseando-se na abordagem discutida na Seção 4.2. O classificador é definido na forma de uma sequência de tabelas *match-action*, uma para cada atributo, que iterativamente calculam as distâncias entre os centróides e o fluxo sendo analisado e acumulam estas distâncias em um vetor em memória. Por fim, uma tabela de classificação analisa as distâncias calculadas e encontra o centróide vinculado à menor distância. Assim, o fluxo recebe como classificação o rótulo do centróide mais próximo. Após a classificação, é adicionado o Cabeçalho K-Flix ao pacote, que é enviado para uma das portas de saída do *switch* de acordo com as regras de encaminhamento atuais.

Antes que a classificação possa ser executada de fato, os atributos calculados pelo componente anterior devem ser normalizados. O Normalizador implementa o segundo estágio do algoritmo de aproximação da normalização, no qual a lista de potências de 2 é percorrida e divisões sequenciais são executadas na forma de *bit shifts*. Dado que a lista de potências de 2 para um atributo é injetada no sistema sob a forma de uma máscara binária, o Normalizador percorre cada *bit* da máscara aplicando as divisões de acordo com os valores. De fato, a tabela de classificação de cada *feature* invoca o Normalizador antes de calcular as distâncias entre os centróides e a instância atual, garantindo que todos os cálculos serão realizados sobre valores normalizados.

5 IMPLEMENTAÇÃO E VALIDAÇÃO

Neste capítulo, é apresentada a implementação do sistema e a sua posterior avaliação. Na Seção 5.1 é detalhado o protótipo construído para os experimentos, discorrendo em especial sobre os hiperparâmetros do modelo e os atributos selecionados. A Seção 5.2 apresenta o ambiente de testes construído para o protótipo, detalhando os *datasets* e tecnologias de *software* utilizados. Por fim, os resultados dos experimentos são demonstrados e discutidos na Seção 5.3.

5.1 Prototipação

Para a avaliação do projeto foi desenvolvido um protótipo inicial. Os componentes externos e do plano de controle foram implementados utilizando a linguagem Python contando com bibliotecas específicas. Os componentes do plano de dados, por outro lado, foram desenvolvidos utilizando a linguagem P4.

Utilizou-se o pacote *pyshark* para Python para implementar o componente Processador de Dados, que foi desenvolvido com a capacidade de analisar múltiplos arquivos *.pcap*, identificar seus fluxos, extrair as *features* necessárias e salvar os dados gerados em arquivos *.csv*. Também nesse componente foi implementada a identificação de fluxos de *streaming* de vídeo de modo a gerar dados rotulados. Neste protótipo, um fluxo unilateral é considerado como *streaming* de vídeo se respeita as seguintes condições:

- O endereço IP de origem é um IP público, indicando um *download*
- Tem um número mínimo de 25 pacotes
- Tem uma duração mínima de 10 segundos
- Provém de uma aplicação de *streaming* de vídeo

Estas condições foram escolhidas baseando-se na detecção de *elephant flows* (KNOB et al., 2017) de modo a garantir um grau aceitável de certeza quanto à identificação dos fluxos como *video streaming*.

Para treinar o modelo K-Means e avaliar seu desempenho, foram selecionados dois repositórios de capturas de tráfego disponíveis publicamente: *ITC-Net-Blend-60* (BAYAT et al., 2024) e *YouTube, Netflix, Web dataset for Encrypted Traffic Classification* (SHAMSIMUKHAMETOV et al., 2021). Ambos os *datasets* são compostos de capturas de poucos minutos rotuladas pela aplicação sendo utilizada, simplificando a identificação de trá-

fego referente a aplicações de *video streaming*. Além disso, *ITC-Net-Blend-60* consiste de capturas realizadas em um dispositivo Mobile, enquanto o segundo *dataset* se trata de capturas realizadas em um ambiente Desktop.

A partir desses *datasets*, o Processdor de Dados foi utilizado para filtrar capturas referentes tanto a aplicações específicas de *video streaming* quanto a outras aplicações. No total, o *dataset* gerado pelo componente somou 8885 fluxos, dos quais 4642 foram identificados como *streaming* de vídeo segundo os parâmetros citados anteriormente. A distribuição de tais fluxos nas aplicações analisadas pode ser encontrada nas Tabelas 5.1 e 5.2, divididas entre o cenário *mobile* e o cenário *desktop*. As aplicações acessadas em um ambiente *Desktop* incluem Netflix e Twitch, enquanto que as aplicações de vídeo Android consideradas incluem Filimo e Telewebion, dois aplicativos de *streaming* de vídeo similares à Netflix. A aplicação Youtube foi incluída em duas versões, tanto *mobile* quanto *desktop*. Das capturas referentes a essas aplicações, foram filtrados apenas os fluxos de vídeo, enquanto que os demais fluxos do *dataset* são provenientes das seguintes aplicações e *websites*: Stack Overflow, Wikipedia e Ali Express no cenário *desktop* e Outlook, Google Play Store e Google Maps no cenário *mobile*. Em ambos os cenários foram incluídos também fluxos referentes à aplicação Dropbox.

Tabela 5.1: *Dataset* de Treino - Mobile

Aplicação	*Número de Fluxos*
Youtube	401
Filimo	244
Telewebion	656
Outras aplicações	2360
Total fluxos de Vídeo	1301
Total	3661

Fonte: Autor

No componente Treinador, os arquivos *csv* provenientes do Gerador de *Datasets* são carregados e concatenados. Utilizou-se o modelo K-Means da biblioteca *scikit-learn* em Python para realizar o treinamento do modelo de agrupamento. Para a definição do hiperparâmetro K, referente ao número de *clusters*, utilizou-se o Elbow Method aplicado à WSS dos agrupamentos. Na Figura 5.1, apresenta-se o gráfico resultante do cálculo do WSS para números de *clusters* variando entre 2 e 12. Para cada K, o algoritmo foi treinado 30 vezes com o WSS sendo calculado para cada iteração e, por fim, foram computadas as médias dos valores. O Elbow Method é então aplicado sobre as médias para escolher um

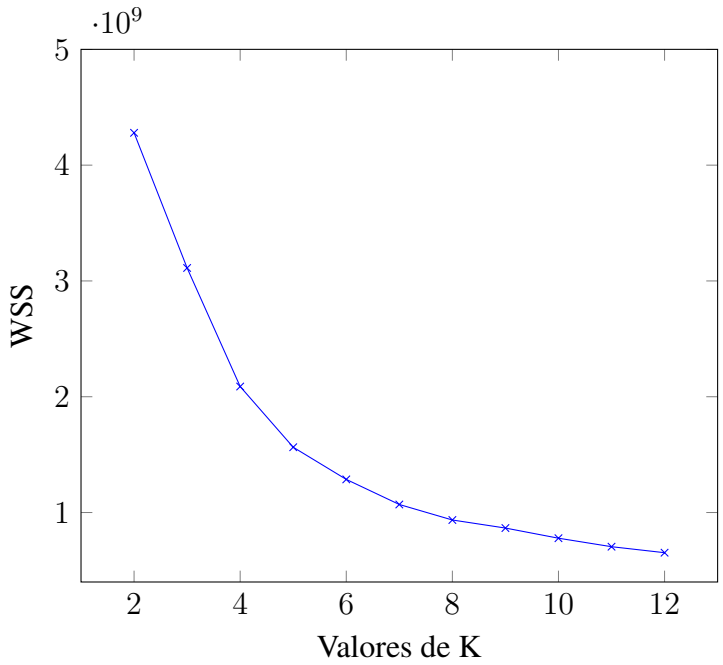
Tabela 5.2: *Dataset* de Treino - Desktop

Aplicação	*Número de Fluxos*
Youtube	2665
Netflix	633
Twitch	43
Outras aplicações	1883
Total fluxos de Vídeo	3341
Total	5224

Fonte: Autor

número de *clusters* que maximize a qualidade dos agrupamentos sem causar *overfitting* (HUMAIRA; RASYIDAH, 2020). Tal escolha se dá identificando-se o ponto na curva onde o aumento da métrica analisada deixa de acontecer de maneira dramática. Dados os resultados obtidos, fixou-se o valor de K em 7.

Figura 5.1: Elbow Method para Seleção de K



Fonte: Autor

Também em Python se implementaram o Classificador de Agrupamentos e a geração dos parâmetros de normalização conforme descrito no capítulo anterior. O Gerador do Plano de Dados utiliza os centróides rotulados e os parâmetros de normalização para escrever o programa P4 e os comandos de população das tabelas *match-action*.

Para o protótipo, foram selecionadas 15 *features* com base em análises de atributos

relevantes para classificação de tráfego e, especificamente, *video streaming* (ZHANG et al., 2013) (DONG; ZHAO; JIN, 2017). A listagem das *features* selecionadas se encontra na Tabela 5.3.

Tabela 5.3: Atributos Selecionados

Nome	*Descrição*	*Derivados*
PktCount	Número de Pacotes	-
PktLength	Tamanho do pacote em *bytes*	Soma, Max, Min e Média
Iat	*Inter-arrival times*	Soma, Max, Min e Média
FlowDuration	Duração do fluxo	-
InitialWindow	Janela TCP do primeiro pacote	-
Window	Janela TCP	Soma, Max, Min e Média

Fonte: Autor

Para a funcionalidade de manter atributos de múltiplos fluxos concorrentemente, implementou-se um registrador em memória para cada uma das *features* com 65535 posições, um número arbitrariamente grande escolhido para evitar *hash collisions*. Cada um dos atributos é representado como um valor de 32 *bits*, com exceção dos valores relacionados a *timestamps*, que são representados em 48 *bits*. Para cada *feature*, são implementadas duas tabelas *match-action*: uma para preencher os registradores com os valores iniciais de cada atributo ao primeiro pacote de um fluxo e uma segunda para atualizar tais valores ao que os pacotes subsequentes são analisados.

Para o algoritmo de aproximação da normalização, convencionou-se uma escala de 2048 possíveis valores inteiros, de modo que os valores normalizados fossem representados por apenas 11 *bits*, mas contassem com suficiente granularidade. Para o limite de precisão do algoritmo, foi escolhido o número de 32 potências de 2, uma precisão que se mostrou necessária dadas as grandes escalas dos atributos selecionados.

É implementado um registrador em memória com um valor para cada *cluster*, com cada valor representando a distância entre a instância sendo analisada e o centróide correspondente à posição. Dada a escala de normalização de 2048, utilizou-se uma largura de 11 *bits* para cada valor de distância. Para cada *feature*, é implementada uma tabela *match-action* responsável por calcular as distâncias referentes ao seu atributo e acumulá-las no registrador de distâncias. Após a aplicação de todas as tabelas de cálculo de distâncias, uma tabela adicional lê os valores armazenados no vetor e decide qual o centróide mais próximo, vinculando o seu rótulo para o fluxo atual. No total, o Classificador é composto por 16 tabelas, uma para cada *feature* e uma tabela adicional para a classificação.

5.2 Avaliação Experimental

Esta seção discorre sobre a metodologia empregada para a avaliação do protótipo definido anteriormente. A Subseção 5.2.1 apresenta o ambiente de testes utilizado, enquanto que a Subseção 5.2.2 analisa o *workload* que foi aplicado sobre o sistema.

5.2.1 Ambiente de testes

Como ambiente virtual no qual o sistema foi testado, utilizou-se uma máquina virtual Ubuntu com 6GB de memória RAM alocada executando sobre um processador AMD Ryzen 5 de seis *cores*. Nesse ambiente, o programa P4 foi compilado para uma instância do Bmv2, um *switch* programável virtual para desenvolvimento de *software* para PDPs. Foram configuradas duas interfaces de rede virtuais, *veth0* e *veth1*, para simular as interfaces do dispositivo. Desta forma, o *switch* consome pacotes enviados à interface *veth0* e encaminha-os após o processamento à interface *veth1*. Para o envio de pacotes à interface *veth0*, foi utilizado o programa *tcpreplay*, capaz de reproduzir tráfego de internet a partir de arquivos de captura mantendo os tempos de envio relativos de cada pacote. Para capturar os pacotes enviados à outra interface com os resultados das classificações, empregou-se WireShark que, além de capturar tráfego, é também capaz de exportar as capturas para arquivos para análise posterior. Foi desenvolvido em Lua um *dissector* para Wireshark de modo a habilitar o programa a reconhecer o Cabeçalho K-Flix. A configuração do *switch*, incluindo a instalação do programa P4 e a população das tabelas, utilizou-se da biblioteca P4Runtime através de *scripts* em Python.

Dado que o *workload* consistiria de grandes fluxos formados por dezenas de milhares de pacotes, tal ambiente de testes simples foi construído com o objetivo de minimizar os problemas relacionados à latência. Embora seja um *switch* virtual, o Bmv2 não é otimizado para grandes volumes de tráfego e pode ter problemas de atrasos ou perda de pacotes se a demanda da rede for alta demais. Por esses motivos decidiu-se não utilizar uma rede virtual Mininet, como é comum em trabalhos envolvendo *switches* virtuais. Além disso, a implementação oferece maior controle sobre a configuração das interfaces e dos dispositivos virtuais.

5.2.2 *Datasets* de testes

Dos dois *datasets* citados anteriormente, foram selecionadas capturas que não haviam sido utilizadas no processo de treinamento do modelo. Tais capturas referem-se a diferentes aplicações tanto de *streaming* de vídeo quanto miscelâneas. Dentre as capturas Android, foram selecionadas capturas refentes a Filimo, Telewebion e Google Maps. Dentre as capturas Desktop, selecionaram-se Netflix e Amazon. Mais uma vez, ambas as versões de Youtube foram incluídas. Especialmente, se incluíram capturas da aplicação Dropbox em ambas as versões. Dado que Dropbox se trata de um *software* de genrenciamento de arquivos, suas capturas incluem *downloads* e *uploads*, de modo que seria relevante testar a capacidade do sistema em diferenciar estas de *streaming* de vídeo.

No total, foram enviados 1275 fluxos ao *switch* virtual, dos quais apenas 70 se caracterizariam como *streaming* de vídeo a partir das condições definidas na Seção 5.1. Esta disparidade entre as duas classes se dá pelo fato de que os fluxos referentes a *streaming* de vídeo tendem a ser longos e consistir de uma grande quantidade de pacotes, assim ocupando a maior parte das capturas. O número total de pacotes enviados através do *switch* foi de 281443, somando um total de aproximadamente 244MB. As distribuições dos fluxos no *dataset* de testes podem ser vistas nas Tabelas 5.4 e 5.5.

O envio dos pacotes de cada captura foi feito sequencialmente com pausas de poucos segundos entre uma aplicação e a seguinte, de modo que os pacotes resultantes encaminhados pelo *switch* pudessem ser capturados e exportados para arquivos *pcap* onde seriam posteriormente analisados.

Tabela 5.4: *Dataset* de Teste - Mobile

Aplicação	*Fluxos*	*Caracterizam Streaming de Vídeo*
Youtube	114	9
Filimo	49	5
Telewebion	108	34
Google Maps	172	0
Dropbox	102	0
Total	545	48

Fonte: Autor

Tabela 5.5: *Dataset* de Teste - Desktop

Aplicação	*Fluxos*	*Caracterizam Streaming de Vídeo*
Youtube	106	12
Netflix	151	10
Amazon	263	0
Dropbox	210	0
Total	730	22

Fonte: Autor

5.3 Resultados

Após a aplicação dos *workloads* de teste supracitados, o tráfego encaminhado pelo *switch* foi capturado na forma de arquivos *.pcap*. Como citado anteriormente, o sistema K-Flix oferece a capacidade de distinguir fluxos de *streaming* de vídeo através do campo binário *isVideo* presente no Cabeçalho K-Flix que é adicionado ao pacote apõs a classificação. Na implementação testada, este valor é 1 se o fluxo for classificado como *streaming* de vídeo e 0 caso contrário.

Utilizando os mesmos parâmetros de identificação de fluxos de *streaming* de vídeo apresentados na Seção 5.1, foi identificada a *Ground Truth* dos fluxos presentes no *workload*. Comparando esta informação com a classificação realizada pelo modelo, foram calculadas acurácia, acurácia por *bytes*, precisão, *recall* e F1 para diferentes subconjuntos do *workload* de testes de modo a analisar a eficácia do modelo em cenários variados.

Acurácia. É a proporção de classificações corretas feitas pelo modelo. É calculada pela divisão do total de classificações corretas pelo total de instâncias consideradas. Em um *dataset* desbalanceado, como é o caso dos testes realizados com o protótipo, pode não ser expressiva o suficiente, se fazendo necessária a consideração das demais métricas.

Acurácia por *bytes*. Como apresentado na Seção 2.2, é um valor análogo à acurácia, mas que considera o número de *bytes* classificados corretamente ao invés do número de fluxos. Pode ser uma métrica especialmente apropriada para aplicações referentes a *streaming* de vídeo, visto que os fluxos que se objetiva classificar são, em sua grande maioria, *elephant flows* e, portanto, contém grande parte dos *bytes* que transitam na rede. Desta forma, é interessante medir o quão bem o sistema consegue classificar tais fluxos em particular.

Precisão. O cálculo da precisão objetiva expressar a proporção de classificações positivas corretas. No sistema avaliado, uma classificação positiva se refere a um fluxo classificado como *streaming* de vídeo. Para tanto, seu cálculo é a divisão do número de verdadeiros

positivos pelo número total de classificações positivas.

Recall. A partir do número de instâncias da classe positiva no *dataset*, o Recall pretende expressar quantas instâncias foram classificadas corretamente. Seu cálculo se dá pela divisão do número de verdadeiros positivos da classificação pelo número de instâncias positivas no *dataset*.

F1. É um *score* baseado nos valores de Precisão e Recall que busca expressar a qualidade geral do modelo. O seu cálculo é apresentado na Equação 5.1

$$F1 = \frac{2 * precisao * recall}{precisao + recall} \tag{5.1}$$

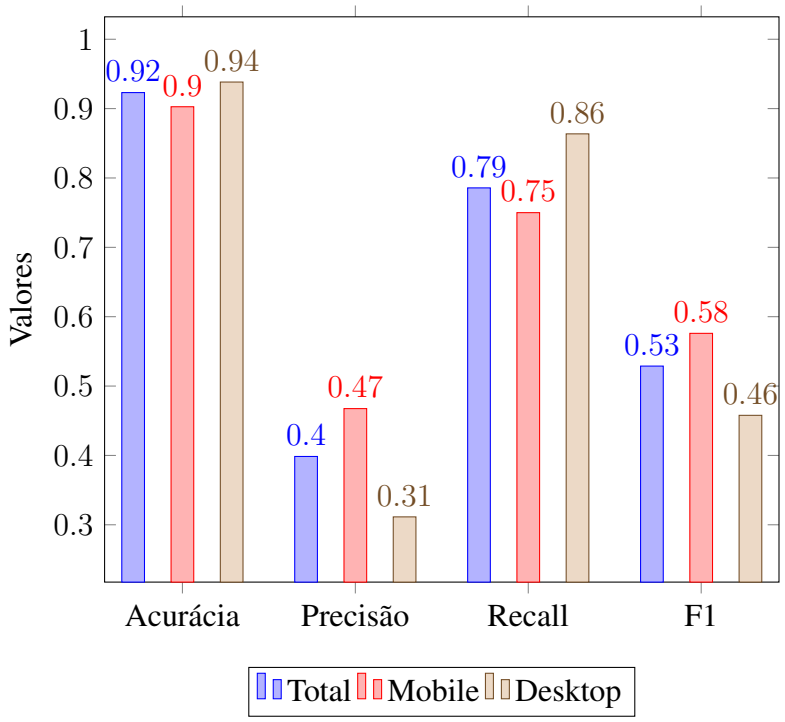
Nas subseções seguintes, são apresentadas as métricas obtidas considerando-se diferentes subconjuntos do *dataset* de testes, de modo a avaliar com maior granularidade a performance do sistema proposto. Na Subseção 5.3.1, se apresentam as métricas gerais, calculadas considerando-se todos os fluxos presentes no *workload*. A Subseção 5.3.2, por sua vez, discorre sobre as métricas obtidas considerando-se apenas os fluxos pertencentes às aplicações de *streaming* de vídeo incluídas no *dataset*.

5.3.1 Resultados Gerais

Considerando todos os 1275 fluxos do *workload* de testes, a acurácia obtida foi de 0.92, a precisão foi de 0.4, o *recall* foi calculado em 0.79 e o F1, em 0.53. A *byte accuracy* considerando o *dataset* completo foi de 0.41. Em geral, os resultados obtidos foram satisfatórios, mas indicam a necessidade de aprimoramentos do sistema no futuro. Em especial, o baixo valor de precisão indica a presença considerável de falsos positivos, isto é, fluxos que foram classificados como *streaming* de vídeo, mas não cumprem as condições definidas anteriormente para serem considerados como tal. A Figura 5.2 apresenta uma comparação entre as métricas obtidas no resultado geral e as referentes apenas aos subgrupos de aplicações Mobile e Desktop.

Analisando os resultados, nota-se uma performance bastante similar nos dois subgrupos, com a Precisão sendo relativamente baixa e resultando em um valor diminuído de F1. Os resultados baixos podem ser vinculados à considerável presença de falsos positivos entre as classificações. No cenário Mobile, de um total de 77 fluxos classificados como *video streaming*, 41 foram falsos positivos. No cenário Desktop, os resultados são ainda mais expressivos, com 42 falsos positivos em um total de 61 classificações positi-

Figura 5.2: Métricas em Diferentes Cenários



Fonte: Autor

vas. Por outro lado, analisar as classificações negativas indica uma performance melhor, com apenas 15 falsos negativos de um total de 1137.

Por outro lado, observando-se apenas o subgrupo de aplicações não relacionadas a *video streaming*, é possível notar uma proporção pequena de falsos positivos. De fato, a maior proporção de falsos positivos por aplicação neste subgrupo foi de menos de 10% e ocorreu nos fluxos da aplicação Google Maps. Tais resultados indicam que a maior parte dos falsos positivos reportados se deu em fluxos que pertencem justamente a aplicações de *video streaming*, demonstrando que o sistema foi capaz de identificar as assinaturas de tais aplicações com sucesso.

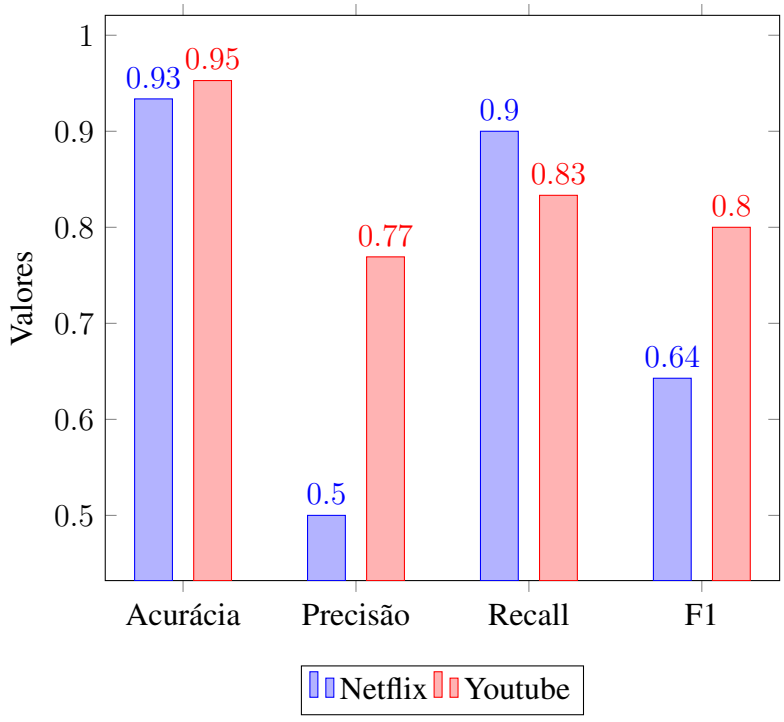
5.3.2 Aplicações de Vídeo

Dado o objetivo do presente projeto de investigar a classificação de tráfego referente a aplicações de *streaming* de vídeo, esta subseção apresenta uma análise do subgrupo de fluxos formado pelo tráfego referente às aplicações de *streaming* de vídeo analisadas. As Figuras 5.3 e 5.4 apresentam as métricas resultantes das aplicações Desktop e Mobile, respectivamente.

Em geral, os resultados obtidos foram bastante promissores. Notam-se as melhores métricas com a versão Desktop do Youtube e o aplicativo Telewebion, enquanto que as métricas mais baixas se referem à versão Android do Youtube. Quanto à *byte accuracy* para estas aplicações, o valor total foi de 0.40. No *dataset* utilizado, os fluxos de *video streaming* compunham a maior parte dos *bytes* trafegados pela rede, de modo que um pequeno número de falsos negativos é capaz de afetar dramaticamente a métrica de *byte accuracy*. Em especial, isso ocorreu com os fluxos referentes aos aplicativos Filimo e Telewebion. Não considerando os fluxos das duas aplicações, a *byte accuracy* sobe para 0.99, um resultado bastante expressivo.

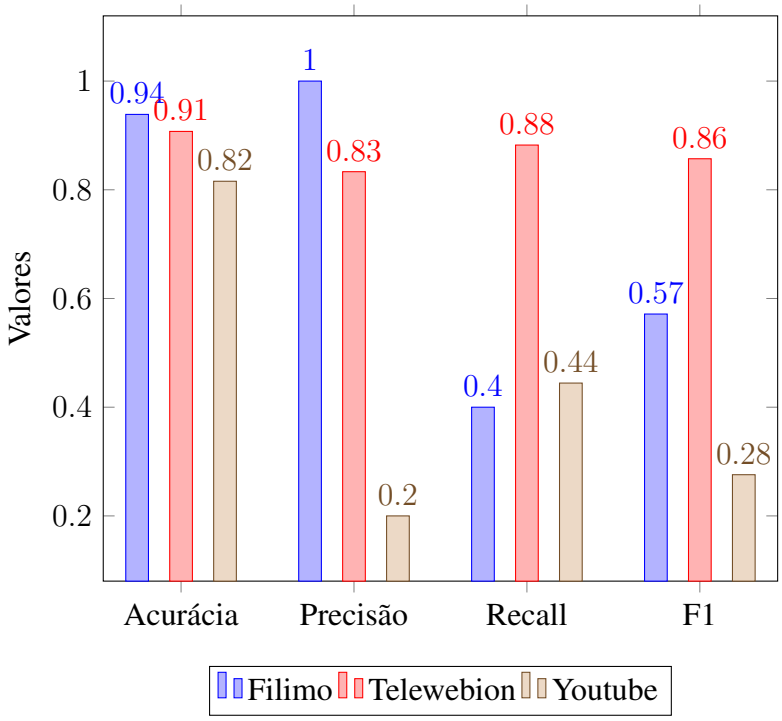
Uma análise das proporções de falsos positivos nas aplicações de vídeo revela um número maior de tais classificações neste grupo do que nas aplicações miscelâneas, principalmente na versão Mobile do Youtube, que tem uma proporção de falsos positivos de 14%. A maior recorrência de falsos positivos em aplicações de vídeo indica a capacidade do modelo de identificar satisfatoriamente as assinaturas de tais aplicações, mas também a sua dificuldade em diferenciar fluxos *upstream* e *downstream*. Os baixos índices de Precisão apresentados pelo protótipo, principalmentes nos fluxos referentes ao Youtube Android, podem ser diretamente conectados a tal limitação.

Figura 5.3: Métricas em Aplicações de Vídeo Desktop



Fonte: Autor

Figura 5.4: Métricas em Aplicações de Vídeo Mobile



Fonte: Autor

6 CONCLUSÃO

Neste capítulo, é apresentada a conclusão deste trabalho. Na Seção 6.1 é apresentado um resumo das contribuições realizadas por este trabalho. Por fim, a Seção 6.2 discute propostas de trabalhos futuros.

6.1 Resumo de Contribuições

Neste trabalho, foi proposto K-Flix, um sistema de classificação de tráfego referente a *streaming* de vídeo para planos de dados programáveis. O sistema implementa um Classificador de Centróide Mais Próximo baseado em agrupamentos gerados por K-Means, mapeando o classificador para uma estrutura própria para *switches* programáveis composta de um programa P4 e uma *pipeline* de tabelas *match-action*. Com o classificador devidamente mapeado, são calculadas e normalizadas as *features* relevantes de cada fluxo analisado a cada novo pacote. Tais atributos são então utilizados em uma sequência de tabelas *match-action* para calcular as distâncias entre o fluxo e cada um dos centróides gerados anteriormente, com uma última tabela responsável por identificar o centróide mais próximo e vincular ao fluxo o rótulo correspondente. Assim, implementa-se um classificador semi-supervisionado com um processo de inferência simplificado.

Desenvolveu-se um protótipo do sistema em um ambiente simulado e executaram-se testes com o objetivo de avaliar a qualidade das classificações. Dados os bons resultados obtidos nos testes, acredita-se que este trabalho prova o potencial da utilização de técnicas de *clustering* em PDPs, sendo possível alcançar altos índices de acurácia. Além disso, a capacidade de modelos de ML Não-Supervisionado de resumir grandes *datasets* em números comparativamente baixos de centróides é de grande relevânciapara o desenvolvimento de novas soluções em PDPs, visto as limitações de recursos presentes nesse contexto.

Ademais, foi proposto, implementado e avaliado um novo algoritmo para aproximação da normalização de *features* em PDPs. As técnicas de normalização consistem no mapeamento de valores de atributos para uma escala pré-definida, de modo que todas as *features* variem na mesma ordem de grandeza. Para modelos de ML suscetíveis à escala dos atributos, como é o caso do modelo K-Means, a normalização é fundamental para o correto funcionamento do algoritmo. Porém, dada a ausência de operações de divisão e laços em linguagens próprias para *switches* programáveis, a implementação da norma-

lização neste contexto é dificultada. O algoritmo apresentado propõe-se a aproximar a operação de divisão necessária à normalização Min-Max através de uma sequência de divisões por potências de 2, implementadas como *bit shifts*. Embora o algoritmo seja relativamente simples e ainda possa ser otimizado, as aproximações geradas por ele são de qualidade considerável, tornando possível a implementação da normalização em tempo de execução em *switches* programáveis

6.2 Trabalhos Futuros

Como propostas de trabalhos futuros, pretende-se otimizar diversos aspectos do sistema. Em especial, a utilização dos recursos de memória é bastante elevada, de modo que o sistema ainda não é factível de implementação em dispositivos programáveis reais. Em geral, *switches* programáveis têm severas limitações quanto à utilização e ao acesso à memória, inclusive impondo limites à quantidade de vezes que uma posição de memória pode ser acessada durante o processamento de um pacote (PARIZOTTO, 2024). Desta forma, a implementação atual do sistema deve ser otimizada antes que se possa utilizar em uma aplicação real.

Com o objetivo de otimizar a utilização dos recursos de memória, se planeja realizar um estudo mais aprofundado sobre os padrões de tráfego na Internet de modo a definir com mais clareza quantos fluxos precisariam ser considerados concorrentemente. Além disso, também se faz necessária uma análise melhor quanto à seleção das *features* necessárias para a identificação de *video streaming*, de modo a otimizar o número de atributos que devem ser mantidos em memória. Por exemplo, *Principal Component Analysis* e *Genetic Algorithm* são técnicas que podem ser aplicadas sobre um conjunto de atributos para reduzir a dimensionalidade ou identificar os atributos mais relevantes (ADHAO; PACHGHARE, 2020). Por fim, o componente Classificador deverá ser refatorado com o objetivo de minimizar o número de acessos à memória necessários para realizar uma classificação.

Quanto à qualidade do classificador proposto, também se faz necessário o estudo e implementação do cálculo de atributos que não foram considerados na atual versão do sistema. Por exemplo, um atributo aparentemente relevante para os fluxos de *streaming* de vídeo é o número de Pontos de Pico de Payload (PPP - *Payload Peak Points*), isto é, pontos ao longo do tempo de vida de um fluxo em que o tamanho dos pacotes atinge um máximo local (DONG; ZHAO; JIN, 2017). Além disso, também pretende-se realizar testes mais

diversos com o protótipo, analisando os resultados obtidos com diferentes conjuntos de atributos e valores de hiperparâmetros. Ademais, pretende-se implementar a filtragem de *elephant flows* de modo a somente investir os recursos necessários à classificação destes. Dadas as características próprias das aplicações de *video streaming*, ter a capacidade de ignorar *mice flows*, isto é, fluxos pequenos que transportam uma quantidade menor de *bytes*, será uma grande otimização do modelo. Além de possibilitar uma utilização mais eficiente dos recursos do *switch*, a implementação desta funcionalidade melhorará a qualidade das classificações, visto que fluxos menores não serão considerados e, portanto, não poderão ser classificados como falsos positivos.

Por fim, ainda propõe-se a investigação da utilização de diferentes modelos de Aprendizado Não-Supervisionado para a geração dos centróides no qual baseia-se o sistema K-Flix. Embora K-Means seja um modelo bastante popular, ele apresenta uma série de limitações, como o seu viés em identificar agrupamentos esféricos e a alta suscetibilidade a ruído e *outliers* (IKOTUN et al., 2023). No contexto de classificação de tráfego, o *hard clustering* realizado por K-Means, isto é, o vínculo de uma instância a um único agrupamento, pode gerar resultados ruins ao que fluxos de diferentes aplicações podem apresentar comportamentos bastante similares (BOUTABA et al., 2018).

Dadas as limitações citadas, o estudo de algoritmos alternativos se faz necessário. O algoritmo *K-Medoids*, por exemplo, usa as medianas dos *clusters*, isto é, pontos existentes no *dataset*, como centróides ao invés de calculá-los através da média, o que o torna mais robusto quanto à presença de *outliers* (JIN; HAN, 2010b). Outras variações do algoritmo visam resolver o viés por agrupamentos esféricos, causado principalmente pelo uso da Distância Euclidiana (IKOTUN et al., 2023). Por exemplo, (CHOKNIWAL; SINGH, 2016) utiliza *Gaussian Mixture Models* com distância de Mahalanobis para identificar agrupamentos elípticos. Outros trabalhos incluem formulações para identificar *clusters* de formas arbitrárias ou com intersecções (IKOTUN et al., 2023). A implementação de K-Flix com um algoritmo de *clustering* mais robusto pode aumentar significativamente a qualidade das classificações geradas pelo sistema.

REFERÊNCIAS

ADHAO, R.; PACHGHARE, V. Feature selection using principal component analysis and genetic algorithm. **Journal of Discrete Mathematical Sciences and Cryptography**, v. 23, p. 595–602, 02 2020.

AMARAL, P. et al. Machine learning in software defined networks: Data collection and traffic classification. In: **2016 IEEE 24th International Conference on Network Protocols (ICNP)**. [S.l.: s.n.], 2016. p. 1–5.

AZAB, A. et al. Network traffic classification: Techniques, datasets, and challenges. **Digital Communications and Networks**, 2022. ISSN 2352-8648. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2352864822001845>.

BAYAT, M. et al. Itc-net-blend-60: a comprehensive dataset for robust network traffic classification in diverse environments. **BMC Research Notes**, v. 17, 06 2024.

BERNAILLE, L.; TEIXEIRA, R. Implementation issues of early application identification. In: FDIDA, S.; SUGIURA, K. (Ed.). **Sustainable Internet**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 156–166. ISBN 978-3-540-76809-8.

BOSSHART, P. et al. P4: programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul 2014. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2656877.2656890>.

BOSSHART, P. et al. Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 43, n. 4, p. 99–110, aug 2013. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2534169.2486011>.

BOUTABA, R. et al. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. **Journal of Internet Services and Applications**, v. 9, 05 2018.

BUDIU, M.; DODD, C. The p416 programming language. **SIGOPS Oper. Syst. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 1, p. 5–14, sep 2017. ISSN 0163-5980. Available from Internet: <https://doi.org/10.1145/3139645.3139648>.

CHOKNIWAL, A.; SINGH, M. Faster mahalanobis k-means clustering for gaussian distributions. In: **2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [S.l.: s.n.], 2016. p. 947–952.

COELHO, B.; SCHAEFFER-FILHO, A. Backorders: using random forests to detect ddos attacks in programmable data planes. In: **Proceedings of the 5th International Workshop on P4 in Europe**. New York, NY, USA: Association for Computing Machinery, 2022. (EuroP4 '22), p. 1–7. ISBN 9781450399357. Available from Internet: <https://doi.org/10.1145/3565475.3569074>.

DONG, Y.-n.; ZHAO, J.-j.; JIN, J. Novel feature selection and classification of internet video traffic based on a hierarchical scheme. **Comput. Netw.**, Elsevier North-Holland, Inc., USA, v. 119, n. C, p. 102–111, jun 2017. ISSN 1389-1286. Available from Internet: <https://doi.org/10.1016/j.comnet.2017.03.019>.

ERMAN, J.; ARLITT, M.; MAHANTI, A. Traffic classification using clustering algorithms. In: **Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data**. New York, NY, USA: Association for Computing Machinery, 2006. (MineNet '06), p. 281–286. ISBN 159593569X. Available from Internet: <https://doi.org/10.1145/1162678.1162679>.

ERMAN, J.; MAHANTI, A.; ARLITT, M. Byte me: A case for byte accuracy in traffic classification. In: . [S.l.: s.n.], 2007. p. 35–38.

FACELI, K. et al. **Inteligência artificial: uma abordagem de aprendizado de máquina**. [S.l.]: LTC, 2011.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: an intellectual history of programmable networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 2, p. 87–98, apr 2014. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2602204.2602219>.

FRIEDMAN, R.; GOAZ, O.; ROTTENSTREICH, O. Clustreams: Data plane clustering. In: **Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)**. New York, NY, USA: Association for Computing Machinery, 2021. (SOSR '21), p. 101–107. ISBN 9781450390842. Available from Internet: <https://doi.org/10.1145/3482898.3483356>.

HAUSER, F. et al. A survey on data plane programming with p4: Fundamentals, advances, and applied research. **Journal of Network and Computer Applications**, v. 212, p. 103561, 2023. ISSN 1084-8045. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1084804522002028>.

HUMAIRA, H.; RASYIDAH, R. Determining the appropiate cluster number using elbow method for k-means algorithm. In: . [S.l.: s.n.], 2020.

IKOTUN, A. M. et al. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. **Information Sciences**, v. 622, p. 178–210, 2023. ISSN 0020-0255. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0020025522014633>.

JIN, X.; HAN, J. K-means clustering. In: ____. **Encyclopedia of Machine Learning**. Boston, MA: Springer US, 2010. p. 563–564. ISBN 978-0-387-30164-8. Available from Internet: <https://doi.org/10.1007/978-0-387-30164-8_425>.

JIN, X.; HAN, J. K-meoids clustering. In: ____. **Encyclopedia of Machine Learning**. Boston, MA: Springer US, 2010. p. 564–565. ISBN 978-0-387-30164-8. Available from Internet: <https://doi.org/10.1007/978-0-387-30164-8_425>.

KALJIC, E. et al. A survey on data plane flexibility and programmability in software-defined networking. **IEEE Access**, v. 7, p. 47804–47840, 2019.

KNOB, L. A. D. et al. Mitigating elephant flows in sdn-based ixp networks. In: **2017 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.: s.n.], 2017. p. 1352–1359.

LESSA, J. v. F. **Explorando redes neurais em planos de dados programáveis para classificação de tráfego**. Monografia (Trabalho de conclusão de graduação) — Universidade Federal do Rio Grande do Sul, Porto Alegre, BR-RS, 2022.

MAZZARDO, G. C. **Desenvolvimento de um sistema de classificação de tráfego de rede em planos de dados programáveis com base na linguagem P4**. Monografia (Trabalho de Conclusão de Curso de Graduação) — Universidade Federal de Santa Maria, Santa Maria, RS, Brasil, 2019.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 2, p. 69–74, mar 2008. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/1355734.1355746>.

MICHEL, O. et al. The programmable data plane: Abstractions, architectures, algorithms, and applications. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 4, may 2021. ISSN 0360-0300. Available from Internet: <https://doi.org/10.1145/3447868>.

MOORE, A. W.; PAPAGIANNAKI, K. Toward the accurate identification of network applications. In: **Proceedings of the 6th International Conference on Passive and Active Network Measurement**. Berlin, Heidelberg: Springer-Verlag, 2005. (PAM'05), p. 41–54. ISBN 3540255206. Available from Internet: <https://doi.org/10.1007/978-3-540-31966-5_4>.

NGUYEN, T. T.; ARMITAGE, G. A survey of techniques for internet traffic classification using machine learning. **IEEE Communications Surveys Tutorials**, v. 10, n. 4, p. 56–76, 2008.

NGUYEN, T. T.; ARMITAGE, G. A survey of techniques for internet traffic classification using machine learning. **IEEE Communications Surveys Tutorials**, v. 10, n. 4, p. 56–76, 2008.

PARIZOTTO, R. **In-network computing: overcoming constraints, failures, and configuration challenges**. Thesis (PhD thesis) — UFRGS, Porto Alegre, RS, June 2024.

PARIZOTTO, R. et al. Offloading machine learning to programmable data planes: A systematic survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 56, n. 1, aug 2023. ISSN 0360-0300. Available from Internet: <https://doi.org/10.1145/3605153>.

PATEL, V. R.; MEHTA, R. G. Performance analysis of mk-means clustering algorithm with normalization approach. In: **2011 World Congress on Information and Communication Technologies**. [S.l.: s.n.], 2011. p. 974–979.

SANDVINE, I. Global internet phenomena report 2023. **URL: https://www.sandvine.com/global-internet-phenomena-report-2023**, 2023.

SHAMSIMUKHAMETOV, D. et al. **YouTube, Netflix, Web dataset for Encrypted Traffic Classification**. IEEE Dataport, 2021. Available from Internet: <https://dx.doi.org/10.21227/s7x7-wd58>.

SONCHACK, J. et al. Turboflow: information rich flow record generation on commodity switches. In: . [S.l.: s.n.], 2018. p. 1–16.

USAMA, M. et al. Unsupervised machine learning for networking: Techniques, applications and research challenges. **IEEE Access**, v. 7, p. 65579–65615, 2019.

WU, Q. et al. P4sqa: A p4 switch-based qos assurance mechanism for sdn. **IEEE Transactions on Network and Service Management**, v. 20, n. 4, p. 4875–4886, 2023.

XAVIER, B. M. et al. Map4: A pragmatic framework for in-network machine learning traffic classification. **IEEE Transactions on Network and Service Management**, v. 19, n. 4, p. 4176–4188, 2022.

XIONG, Z.; ZILBERMAN, N. Do switches dream of machine learning? toward in-network classification. In: **Proceedings of the 18th ACM Workshop on Hot Topics in Networks**. New York, NY, USA: Association for Computing Machinery, 2019. (HotNets '19), p. 25–33. ISBN 9781450370202. Available from Internet: <https://doi.org/10.1145/3365609.3365864>.

ZHANG, J. et al. Network traffic classification using correlation information. **IEEE Transactions on Parallel and Distributed Systems**, v. 24, n. 1, p. 104–117, 2013.

ZHANG, Q.; SUN, S. A centroid k-nearest neighbor method. In: CAO, L.; FENG, Y.; ZHONG, J. (Ed.). **Advanced Data Mining and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 278–285. ISBN 978-3-642-17316-5.

ZHENG, C. et al. Automating in-network machine learning. **arXiv preprint arXiv:2205.08824**, 2022.

ZHU, X.; ZHANG, Y. Machine-learning-assisted traffic classification of user activities at programmable data plane. In: **2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)**. [S.l.: s.n.], 2022. p. 01–04.