UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MÉCANISMES DE GESTION DE TRAFIC INTELLIGENTS ET RÉSILIENTS POUR LES RÉSEAUX PROFONDÉMENT PROGRAMMABLES

THÈSE

PRÉSENTÉE

COMME EXIGENCE PARTIELLE

DU DOCTORAT EN INFORMATIQUE

PAR

MUHAMMAD SAQIB

JUIN 2025

REMERCIEMENTS

Je suis éternellement reconnaissant à Allah pour toutes les bénédictions, en particulier les personnes merveilleuses qu'Allah a placées dans ma vie, y compris celles mentionnées ci-dessous, qui m'ont soutenu tout au long de mon parcours de doctorat.

Je voudrais exprimer ma profonde gratitude au professeur Halima Elbiaze, mon directeur de thèse, pour sa patience, ses conseils inestimables et son soutien tout au long de mon parcours d'apprentissage. Je lui suis profondément reconnaissant pour ses commentaires constructifs, qui m'ont motivé à relever les défis, et pour ses qualités humaines exemplaires, qui ont fait de cette expérience une source d'inspiration. Elle m'a aidé non seulement à améliorer mes capacités de recherche, mais aussi à affiner mes compétences interpersonnelles.

Je remercie sincèrement ma famille, en particulier mes parents, pour leur amour inconditionnel, leur soutien et leurs innombrables sacrifices, qui m'ont même permis de poursuivre mes études alors qu'on avait besoin de moi à la maison. À mes frères et sœurs, merci pour vos encouragements constants, votre patience et votre positivité, qui m'ont toujours motivé à viser l'excellence.

Je suis particulièrement reconnaissant à ma femme, qui m'a rejoint vers la fin de mon doctorat et m'a soutenu avec patience et compréhension, en particulier lorsque j'étais à l'étranger pour mes recherches. Votre soutien a été inestimable pour m'aider à franchir cette étape.

Je remercie sincèrement le professeur Roch Glitho pour sa disponibilité et ses

précieux conseils. Je suis également profondément reconnaissant au professeur Yacine Ghamri-Doudane et au professeur Wessam Ajib pour leurs commentaires perspicaces et leurs conseils. Je remercie mes coauteurs pour leur coopération, leur disponibilité et leurs conseils.

Mouhamad Dieye, pour ses conseils et sa disponibilité à chaque fois que j'ai eu besoin de lui pour des discussions. Je tiens également à remercier Manel Gherari et Zakaria Ait Hmitti pour leur coopération et leur soutien au sein du laboratoire TRIME de l'UQAM. Ma sincère gratitude va également à tous mes collègues du laboratoire TRIME. Je remercie également tous les professeurs de l'UQAM qui ont généreusement partagé leur expertise avec moi.

Tout au long de ce parcours, j'ai eu l'honneur de bénéficier du soutien financier de la Fondation de l'UQAM et du Département d'informatique de l'UQAM pour l'orientation de mes recherches. Je vous remercie sincèrement.

À toute ma famille, à mes amis et à tous ceux qui, de près ou de loin, m'ont aidé à atteindre mes objectifs, j'exprime mes plus sincères remerciements.

DÉDICACES

À mon mentor, le Mufti Syed Adnan Kakakhail (DB), dont la motivation et les conseils m'ont incité à entreprendre ce voyage et à le poursuivre avec une vision et un objectif plus larges.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

A

AI Artificial Intelligence

B

BMv2 Behavioral Model version 2

C

CPU Central Processing Unit

D

DDoS Distributed Denial of Service

DTC Decision Tree Classifier

F

FN False Negative

FP False Positive

I

ILP Integer Linear Programming

InP Infrastructure Provider

IoT Internet of Things

ISP Internet Service Provider

L

LTE Long-Term Evolution

M

MAT Match-Action Table

MF Multi-Feature

ML Machine Learning

N

NGN Next-Generation Network

P

P4 Programming Protocol-Independent Packet Processors

PDP Programmable Data Plane

Q

QCI QoS Class Identifier

QoS Quality of Service

S

SDN Software-Defined Networking

SF Single Feature

SLA Service Level Agreement

SLO Service Level Objective

T

TN True Negative

TP True Positive

TPR True Positive Rate

RÉSUMÉ

L'augmentation des applications et des services ayant des exigences de performance distinct, telles que celles liées à la perte, à la bande passante et à la latence, a accru les demandes de trafic hétérogène que les réseaux de nouvelles génération (NGN) doivent prendre en charge de manière concomitante. Dans cet environnement en constante évolution, les applications critiques - de la surveillance des soins de santé en temps réel aux systèmes de contrôle industriel réactifs - imposent des exigencies strictes en termes de qualité de service (QoS), par consequent (ainis) nécessitant une utilisation efficace des ressources et des garanties de performance solides.

L'hétérogénéité du trafic réseau a accru la complexité de la gestion des ressources réseau d'une maniére efficase afin d' atteindre les niveaux de qualité de service souhaités. La classification du trafic réseau joue un rôle clé dans la gestion de ces ressources en affectant les flux de trafic à des groupes de qualité de service appropriés, ce qui permet un approvisionnement efficace en qualité de service tout en reduisant le trafic malveillant, ainsi maintenir la performance globale du réseau. Il est conseiller d'identifier les types de trafic dès les premières étapes de la création du flux pour une mise en œuvre rapide des politiques de qualité de service ou L'élimination du trafic malveillant pour se protéger contre les menaces potentielles. Toutefois, cela pose des problèmes, notamment parce que la diversité du trafic, y compris le trafic malveillant augmente, et pour différencier les types de trafic, les classificateurs basés sur l'apprentissage ont souvent besoin de données agrégées provenant de plusieurs paquets. En outre, l'évolution des schémas de trafic due à l'émergence de nouvelles applications ou aux caractéristiques changeantes du trafic d'attaque nécessite une adaptation continue du classificateur de trafic réseau. Cette adaptabilité permet de réagir rapidement à l'évolution des schémas de trafic, de réduire les erreurs de classification et de maintenir la qualité de service tout en faisant face à l'évolution des menaces qui pèsent sur le réseau. Pour atteindre ces objectifs dans les NGN, il faut adapter l'architecture du réseau, tirer parti de la programmabilité du réseau, du plan de contrôle au plan de données, et mettre en œuvre des stratégies de gestion adaptative du trafic pour gérer efficacement les demandes de trafic hétérogène et ameliorerla performances globales du réseau.

Cette thèse basée sur un article propose un cadre de gestion du trafic cadre de gestion du trafic pour la mise en œuvre de la qualité de service et la sécurité dans les réseaux de prochaine génération, en mettant l'accent sur la classification du trafic

à un stade précoce et adaptatif. Les contributions sont tripartite comme suite : (1) une solution efficace et précise de classification du trafic à un stade précoce pour une gestion efficace de la qualité de service ; (2) une approche adaptative de la classification du trafic pour reduire l'impact d'une mauvaise classification du trafic sur la qualité de service ; et (3) une approche fondée sur les données pour protéger le réseau contre les attaques volumétriques en évolution. Ensemble, ces contributions offrent un cadre pragmatique pour maintenir les performances et la sécurité des applications critiques dans un contexte d'évolution du trafic.

Le premier article de cette thèse, intitulé **An Intelligent and Programmable Data Plane for QoS-Aware Packet Processing**, propose un cadre de gestion du trafic pour gérer efficacement les demandes de trafic hétérogène dans le réseau. Il est difficile d'atteindre l'efficacité et la précision dans la classification du trafic à un stade précoce, car ces objectifs sont souvent contradictoires. En outre, le déploiement de règles spécifiques aux flux à l'échelle du réseau pour la mise en œuvre de la qualité de service peut entraîner une surchage et une utilisation importante de la mémoire. Pour relever ces défis, nous proposons un algorithme de classification du trafic léger, basé sur une seule fonctionnalité, ainsi qu'un mécanisme d'ordonnancement des paquets tenant compte de la qualité de service sans état. Le classificateur de trafic proposé intègre des objectifs de niveau de service (SLO) dans les en-têtes des paquets, ce qui permet un traitement des paquets sans état, toute en assurant la qualité de service et promouvoir une utilisation efficace des ressources du réseau.

Le deuxième article, intitulé **Adaptive In-Network Traffic Classifier : Bridging the Gap for Improved QoS by Minimizing Misclassification**, axer sur la maintenance de la précision de la classification dans un contexte d'évolution des modèles de trafic. Bien qu'initialement efficaces, les modèles de classification basés sur l'apprentissage à taille unique deviennent obsolètes au fur et à mesure que les schémas de trafic évoluent, ce qui conduit à une mauvaise classification et à une cartographie incorrecte de la qualité de service des flux de trafic. Ces erreurs de classification peuvent entraîner des violations de la qualité de service et des pénalités financières pour les fournisseurs d'infrastructure. Le cadre proposé quantifie les violations des accords de niveau de service (SLA) dues à une mauvaise classification, introduit un modèle économique pour évaluer son impact sur la rentabilité du fournisseur, et fournit un modèle d'apprentissage adaptatif pour améliorer continuellement la précision de la classification, en garantissant le maintien de la qualité de service.

Le dernier article, intitulé **A Data-Driven Approach to Mitigate Evolving Volumetric Attacks in Programmable Networks**, aborde le défi du maintien de la précision de la classification lors d'attaques volumétriques évolutives. Les

méthodes d'apprentissage uniques existantes ne parviennent pas à s'adapter à l'évolution des schémas d'attaque, ce qui entraîne un processus de mise à jour des modèles d'apprentissage dans les réseaux programmables qui prend du temps et demande beaucoup de travail. Pour surmonter ces limitations, nous proposant une approche automatisée, basée sur les données, afin d'identifier de nouveaux modèles de trafic malveillant et mettre à jour de manière transparente les modèles d'apprentissage, du plan de contrôle au plan de données, ce qui permet une défense continue contre les menaces.

Mots clés :

sification du trafic réseau, qualité de service, réseaux programmables, programmation du plan de données, réseaux de nouvelle génération, gestion du trafic hétérogène, mauvaise classification du trafic, objectifs de niveau de service, apprentissage automatique au sein du réseau, optimisation des ressources du réseau, attaques volumétriques, sécurité du réseau basée sur les données.

ABSTRACT

The rise of applications and services with **distinct** performance requirements, such as those related to loss, bandwidth, and latency, has increased the heterogeneous traffic demands that next-generation networks (NGNs) must concurrently support. In this evolving landscape, critical applications—from real-time healthcare monitoring to responsive industrial control systems—are driving increasingly **stringent** Quality of Service (QoS) requirements, necessitating **efficient** resource utilization and robust performance guarantees.

The heterogeneity and dynamicity of network traffic has increased the complexity of effectively managing network resources to achieve desired QoS levels. Network traffic classification plays a key role in managing network resources by assigning traffic flows to appropriate QoS groups, enabling efficient QoS provisioning while mitigating malicious traffic, thus maintaining overall network performance. Identifying traffic type at the earliest stage of flow creation is desirable for timely applying QoS policies or dropping malicious traffic to protect against potential threats. However, this presents challenges, particularly as the diversity of traffic, including malicious traffic, increases, and learning-based classifiers often require aggregated data from multiple packets to differentiate between traffic types. Furthermore, evolving traffic patterns due to the emergence of new applications or changing characteristics of attack traffic necessitate continuous adaptation of the network traffic classifier. This adaptability ensures a prompt response to changing traffic patterns, reduces misclassification, and maintains QoS while addressing evolving network threats. Achieving these objectives in NGNs requires tailoring network architecture, leveraging network programmability from the control plane to the data plane, and implementing adaptive traffic management strategies to efficiently handle the diverse traffic demands and obtain overall network performance.

This article-based thesis proposes deep programmable traffic management mechanisms for QoS provisioning and security in NGNs, with a focus on **early-stage and adaptive traffic classification**. The contributions are threefold: (1) an efficient yet accurate early-stage traffic classification solution for effective QoS management; (2) an adaptive traffic classification approach to mitigate the impact of traffic misclassification on QoS; and (3) a data-driven approach to safeguard the network against evolving volumetric attacks. Together, these contributions offer a pragmatic framework for maintaining the performance and security of critical applications amid evolving traffic patterns.

The first article in this thesis, titled **An Intelligent and Programmable Data Plane for QoS-Aware Packet Processing**, proposes a traffic management framework for efficiently managing heterogeneous traffic demands in the network. Achieving efficiency and accuracy in early-stage traffic classification is challenging, as these objectives often conflict. Additionally, deploying network-wide flow-specific rules for QoS provisioning can lead to significant memory usage and overhead. To address these challenges, the authors propose a lightweight, single-feature-based traffic classification algorithm and a stateless QoS-aware packet scheduling mechanism. The proposed traffic classifier embeds service level objectives (SLOs) into packet headers, leading to stateless, QoS-aware packet processing and promoting efficient use of network resources.

The second article, titled **Adaptive In-Network Traffic Classifier: Bridging the Gap for Improved QoS by Minimizing Misclassification**, focuses on maintaining classification accuracy amid evolving traffic patterns. While initially effective, one-size-fits-all learning-based classification models become outdated as traffic patterns evolve, leading to misclassification and incorrect QoS mapping of traffic flows. Such misclassification can result in service quality violations and financial penalties for Infrastructure Providers (InPs). The proposed framework quantifies service-level agreement (SLA) violations due to misclassification, introduces an economic model to evaluate its impact on provider profitability, and provides an adaptive learning model to continually improve classification accuracy, ensuring QoS is maintained.

The last article, titled **A Data-Driven Approach to Mitigate Evolving Volumetric Attacks in Programmable Networks**, addresses the challenge of maintaining classification accuracy during evolving volumetric attacks. Existing one-size-fits-all learning-based methods fail to adapt to changing attack patterns, leading to a time-consuming and labor-intensive process to update learning models in programmable networks. To overcome these limitations, the authors propose an automated, data-driven approach for identifying novel malicious traffic patterns and seamlessly updating learning models, from the control plane to the data plane, enabling continuous threat defense.

Keywords:

Network traffic classification, quality of service, programmable networks, data plane programming, next-generation networks, heterogeneous traffic management, traffic misclassification, service level objectives, in-network machine learning, network resource optimization, volumetric attacks, data-driven network security.

CHAPTER I

INTRODUCTION

1.1 Contexte et motivation

1.1.1 Caractéristiques du trafic sur le réseau

La dépendance croissante à l'égard d'Internet, stimulée par les progrès des technologies de connectivité telles que la fibre et la 5G, a entraîné une croissance sans précédent du trafic réseau. Le nombre d'utilisateurs d'Internet a dépassé les 5 milliards en 2023 (International Telecommunication Union, 2023), reflétant une augmentation substantielle qui continue à défier la capacité des Internet Service Providers (ISPs) à gérer ce volume efficacement. Cette augmentation, associée à l'accessibilité généralisée de divers appareils et à des options de connectivité rentables, a considérablement intensifié la charge de trafic sur le réseau. Par conséquent, les fournisseurs de services Internet sont de plus en plus contraints de gérer efficacement le trafic hétérogène et de répondre aux exigences en matière de qualité de service (QoS).

En outre, l'essor des applications critiques, telles que la surveillance des soins de santé en temps réel et les systèmes de contrôle industriel réactifs, a conduit à des exigences de qualité de service de plus en plus rigoureuses (De Alwis et al., 2021). Ces applications exigent des garanties de performance précises, notamment un délai

minimal de bout en bout, une faible gigue et une perte de paquets proche de zéro, afin de garantir un fonctionnement fiable et sûr. Par conséquent, les utilisateurs finaux génèrent un trafic très hétérogène et variable, caractérisé par des fluctuations de charge imprévisibles avec des débits de données allant du mégabit au térabit par seconde et des exigences de latence allant de la seconde à la nanoseconde. Cette combinaison de **caractéristiques de trafic diverses et dynamiques** pose des défis considérables aux mécanismes conventionnels de gestion du trafic.

Cas d'usage / Applications principales	**Débit**	**Latence de bout en bout**	**Taille du réseau / Disponibilité**	**Niveau de sécurité**
Internet de tout	100-1000 Mbps	ms à s	$> 20 \times 10^6$ dispositifs	MOYEN
Réseau électrique intelligent 2.0	10-2000 Mbps	1-10 ms	> 2 sur 10-100 km	MOYEN
Téléprésence holographique	> 2-4 Tbps	0.1 ms	> 1	MOYEN
Mobilité basée sur UAV	10-100 Mbps	1-10 ms	> 10	MOYEN
Réalité étendue	> 2-4 Tbps	0.1 ms	> 1	MOYEN
Véhicules connectés et autonomes	1-10 Gbps	> 10	> 1	MOYEN
Industrie 5.0	100-1000 Mbps	1-10 ms	10^6 dispositifs par km^2	MOYEN
IoT hyper-intelligent	> 24 Gbps	10-100 μs	$> 125 \times 10^6$ dispositifs	MOYEN
Cobots	> 24 Gbps	2.8 ms	> 1	MOYEN
BANs personnalisés	1-50 Mbps	ms à s	$> 10^3$ nœuds par cm^3	TRÈS ÉLEVÉ
Santé intelligente	> 24 Gbps	< 1 ms	> 99,999 %	ÉLEVÉ

Table 1.1 Exigences en matière de qualité de service pour les services et les applications prévus pour les NGN (De Alwis et al., 2021)

Dans les réseaux de prochaine génération (NGN), c'est-à-dire la 5G et au-delà, le découpage du réseau joue un rôle essentiel dans la gestion du trafic hétérogène, l'utilisation efficace des ressources et la satisfaction des diverses exigences de performance (Zhang, 2019). Le trafic est généralement mis en correspondance avec des groupes de qualité de service à l'aide d'une table d'identification de classe de qualité de service (QCI), qui fournit un ensemble prédéfini de valeurs liées à des caractéristiques de qualité de service spécifiques (ShareTechnote, 2024). Bien que la table QCI permette de classer divers types de trafic, sa nature statique

limite la souplesse nécessaire pour répondre aux exigences de qualité de service des applications émergentes, notamment en ce qui concerne les débits de données, la latence et la priorité (résumés dans le tableau 1.1). Dans le même temps, les attaquants peuvent imiter des modèles de trafic légitimes pour injecter des cyberattaques dans le réseau. Même une attaque volumétrique à petite échelle peut dégrader les performances et avoir un impact important sur les flux de trafic critiques.

Le classificateur de trafic réseau basé sur l'apprentissage est un **composant clé du cadre de gestion du trafic**, servant à la fois la gestion de la qualité de service et la sécurité du réseau (Salman et al., 2020). Du point de vue de la qualité de service, le classificateur identifie et classe avec précision les différents flux de trafic, en veillant à ce qu'ils soient affectés aux groupes de qualité de service appropriés. Cela permet au réseau d'allouer les ressources de manière efficace, en veillant à ce que les applications sensibles à la latence bénéficient de chemins à faible latence, tandis que les applications gourmandes en bande passante disposent d'une capacité suffisante. En outre, en filtrant le trafic malveillant et en donnant la priorité aux flux légitimes, le classificateur contribue à maintenir les performances et la fiabilité globales du réseau, garantissant ainsi que les services essentiels restent ininterrompus. Cependant, dans le paysage évolutif du trafic Internet, les méthodes de classification du trafic existantes sont souvent inefficaces ou insuffisamment adaptatives pour prendre en charge l'approvisionnement en qualité de service et la sécurité dans les NGN. Par conséquent, la classification du trafic basée sur l'apprentissage adaptatif, qui apprend de manière dynamique les modèles de trafic en évolution - induits par l'émergence de nouvelles applications ou de techniques d'évasion utilisées par les attaquants - est cruciale pour la gestion du trafic dynamique et hétérogène tout en maintenant les performances du réseau.

1.1.2 Exigences

Il est souhaitable de disposer d'un classificateur de trafic réseau doté des propriétés suivantes pour gérer efficacement les demandes de trafic dynamique et hétérogène dans les NGN :

- **Classification du trafic aux premiers stades** La classification du trafic aux premiers stades de la transmission des données est cruciale pour la fourniture de la qualité de service et la sécurité du réseau. En identifiant la nature du trafic dès son entrée dans le réseau, les opérateurs peuvent prendre instantanément des décisions cruciales concernant la hiérarchisation et le routage, en veillant à ce que les applications sensibles aux temps de latence bénéficient de la largeur de bande nécessaire et de chemins à faible temps de latence. Cette approche est particulièrement importante dans les NGN, où la diversité et l'ampleur des appareils connectés exigent une allocation efficace des ressources pour maintenir les performances. En outre, la classification à un stade précoce renforce la sécurité en permettant l'identification immédiate des schémas de trafic suspects ou anormaux, ce qui est essentiel pour atténuer les menaces émergentes telles que les attaques par déni de service distribué (DDoS). À mesure que la complexité et le volume du trafic réseau augmentent, la classification à un stade précoce jouera un rôle essentiel dans le maintien des performances et de la sécurité des réseaux de prochaine génération.

- **L'apprentissage adaptatif:** Un modèle unique basé sur l'apprentissage devient obsolète au fur et à mesure que les modèles de trafic évoluent. Cette évolution conduit inévitablement à des erreurs de classification, ce qui entraîne une mauvaise affectation des flux de trafic aux groupes de qualité de service. Le problème est encore plus critique du point de vue de la sécurité, car les attaquants peuvent exploiter la rigidité d'un modèle de classification

appris en imitant des schémas de trafic légitimes, ce qui rend le classificateur inefficace et permet au trafic malveillant de contourner la détection. Il est donc impératif d'adapter la nature évolutive des modèles de trafic pour maintenir les performances du réseau.

- **L'approvisionnement en qualité de service sans état** Avec l'augmentation des services ayant des exigences strictes en matière de qualité de service, l'éventail des demandes d'applications que les réseaux doivent prendre en charge s'élargit. Les réseaux doivent gérer des milliers de flux simultanés, chacun ayant des besoins distincts en matière de qualité de service. La mise en œuvre de règles spécifiques à chaque flux à chaque nœud du réseau impose une mémoire substantielle et une surcharge de calcul, ce qui la rend irréalisable. Pour relever ce défi, le classificateur doit intégrer les objectifs de service dans l'en-tête des paquets à la périphérie du réseau, ce qui permet aux nœuds du réseau de traiter les paquets en fonction de leurs exigences de service d'une manière apatride. Cette approche pourrait réduire les frais généraux et garantir que les divers besoins en matière de qualité de service sont satisfaits de manière efficace.

1.2 Technologies habilitantes

L'un des principaux catalyseurs des NGN est la programmabilité des réseaux, qui a considérablement évolué au fil des ans (Alberti et al., 2024). Au départ, le Software-Defined Networking (SDN) a joué un rôle central en séparant le plan de contrôle du plan de données (Xia et al., 2014). Ce découplage a permis un contrôle centralisé du réseau par le biais de contrôleurs programmables, utilisant généralement des protocoles comme OpenFlow pour gérer et programmer le comportement du réseau. Les techniques d'apprentissage automatique (ML) et d'intelligence artificielle (IA) sont devenues des outils essentiels pour gérer intelligemment les demandes de trafic

de plus en plus hétérogènes et dynamiques dans les NGN (El Rajab et al., 2024; Schwarzmann et al., 2024), permettant aux opérateurs de configurer et de gérer les dispositifs de réseau de manière plus flexible et automatique, améliorant ainsi l'efficacité globale des opérations de réseau.

L'évolution de la programmation du plan de contrôle vers le plan de données représente un changement vers un contrôle plus granulaire du traitement des paquets (Hauser et al., 2023). Si la programmabilité du plan de contrôle permet un contrôle centralisé du réseau, sa dépendance à l'égard du matériel à fonction fixe pour le traitement et l'acheminement des paquets limite sa flexibilité en ce qui concerne l'adaptation du réseau à certains objectifs de performance. La programmabilité du plan de données est apparue pour remédier à cette limitation, en permettant aux opérateurs de réseaux de personnaliser directement les fonctions de traitement et d'acheminement des nœuds de réseau. Cette évolution permet de mettre en œuvre des fonctions de réseau qui sont non seulement gérées de manière centralisée, mais aussi finement adaptées à des exigences spécifiques, directement au niveau du traitement des paquets.

La programmation du plan de données permet aux développeurs de définir précisément comment les paquets sont identifiés, traités et acheminés à travers le réseau. Des langages de programmation tels que P4 (Programming Protocol-independent Packet Processors) permettent d'élaborer des instructions personnalisées pour les dispositifs de réseau, en spécifiant la manière dont les paquets doivent être traités. Contrairement aux plans de données traditionnels à fonction fixe, les plans de données programmables (PDP) permettent des ajustements dynamiques en temps réel, ce qui est essentiel pour s'adapter aux modèles de trafic en constante évolution des réseaux modernes.

Un aspect important de la programmation des plans de données est sa capacité à

permettre la classification du trafic dans le réseau (Xiong and Zilberman, 2019; Xavier et al., 2021). En employant un classificateur au niveau du plan de données, le trafic peut être identifié et traité au débit de ligne, ce qui signifie que la classification des paquets a lieu dès que les paquets entrent dans le réseau. Cette capacité est particulièrement utile pour gérer les applications sensibles à la latence et garantir que les services critiques reçoivent les ressources nécessaires sans introduire de retard. La flexibilité offerte par la programmation du plan de données permet aux opérateurs de définir des critères personnalisés de correspondance des paquets, ce qui permet une prise de décision rapide pour la hiérarchisation et le routage du trafic.

Par exemple, avec la programmation du plan de données, un dispositif de réseau peut être chargé d'inspecter les en-têtes des paquets, d'identifier des flux de trafic spécifiques, d'éliminer le trafic malveillant tout en assignant les flux légitimes aux classes de qualité de service appropriées. Cette classification précoce au niveau du plan de données garantit une allocation optimale des ressources du réseau dès le départ, réduisant la probabilité de congestion et minimisant la dégradation du service. Il en résulte également une infrastructure de réseau plus efficace et plus réactive, capable de répondre aux besoins d'applications diverses à un débit proche de celui de la ligne.

En déplaçant la programmabilité du plan de contrôle vers le plan de données, les NGN acquièrent la capacité de mettre en œuvre des stratégies sophistiquées de gestion du trafic directement là où les paquets sont traités. Cette évolution a permis de répondre aux exigences strictes des applications modernes, en jetant les bases d'une gestion efficace de la qualité de service et de la sécurité du réseau.

1.3 Enoncé du problème

Les mécanismes actuels de gestion du trafic sont confrontés à trois limitations critiques : une classification inefficace du trafic à un stade précoce, une adaptabilité limitée à l'évolution des schémas de trafic et des goulets d'étranglement en matière d'évolutivité dans la fourniture de la qualité de service. Plus précisément, les classificateurs de trafic existants au sein du réseau ne peuvent pas identifier le trafic de manière efficace et précise, car il s'agit souvent d'objectifs concurrents (Xavier et al., 2021). Cette limitation réduit leur efficacité à fournir la qualité de service souhaitée pour les applications sensibles à la latence et à atténuer rapidement les menaces émergentes, telles que les attaques volumétriques. En outre, les modèles statiques de ML dans le réseau ne parviennent pas à s'adapter à l'évolution des modèles de trafic (Zheng et al., 2023), ce qui entraîne des erreurs de classification, des violations des accords de niveau de service (SLA) et des vulnérabilités accrues du réseau. Enfin, les mécanismes de provisionnement de la qualité de service avec état introduisent une surcharge importante, ce qui limite leur évolutivité dans les grands réseaux (Karakus and Durresi, 2017), tandis que les approches sans état restent sous-explorées en dépit de leur potentiel. Pour relever ces défis, il faut une nouvelle solution qui unifie la classification adaptative dans le réseau à un stade précoce et le provisionnement de la qualité de service sans état pour soutenir à la fois la performance et la sécurité dans les NGN.

1.4 Objectifs de la recherche

L'objectif général de cette thèse est de développer des mécanismes de gestion du trafic intelligents et résilients qui améliorent la fourniture de la qualité de service et renforcent la sécurité du réseau dans les NGN. À cette fin, les objectifs de recherche spécifiques sont les suivants :

un cadre complet qui intègre la classification du trafic avec un approvisionnement en qualité de service sans état qui est basé sur des objectifs de service afin de gérer intelligemment les demandes de trafic hétérogènes. l'impact d'une mauvaise classification du trafic sur les performances du réseau et la rentabilité des fournisseurs de services, en particulier dans des conditions dynamiques et de trafic élevé. les limites des classificateurs existants dans le réseau en développant des méthodes d'apprentissage adaptatif qui vont au-delà des modèles uniques et répondent efficacement à l'évolution des modèles de trafic.

1.5 Organisation de la thèse

Cette thèse se compose de six chapitres, dont le premier traite des caractéristiques du trafic dans les NGN. Ce chapitre établit le contexte de la thèse et présente les idées clés qui permettent de comprendre le travail décrit dans cette thèse, tout en explorant les défis fondamentaux liés à la gestion du trafic hétérogène et dynamique dans les réseaux futuristes.

L'introduction générale fournit une vue d'ensemble de l'étude, reliant le contexte, le problème, les objectifs et les questions de recherche pertinentes aux trois articles qui constituent le cœur de la thèse. Elle vise à définir le contexte général de la recherche avant d'aborder les contributions spécifiques, en mettant l'accent sur les questions relatives à la gestion du trafic hétérogène, aux exigences strictes en matière de qualité de service et à l'évolution des menaces pour la sécurité. La littérature relative au sujet de recherche indique les facteurs problématiques suivants :

- Il n'existe actuellement aucun cadre global capable de gérer intelligemment les demandes de trafic hétérogène en intégrant la classification du trafic avec

un mécanisme de fourniture de qualité de service tenant compte des objectifs de service;

- Les conséquences d'une mauvaise classification du trafic sur les performances du réseau et la rentabilité des fournisseurs de services n'ont pas été étudiées;
- Les classificateurs de trafic en réseau actuels, basés sur l'apprentissage, reposent souvent sur des méthodes uniques qui ne parviennent pas à s'adapter à l'évolution des schémas de trafic.

Ce chapitre jette les bases permettant de comprendre les motivations qui sous-tendent les solutions proposées et fournit un lien cohérent avec les articles de recherche ultérieurs qui abordent ces défis en détail.

Le deuxième chapitre présente les étapes méthodologiques utilisées dans cette étude et justifie nos choix méthodologiques. Il offre également une vue d'ensemble des contributions de cette thèse avant de résumer les concepts clés liés aux méthodes sélectionnées. Afin d'améliorer la lisibilité et de maintenir la pertinence, une revue de la littérature spécifique est effectuée pour chaque article examiné dans cette thèse.

Le troisième chapitre présente le premier article de cette thèse, qui se concentre sur la classification du trafic à un stade précoce et la fourniture de la qualité de service pour la gestion du trafic hétérogène. Ce chapitre jette les bases permettant de comprendre les défis auxquels les NGN seront confrontés avec l'émergence d'applications caractérisées par des exigences strictes en matière de qualité de service.

Le quatrième chapitre présente le deuxième article, qui étudie les effets d'une mauvaise classification du trafic sur les violations des accords de niveau de service dans les réseaux multi-classes et multi-trajets. L'approche proposée utilise un

classificateur de trafic adaptatif pour minimiser l'impact négatif de la mauvaise classification du trafic sur la qualité de service.

Le cinquième chapitre couvre le troisième article, qui aborde les limites des classificateurs de trafic non adaptatifs existants et s'appuie sur un cas d'utilisation de la cybersécurité pour protéger le réseau contre les attaques volumétriques en évolution.

Enfin, le sixième chapitre conclut la thèse par une discussion des principaux résultats et des pistes de recherche.

CHAPTER II

MÉTHODOLOGIE

Ce chapitre décrit l'approche méthodologique utilisée dans cette étude. Il se compose de trois parties :

- La première partie traite de la motivation qui sous-tend les choix méthodologiques de cette thèse;
- La deuxième partie décrit les différentes contributions de cette thèse;
- La troisième partie passe en revue les mesures prises pour maintenir une qualité satisfaisante de cette étude.

2.1 Methodological approach

2.1.1 L'ampleur et la nature du problème

La gestion du trafic dans les NGN présente des défis importants en raison de sa nature dynamique, qui est influencée par l'émergence de nouvelles applications avec des exigences de qualité de service distinctes et les techniques d'évasion employées par les attaquants. Du point de vue des utilisateurs finaux, chaque application ou service nécessite des garanties de qualité de service spécifiques, ce qui complique l'allocation des ressources à mesure que de nouveaux types de trafic apparaissent.

Les mécanismes traditionnels de qualité de service, tels que IntServ (Shenker et al., 1997), DiffServ (Black and Jones, 2015) et DetNet (Finn et al., 2019), reposent sur une logique de fonction fixe et dépendent de règles spécifiques aux flux à l'échelle du réseau. Ces mécanismes deviennent inefficaces et non viables à mesure que le volume des flux de services augmente. La situation est encore compliquée par l'évolution des modèles de trafic, qu'il s'agisse de nouvelles applications ou de changements dans les caractéristiques du trafic d'attaque, ce qui rend les modèles de classification et de qualité de service statiques et uniformes inefficaces pour s'adapter. Il en résulte souvent des classifications inexactes et des performances de réseau sous-optimales. Par conséquent, la gestion du trafic dans les NGN nécessite des mécanismes de gestion du trafic intelligents, adaptatifs et réactifs pour garantir à la fois les performances et la sécurité dans un contexte de forte variabilité du trafic.

2.1.2 Etapes méthodologiques

La méthodologie de cette thèse se situe à l'intersection de l'analyse du trafic réseau, de la ML et de la programmabilité du réseau, englobant à la fois les plans de contrôle et de données. Nous effectuons une analyse du trafic à partir d'ensembles de données accessibles au public afin de discerner les modèles et les tendances du trafic. Ensuite, des algorithmes de ML sont utilisés pour former des modèles de classification, qui sont ensuite déployés dans des réseaux programmables pour classer le trafic au fur et à mesure qu'il traverse le réseau.

La programmabilité du réseau facilite le déploiement des modèles de ML au sein du réseau. Le plan de contrôle fournit d'importantes ressources de stockage et de calcul pour l'apprentissage et le déploiement des modèles. Cependant, sa nature centralisée introduit des surcharges de latence et de débit dues aux fréquentes

requêtes des dispositifs du plan de données (Chen et al., 2022). En revanche, le plan de données excelle dans l'inférence du débit proche de la ligne, ce qui permet une prise de décision rapide. Toutefois, il est limité par les ressources opérationnelles et informatiques (Sharma et al., 2017). Par conséquent, dans cette étude, le plan de contrôle est utilisé pour la formation des modèles ML, tandis que les règles de modèle résultantes sont compilées dans des formats simplifiés et mappées sur le plan de données pour une inférence rapide et en temps réel.

La sélection des caractéristiques d'entrée et des algorithmes de ML est fortement influencée par les contraintes de déploiement dans les PDP. Par exemple, les langages de programmation du plan de données tels que P4 ne prennent pas en charge les opérations telles que l'arithmétique à virgule flottante et les boucles (Bosshart et al., 2014). En outre, les dispositifs cibles P4 possèdent une capacité de mémoire limitée [1][2]. Par conséquent, l'utilisation d'un nombre minimal de fonctionnalités est hautement souhaitable pour respecter les contraintes du plan de données. Pour répondre à ces contraintes, il est essentiel de concevoir des classificateurs de trafic qui utilisent des fonctions minimales et légères et qui prennent des décisions rapides tout en respectant les limites du langage P4 ou de sa cible. Tout au long de cette recherche, nous mettons l'accent sur l'utilisation d'un ensemble minimal et simple de caractéristiques qui peuvent être directement extraites des en-têtes de paquets ou des statistiques de flux de base, garantissant ainsi une faible surcharge sur les dispositifs de réseau programmables. Compte tenu des capacités actuelles de P4, le classificateur à arbre de décision (DTC) est choisi comme l'algorithme le plus approprié (Xiong and Zilberman, 2019). Sa logique de décision, basée sur des comparaisons séquentielles, est intrinsèquement compatible avec les tables de

[1]http://bmv2.org/

[2]https://www.barefootnetworks.com/products/brief-tofino/

correspondance et d'action (MAT) employées dans le plan de données P4.

La programmation des plans de contrôle et de données implique un compromis : le plan de contrôle offre une capacité de calcul substantielle mais introduit une latence et des frais généraux, tandis que le plan de données facilite un traitement proche du débit de ligne mais est limité en termes d'opérations et de ressources. Nos recherches tirent parti des atouts des deux plans pour gérer la variabilité du trafic dans les réseaux de prochaine génération. Le plan de contrôle est responsable de l'entraînement des modèles ML à l'aide de données historiques et du mappage ultérieur des règles du modèle au plan de données pour une inférence en temps quasi réel. Le plan de données classifie les flux au débit de la ligne et incorpore les objectifs de niveau de service (SLO) dans les paquets pour diriger leur traitement dans le réseau. Lorsqu'il détecte la variabilité du trafic ou de nouveaux schémas, le plan de données enregistre ces instances en vue d'une analyse hors ligne et d'un réapprentissage. Le plan de contrôle réapprend alors le modèle ML sur les instances enregistrées et met à jour les règles du modèle dans le plan de données, garantissant ainsi que la classification s'adapte à l'évolution des schémas de trafic avec un minimum de perturbations. Cette intégration transparente entre les plans de contrôle et de données est essentielle pour maintenir la qualité de service et se défendre contre les comportements malveillants dans les réseaux de prochaine génération.

2.2 Contribution de la thèse

Cette thèse vise à fournir des contributions au problème de la gestion du trafic dans les NGN qui seraient à la fois académiques et industrielles. Ainsi, chaque article de cette thèse se concentre sur un aspect spécifique du problème et propose des solutions simples et efficaces. L'analyse et la formulation du problème sont basées

sur des approches théoriques bien établies, tandis que les expériences reposent sur des outils testés dans la pratique.

Le premier article de cette thèse, intitulé **An Intelligent and Programmable Data Plane for QoS-Aware Packet Processing**, a été publié dans la revue *IEEE Transactions on Machine Learning in Communication and Networking*. L'auteur de cette thèse ont joué un rôle actif dans la conception et le développement d'un algorithme léger de classification du trafic et d'un mécanisme de fourniture de QoS sans état, ainsi que dans la mise en œuvre de l'évaluation expérimentale, l'acquisition des résultats et l'exécution de diverses tâches d'édition et de correction basées sur les commentaires des évaluateurs de la revue.

Ce document présente les éléments suivants :

- Etude de cas soulignant la nécessité d'un cadre intégré pour la classification du trafic à un stade précoce et le provisionnement efficace de la qualité de service. Elle décrit les conditions préalables à une solution potentielle et souligne les différences de notre travail par rapport à la recherche existante dans ce domaine ;
- Cadre pragmatique pour permettre une classification précoce des flux de trafic et un mécanisme de traitement des paquets tenant compte de la qualité de service, en tenant compte des demandes de trafic hétérogènes de plusieurs applications émergentes ;
- Une évaluation expérimentale comparative de notre approche par rapport aux travaux existants dans le domaine.

Le second article, intitulé **Adaptive In-Network Traffic Classifier : Bridging the Gap for Improved QoS by Minimizing Misclassification**, a été publié

dans la revue *IEEE Open Journal of the Communication Society*. L'auteur de cette thèse ont contribué activement à la formulation de la programmation linéaire en nombres entiers (ILP) et au développement de l'algorithme, à la conception du système, au développement et à la mise en œuvre de l'évaluation expérimentale, à l'acquisition des résultats et à la gestion de diverses éditions et révisions en réponse aux commentaires des évaluateurs.

Cet article présente les éléments suivants :

- Formulation du problème d'allocation des demandes de trafic entre plusieurs classes et chemins dans le réseau en tant qu'ILP ;
- Étude et quantification de l'impact d'une mauvaise classification du trafic sur les violations des accords de niveau de service et sur la rentabilité de l'InP dans des environnements de réseaux multi-classes et multi-trajets ;
- Un classificateur de trafic adaptatif basé sur le ML qui s'adapte continuellement à l'évolution des modèles de trafic, maintenant la qualité de service dans le réseau en améliorant la précision de la classification du trafic.

Le dernier article, intitulé **A Data-Driven Approach to Mitigate Evolving Volumetric Attacks in Programmable Networks**, est soumis à la revue *IEEE Transactions on Machine Learning in Communication and Networking*. Une première version de cet article intitulé **In-Network Defense : Safeguarding the Network Against Evolving DDoS Attacks** est acceptée pour présentation à la *IEEE Global Communication Conference (GLOBECOM) 2024*. L'auteur de cette thèse ont contribué activement à la formulation du problème et au développement de l'algorithme, à la conception du système, au développement et à la mise en œuvre de l'évaluation expérimentale, à l'acquisition des résultats et à la gestion

de diverses éditions et révisions en réponse aux commentaires des évaluateurs. Le document présente les éléments suivants :

- Une étude de cas soulignant le besoin de classificateurs de trafic adaptatifs pour faire face aux attaques volumétriques évolutives dans le réseau, en mettant l'accent sur l'Internet des objets (IoT) et le cas d'utilisation métaverse;
- Un nouveau cadre pour automatiser la détection des dérives et les processus d'adaptation transparents dans les réseaux programmables, sans perturber le trafic normal du réseau;
- Des résultats expérimentaux qui valident l'impact de la dérive sur la dégradation des performances du modèle ML et démontrent l'efficacité de la méthode proposée dans l'atténuation des attaques volumétriques évolutives et la protection du réseau contre les menaces émergentes.

2.3 Critères de qualité de l'étude

Nous avons adopté la méthodologie de recherche suivante pour garantir la qualité de l'étude. Pour chaque article, nous avons procédé à une analyse documentaire approfondie afin d'identifier les facteurs pertinents pour le sujet. Chaque article comprend donc une section consacrée à l'analyse des travaux connexes. Cette recherche bibliographique nous a permis d'acquérir une compréhension globale du sujet et d'analyser les limites des solutions existantes. Ensuite, nous avons présenté la formulation mathématique nécessaire et la conception du système, en tenant compte des exigences spécifiques de la conception. Le déroulement de la conception proposée est illustré de manière algorithmique. La dernière étape concerne la phase d'évaluation, au cours de laquelle nous présentons les résultats expérimentaux.

Nous validons ces résultats en les comparant à des méthodes similaires issues de la littérature.

2.3.1 Expériences

Nous avons mis en œuvre nos solutions en utilisant Python pour la programmation du plan de contrôle et P4 pour la programmation du plan de données, en ciblant le modèle comportemental version 2 (BMv2) pour la compilation (P4 Language Consortium, 2024). BMv2 est choisi pour sa reproductibilité et sa facilité de débogage. Mininet (Mininet Project, 2024) est utilisé pour créer et gérer la topologie du réseau, fournissant un environnement réaliste pour tester notre mise en œuvre et émuler différents scénarios de réseau. Les bibliothèques Python telles que scikit-learn, numpy et pandas sont utilisées pour le traitement des données et la mise en œuvre des algorithmes de ML. Pour obtenir des résultats cohérents, nous avons répété les simulations plusieurs fois et éliminé les résultats aberrants si nécessaire.

Pour minimiser les erreurs de mise en œuvre des solutions proposées, nous utilisons souvent la procédure suivante :

- Lorsqu'une mise en œuvre bien documentée d'une approche de conception similaire est disponible, nous avons réutilisé et adapté le code selon les besoins, en utilisant des sources telles que les dépôts GitHub ;
- Les paramètres de simulation sont initialement inspirés de la littérature pertinente ou des simulations précédentes et sont ensuite modifiés de manière expérimentale en fonction des scénarios de simulation spécifiques.

CHAPTER III

ARTICLE 1 - AN INTELLIGENT AND PROGRAMMABLE DATA PLANE FOR QOS-AWARE PACKET PROCESSING

Muhammad Saqib
Université du Québec à Montréal
Montréal (Québec), Canada

Zakaria Ait Hmitti
Université du Québec à Montréal
Montréal (Québec), Canada

Halima Elbiaze, Professeur titulaire
Université du Québec à Montréal
Montréal (Québec), Canada

Roch Glitho, Professeur titulaire
Concordia University
Montréal (Québec), Canada

Yacine Ghamri-Doudane, Professeur associé

La Rochelle University

La Rochelle, France

FOREWORD TO ARTICLE 1

The first paper of this thesis focuses on early-stage network traffic classification for effective QoS management. It aims to design an integrated traffic management framework capable of intelligently handling heterogeneous traffic demands in networks. This paper was published in the scientific journal **IEEE Transactions on Machine Learning in Communication and Networking (IEEE TMLCN)**, Volume 2, Oct 2024.

- M. Saqib, H. Elbiaze, R. H. Glitho and Y. Ghamri-Doudane, **An Intelligent and Programmable Data Plane for QoS-Aware Packet Processing**, in IEEE Transactions on Machine Learning in Communications and Networking, vol. 2, pp. 1540-1557, 2024, doi: 10.1109/TMLCN.2024.3475968.

A related article to the above article was published in the scientific journal (conference) **IEEE Global Communications Conference (GLOBECOM)**, Dec 2022.

- M. Saqib, Z. A. Hmitti, H. Elbiaze and R. H. Glitho, **An Accurate & Efficient Approach for Traffic Classification Inside Programmable Data Plane**, GLOBECOM 2022 - 2022 IEEE Global Communications Conference, Rio de Janeiro, Brazil, 2022, pp. 6331-6336, doi: 10.1109/GLOBECOM48099.2022.10000863.

Papers published in IEEE TMLCN and IEEE GLOBECOM are peer-reviewed in accordance with the requirements set forth in the IEEE PSPB Operations Manual (Sections 8.2.1.C & 8.2.2.A). Each published paper has been reviewed by a minimum of two independent reviewers using a single-blind peer review process,

where the identity of the reviewers is not known to the authors, but the reviewers know the identity of the authors. Papers are subject to plagiarism check before acceptance.

ABSTRACT

One of the main features of data plane programmability is that it allows the easy deployment of a programmable network traffic management framework. One can build an early-stage Internet traffic classifier to facilitate effective Quality of Service (QoS) provisioning. However, maintaining accuracy and efficiency (i.e., processing delay/pipeline latency) in early-stage traffic classification is challenging due to memory and operational constraints in the network data plane. Additionally, deploying network-wide flow-specific rules for QoS leads to significant memory usage and overheads. To address these challenges, we propose new architectural components encompassing efficient processing logic into the programmable traffic management framework. In particular, we propose a single feature-based traffic classification algorithm and a stateless QoS-aware packet scheduling mechanism. Our approach first focuses on maintaining accuracy and processing efficiency in early-stage traffic classification by leveraging a single input feature - sequential packet size information. We then use the classifier to embed the Service Level Objective (SLO) into the header of the packets. Carrying SLOs inside the packet allows QoS-aware packet processing through admission control-enabled priority queuing. The results show that most flows are properly classified with the first four packets. Furthermore, using the SLO-enabled admission control mechanism on top of the priority queues enables stateless QoS provisioning. Our approach outperforms the classical and objective-based priority queuing in managing heterogeneous traffic demands by improving network resource utilization.

Keywords: Machine learning, Quality of Service, data plane programming, in-network traffic classification, network traffic management

3.1 Introduction

In recent years, the rise of services with stringent quality requirements—specifically loss, bandwidth, and latency—has expanded the spectrum of application demands that networks must concurrently support. This trend is particularly pronounced in the IoT and metaverse, where the growth of latency-sensitive and bandwidth-intensive services and applications continues unabated. Within this evolving landscape, the Quality of Service (QoS) might be underscored by the escalating requirements of latency-critical Internet of Things (IoT) applications (Schulz et al., 2017). These applications, ranging from real-time monitoring in healthcare to responsive control systems in industrial IoT, demand efficient resource utilization and stringent latency guarantees. In this context, providing and guaranteeing low-latency services are paramount.

Several provisioning mechanisms have emerged to manage and utilize network resources efficiently, offering differentiated services at the scale (Wu et al., 2022). However, before applying QoS policies, a crucial preliminary step involves precisely mapping traffic flows to the relevant QoS groups. Consequently, identifying various traffic types and the efficient provision of QoS become integral components of any robust traffic management framework. In the frameworks in which QoS provisioning relies on accurate traffic classification, the precise mapping of traffic flows lays the foundation for differentiated QoS provisioning.

Early-stage traffic classification is crucial for the timely application of QoS policies during flow creation. However, running the traffic classifier solely in the control plane introduces considerable latency and network overhead (X. Chen et al., 2022). The programmability of the network data plane enables the definition of customized matching criteria for traffic type identification (Kfoury et al., 2021), enabling fine-grain classification at a near-line rate. This concept is known as in-

network traffic classification, a form of In-Network Computing (INC) (Kianpisheh and Taleb, 2022) that leverages the reconfigurability of the *Match/Action (M/A)* paradigm (Bosshart, Gibb, et al., 2013). In-network traffic classification involves implementing rule-based Machine Learning (ML) models, such as Decision Trees (DTs), within switches to achieve high-speed processing (Xiong and Zilberman, 2019).

The benefits of in-network traffic classification are multifold. Switches offer high performance, with a latency of sub-microseconds. The performance of distributed ML models is often limited by the time it takes to transfer data between nodes. If a network switch can classify traffic at the same rate it handles packet transmission, it can perform better than a single node. In addition, networking devices can act as the first line of defense by terminating unnecessary data near the edge of the network, reducing traffic load, and improving user experience, particularly for latency-sensitive applications.

Despite these advantages, certain constraints limit the deployment of a full-fledged network traffic classifier inside a programmable data plane (PDP). These constraints stem from the classifier's design and implementation perspective, particularly when using P4-enabled programmable switches. For example, choosing between per-packet and per-flow models introduces a trade-off between packet processing delay and accuracy (Xavier et al., 2021). In the per-flow model, the design constraint involves selecting the input features that affect the classification cost (Chakraborty et al., 2021). Implementation constraints in P4-enabled switches include a lack of support for complex operations and limited memory, thereby placing an upper bound on the performance of the classifier. Hence, accurately and efficiently classifying Internet traffic at the earliest stage of flow creation is still challenging because of the clear trade-off between the classifier's accuracy and processing efficiency, which are competing objectives.

For the **provision of QoS**, the conventional QoS models such as IntServ/RSVP (Wroclawski, 1997; Shenker et al., 1997), DiffServ (Black and Jones, 2015), and DetNet (Finn et al., 2019) focus on optimizing the network to deliver a specific service objective. However, the obtained performance parameters for the requested services are to be measured, which must be provided by the design to satisfy strict performance requirements. Moreover, simultaneously supporting services with varying QoS requirements by deploying per-flow rules results in management at the control plane and resource overhead at the data plane (e.g., queues and memory) (Karakus and Durresi, 2017).

Data plane programmability enables the customization of the forwarding and processing logic of the device to meet the QoS requirements of traffic flows (Kianpisheh and Taleb, 2022). It also allows carrying *Service Level Objectives SLOs*, that is, targeted latency as part of packets, and programs the traversing nodes to validate the service objectives and take appropriate actions for each packet (Clemm and Eckert, 2020). Hence, applications/end users can request a specific QoS from the network without keeping flow-specific rules throughout the network, also allowing for service delivery by design to meet stringent performance requirements (Turkovic et al., 2021). However, if traffic classes are not precisely identified and prioritized, low-priority traffic that previously experienced high latency might be given preference over higher-priority traffic at subsequent nodes along the path. In addition, embedding QoS requirements from the application layer necessitates customized integration of the application layer with the transport and network layers.

Our previous work (Saqib, Hmitti, et al., 2022) proposed a novel approach to maintaining accuracy and processing efficiency for ML models in the data plane. However, the previous design relied on using the output of the ML model as *if-else* rules embedded directly within the P4 code, which are then uploaded to the switch

as firmware. Consequently, updating the model required regenerating, uploading, and rebooting the device, reducing the adaptability of the system to real-time changes.

In contrast, this paper presents a more dynamic approach by mapping the output of the ML model to *Match-Action Tables (MATs)*, allowing real-time updates and adaptation without requiring firmware regeneration. Additionally, while the former work demonstrated classification effectiveness at the switch level, this paper considers the broader implications of using a rule-based model for effective QoS provisioning, extending beyond simple classification to include forwarding decisions and QoS mechanisms. We also address the generalizability of our approach to another IoT dataset, further enhancing its applicability in diverse network environments.

Our framework distinguishes itself from closely related works (Xiong and Zilberman, 2019) and (Xavier et al., 2021) by integrating an efficient and precise traffic classification algorithm with a stateless packet processing mechanism. Specifically, (Xiong and Zilberman, 2019) discusses using ML models in the data plane without considering the performance trade-offs, while (Xavier et al., 2021) investigates the balance between accuracy and efficiency, highlighting the challenges of managing these competing objectives. Our work addresses these challenges by providing a classification mechanism that achieves both high accuracy and processing efficiency for effective QoS provisioning. The key contributions of this study can be summarized as follows:

- We introduce an innovative framework with architectural components that achieve early-stage traffic flow classification within the data plane and enable QoS-aware packet processing by carrying service objectives as part of the packets.

- We employ an in-network traffic classifier to enhance QoS provisioning for a few selected latency-critical IoT applications. Specifically, our approach involves the on-the-fly embedding of *SLOs* into the header of the packets, enabling the network to process packets in alignment with defined service objectives. This integration serves as a complementary approach to classical and objective-based priority queuing, with a focus on striking a balance between the rate of deadline violations and bandwidth utilization in networks characterized by heterogeneous traffic demands.
- We conduct an extensive experimental performance evaluation of the framework, showing that the proposed classification method maintains processing efficiency and accuracy despite the limitations imposed by PDP. In addition, embedding *SLOs* into the header of packet enables stateless and service-aware traffic processing, leading to better utilization of network resources.

The remainder of this paper is organized as follows. In Section 3.2, we present an overview of related studies. Section 3.3 provides detailed insights into the design and implementation of the proposed framework. In Section 3.4, we explain our methodology for validating the framework in both traffic-type identification and QoS-provisioning scenarios. The experimental results are presented in Section 3.5, and conclusions are drawn in Section 3.6.

3.2 Related work

Recently, there has been an unprecedented surge in research devoted to network traffic management to meet the QoS demands of end users. However, researchers have focused on enhancing the network traffic classification process or improving QoS provisioning efficiency. Simultaneously, it is crucial to consider both aspects

to effectively handle the diverse traffic demands of emerging applications. Table 3.1 summarizes the state-of-the-art devoted to traffic classification or effective QoS provisioning and positions the objective of our framework as an integrated solution. In this comparative study, the classification process can be implemented in either the control plane or the data plane, wherein the data plane is the preferred option due to its low latency and minimal network overheads. Similarly, QoS provisioning methods can be categorized as either stateful or stateless, with stateless methods being more advantageous due to their reduced overhead and enhanced utilization of network resources. Finally, the first two performance parameters, accuracy and efficiency, are directly related to the classification process, whereas the utilization (i.e., to avoid over/under provisioning) parameter is related to QoS provisioning.

	Architecture			**Algorithms**	**Performance**		
	Classification	**Data Plane**	**QoS Provisioning**	**Classification/ QoS**	**Accuracy**	**Efficiency**	**Utilization**
Ilsy (Xiong and Zilberman, 2019; Zheng, Xiong, et al., 2024)	*Y*	*Y*	*N*	*DT, RF*	*N*	*N*	*N*
Planter (Zheng and Zilberman, 2021; Zheng, Zang, et al., 2024)	*Y*	*Y*	*N*	*DT, RF*	*N*	*Y*	*N*
pForest (Busse-Grawitz et al., 2019), SwitchTree (Lee and Singh, 2020)	*Y*	*Y*	*N*	*RF*	*Y*	*N*	*N*
MAP4 (Xavier et al., 2021; Xavier et al., 2022)	*Y*	*Y*	*N*	*DT, RF*	*Y*	*N*	*N*
Mousika (Xie, Q. Li, Dong, et al., 2022; Xie, Q. Li, Duan, et al., 2023)	*Y*	*Y*	*N*	*BDT*	*Y*	*Y*	*N*
IntServ/RSVP (Wroclawski, 1997; Shenker et al., 1997), DiffServ (Black and Jones, 2015)), MPLS (Tamura et al., 2004), DetNet (Finn et al., 2019)	*N*	*N*	*Stateful*	*Priority Queueing*	*N*	*N*	*Y*
ProgLab (Froes et al., 2020)	*Y*	*N*	*Stateful*	*Priority Queuing*	*N*	*N*	*Y*
SP-PIFO (Alcoz et al., 2020), QVISOR (Gran Alcoz and Vanbever, 2023)	*N*	*N*	*Stateless*	*PIFO Queuing*	*N*	*N*	*Y*
LBF (Clemm and Eckert, 2020), P4QoS (Turkovic et al., 2021)	*N*	*N*	*Stateless*	*AQM*	*N*	*N*	*Y*
This study	*Y*	*Y*	*Stateless*	*DT, AC-PIFO Queueing*	*Y*	*Y*	*Y*

Table 3.1 Summary of the related work

DT: Decision Tree; BDT: Binary Decision Tree; RF: Random Forest; AQM: Active Queue Management; PIFO: Push-In-First-Out.

This section provides an overview of existing in-network traffic classification methods and highlights their limitations. Additionally, it outlines the available QoS provisioning mechanisms, emphasizing the crucial aspects that necessitate the efficient management of multi-priority traffic demands. Moreover, the discussion highlighted the need for an integrated traffic management framework capable of

intelligently handling heterogeneous traffic demands.

3.2.1 In-network traffic classification

Networking devices offer two major advantages over other computing devices: location and data processing speed. Thus, INC results in an improved throughput and latency (Dang et al., 2020). Being a class of INC, in-network traffic classification has become an emerging trend that brings remarkable benefits to the network (Xiong and Zilberman, 2019). Consequently, there has been a growing interest in integrating the output of ML models into the data plane for early stage network traffic classification (Zhang et al., 2021; Xavier et al., 2022; Xie, Q. Li, Dong, et al., 2022).

While in-network traffic classification is pivotal for developing early stage solutions, the limitations of the PDP can impact the applicability of a comprehensive traffic classifier. For instance, the research presented in (Xavier et al., 2021) emphasizes the inherent trade-off between classification accuracy and processing efficiency in the data plane. Individual packet-based classification is known for its low processing delay but tends to sacrifice accuracy. By contrast, per-flow classification prioritizes accuracy at the expense of efficiency. Aggregating information from multiple packets in the per-flow case improves model training but requires more resources to maintain the flow states and derive useful information. However, a per-packet model is more efficient but needs aggregated measurements and temporal correlation learning.

In the existing literature, IIsy (Xiong and Zilberman, 2019; Zheng, Xiong, et al., 2024) and Planter (Zheng and Zilberman, 2021; Zheng, Zang, et al., 2024) took the initial steps in embedding tree based ML models in the network data plane by representing DT branches using *MATs*. Subsequent research efforts have aimed

to enhance the accuracy and processing efficiency of classifiers in the data plane. However, these efforts often prioritize one aspect over another: accuracy or efficiency. For instance, the *Mousika* framework (Xie, Q. Li, Dong, et al., 2022; Xie, Q. Li, Duan, et al., 2023) addresses classifier efficiency by modifying DTs to Binary Decision Trees (BDTs), supporting faster training, and generating fewer rules to improve switch constraints. However, the major limitation of the design is the consideration of stateless per-packet features while ignoring stateful flow-level features, which requires several complex modifications in P4 (Zhou et al., 2023). The necessity of flow-level features is experimentally demonstrated in (Xavier et al., 2021). Conversely, *MAP4* (Xavier et al., 2022) aimed to create simple and accurate ML models that can be deployed in the data plane. Despite P4-imposed limitations, the framework maintains acceptable accuracy, but relies on resource-intensive (i.e., statistical) features for using a flow-specific model. Furthermore, the input features, particularly port numbers, used to accurately identify flows, may be ineffective in the case of encrypted traffic. Hence, the decision to adopt a per-packet or per-flow approach significantly affects the classifier's accuracy, efficiency, and reliability, posing a notable challenge in achieving a harmonious balance among these factors.

3.2.2 QoS provisioning

Over the past two decades, several QoS models have been proposed, including IntServ/RSVP (Wroclawski, 1997; Shenker et al., 1997), DiffServ (Black and Jones, 2015), Multiprotocol Label Switching (MPLS) (Tamura et al., 2004), and DetNet (Finn et al., 2019). DiffServ and IntServ are complementary QoS provisioning mechanisms, where DiffServ manages resources between different traffic classes and IntServ provides per-flow bandwidth guarantees through resource reservation (Wroclawski, 1997; Shenker et al., 1997). DetNet is a recent proposal that offers per-flow service guarantees for maximum end-to-end latency, bounded jitter, packet

loss ratio, and out-of-order packet delivery bounds (Finn et al., 2019).

The management of diverse QoS policies utilizing conventional QoS models, such as DiffServ and MPLS, presents significant challenges due to the static nature of traffic classes with limited granularity of QoS levels and the necessity for installation of specialized software or hardware components within the network infrastructure. The emergence of network programmability, that is, Software Defined Networking SDN (Liatifis et al., 2023), further enriches the management of service models and mechanisms using the support of Application Programming Interfaces (APIs). The global network view and fine-grained measurements enable the implementation of a wide range of network policies and rapid service deployment (Froes et al., 2020; Monge and Szarkowicz, 2015).

In SDN-based networks, integrating precise network measurements with a centralized control plane is vital for the intelligent management of network resources. This integration has spurred the development of various QoS management approaches including QoS-based controller design, dynamic resource allocation, queue scheduling, and QoS-driven optimal routing (Khan et al., 2021). The rapid development of PDP has further accelerated the adoption of ML / Artificial Intelligence (AI) for efficient packet processing and forwarding, resulting in improved network throughput and reduced delay (Quan et al., 2022). AI-enabled PDP contributes to throughput improvement by providing fine-grained measurements to the control plane, enabling the identification of optimal paths (Hardegen and Rieger, 2020). Concurrently, the reduction of network delay primarily focuses on mitigating scheduling i.e., queuing delay, which is a significant contributor to the overall delay (Papagianni and De Schepper, 2019). Programmable packet scheduling allows the operators to specify (new) scheduling policies based on ranks (i.e., priorities) to achieve QoS (Sivaraman et al., 2016). It dynamically adapts the mapping between packet rank and available strict priority queues (Alcoz et al., 2020). The underlying resources of

commodity switches can be further virtualized to enable efficient resource sharing among multiple tenants (Gran Alcoz and Vanbever, 2023).

The closed integration of the control and data planes is crucial for guaranteeing QoS by optimizing resource allocation and traffic forwarding strategies. However, with the emergence of services with stringent QoS requirements (e.g., loss, bandwidth, and latency), the spectrum of application requirements that need to be supported by the network is expanding (De Alwis et al., 2021). This expansion of application requirements necessitates that networks frequently process thousands of flows simultaneously, each with distinct QoS requirements. Consequently, implementing flow-specific rules at network nodes demands substantial memory and computational overhead, rendering them impractical.

The emergence of data plane programmability has presented an opportunity to include *SLOs* within the packet header and apply QoS-aware operations on the packet while traversing the network (R. Li et al., 2018). For instance, Latency Based Forwarding (Clemm and Eckert, 2020) incorporates the latency objective as packet metadata, allowing the network to take differentiated actions based on the remaining latency budget of each packet. Similarly, another study focuses on QoS-enabled queue management to prioritize packets based on service objectives, ensuring that flows with the lowest deadlines are served first (Turkovic et al., 2021). However, without the precise prioritization of traffic classes, the previously encountered high-latency, low-priority traffic may gain preference over higher-priority traffic at subsequent nodes along the path.

Therefore, there is a need for an intelligent and integrated traffic management framework that enables the precise classification of incoming packets at the earliest stage of flow creation and embeds the associated *SLOs* within the packets, allowing the network to process packets based on their class priority and service objectives

in a stateless manner.

3.3 Intelligent programmable traffic management

This section comprehensively overviews our proposed framework for smartly managing multi-priority traffic demands. We begin by discussing the background of in-network ML using programmable network devices, the design requirements that underpin the framework, the proposed system architecture, and a detailed explanation of its implementation.

3.3.1 Background

Recent advancements in network devices, such as Switch-ASICs, smart network interface cards (NICs), and FPGA-based network devices (Ibanez et al., 2019), extend beyond high performance to incorporate programmability using domain-specific languages, such as P4 (Bosshart, Daly, et al., 2014). The programmability of network data plane facilitates offloading of ML operations to the network by introducing the concept of in-network inference (Zheng, Hong, et al., 2023). During this process, the training phase typically resides in the control plane owing to its high operational complexity, whereas the inference phase transitions to the data plane.

In this scenario, the trained ML model undergoes mapping to align with the *M/A* pipeline of the device. This alignment empowers the model to label data in a data plane based on the learned mapping. The programmable nature of these devices, facilitated by languages such as P4, enables the seamless integration of ML capabilities into the network infrastructure. As a result, in-network inference becomes a viable and efficient approach for handling ML tasks, with the training and inference phases strategically divided between the control and data planes.

The structure of tree-based learning algorithms (James et al., 2023) makes them well-suited for application in programmable network devices, particularly when the computational logic of a network device pipeline mimics a tree structure. Therefore, researchers have explored the integration of various tree-based learning algorithms directly into programmable network devices for classification (Zheng, Hong, et al., 2023). Two primary mapping strategies exist for deploying DT to the data plane: a depth-based approach and an encode-based approach.

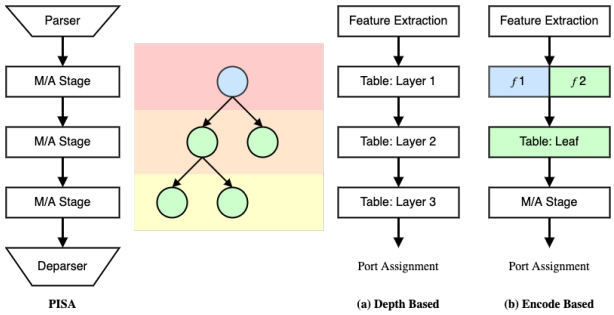


Figure 3.1 Mapping of DT to a M/A pipeline: (a) Depth-based approach and (b) Encode-based approach

In the depth-based approach (Busse-Grawitz et al., 2019; Lee and Singh, 2020; Xavier et al., 2021) (illustrated in Fig. 3.1 (a)), the tree model is hierarchically mapped onto the M/A pipeline. The execution of the model occurs layer-by-layer within the pipeline until it reaches the leaf node. Starting with the extraction of the required features from incoming data, each layer of the DT involves comparing a feature's value with a threshold. The subsequent layers utilize the node ID and comparison results to extract the node information for the current layer.

However, the encode-based approach (Xiong and Zilberman, 2019; Zheng and Zilberman, 2021; Zheng, Zang, et al., 2024; Musumeci et al., 2022) (depicted in Fig. 3.1 (b)) disrupts the hierarchical structure of the tree model. It encodes each feature based on the split value in each branch of the tree and treats each branch

as a one-time slice in the feature space. After slicing, each feature range receives a label code, and a code-to-leaf mapping (M/A table) is employed to map the combined codes of all features to the leaf node.

The former approach, while consuming less memory because of the limited number of nodes in the tree, requires more stages. The number of stages depends on the depth of the tree, creating a trade-off. A simplified version of the depth-based approach may use if and else statements instead of *MATs*, offering intuition but requiring more lines of code without saving stages (Xavier et al., 2022).

The encode-based approach utilizes relatively small stages because feature tables have no dependencies and can share stages. The memory consumption of this approach is also usually less than that of depth-based approaches when the model size is not large (Zheng, Xiong, et al., 2024). A simplified version of the encode-based approach involves using only one leaf (decision) table for classification, using feature values as input, and employing range-matching for labeling instead of a mapped code (Zhang et al., 2021).

3.3.2 Design requirements

The proposed framework design, which includes traffic flow classification and QoS provisioning, enables customizable management of multi-priority traffic. We identified several key requirements that the design must satisfy to ensure the effectiveness and practicality of the framework.

Traffic classification: In addition to the usual requirements for an ML-based traffic classification model (e.g., accuracy and generalization), we aim to build a model, f, that satisfies the following properties when classifying an instance x:

- The operations required to compute $f(x)$ must be readily implementable in

P4, ensuring compatibility with the data plane environment;

- To allow for real-time classification, computing f(x) must be computationally efficient, minimizing the impact on packet processing times;
- A flow must be classified as early as possible, ideally within the first few packets, to ensure timely application of QoS policies.

QoS provisioning: when processing flows with varying service quality requirements, the design must meet the following properties:

- Avoid management and resource overhead when accommodating flows with heterogeneous traffic demands;
- Increase the network resources utilization without over/under-provisioning the network resources.

The traffic classification mechanism is stateful, as it employs a hash and counter to track flow-level information on the switch where the classifier operates. However, after classifying traffic, the remaining switches in the network no longer need to store QoS policies for multi-priority traffic. Instead, the *SLOs* embedded in each packet allow these switches to take stateless actions, processing packets in line with the service objectives without maintaining a flow-specific state.

3.3.3 System architecture

The architecture of the proposed system is shown in Fig. 3.2, incorporating four key components. These components include an in-network traffic classifier, extending the packet header to embed QoS parameters, applying admission control policies to validate service objectives, and utilizing priority queuing to schedule the differentiated transmission of packets. The design for each is described as follows:

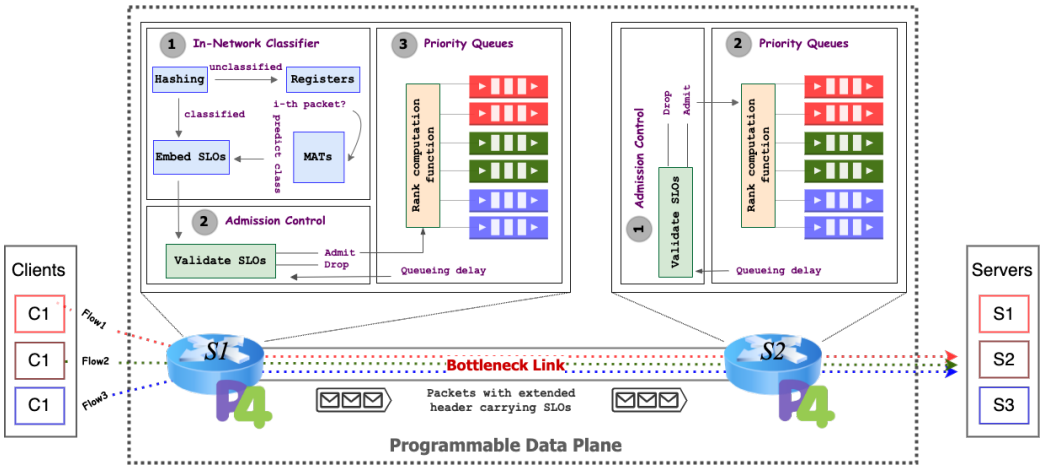


Figure 3.2 Proposed system architecture

3.3.3.1 In-network traffic classifier

Packet and flow are the core objects that need to be identified from headers and payloads for traffic classification. Owing to a trade-off in using packet- and flow-based classification methods, performing early stage traffic flow classification with high accuracy remains challenging. In addition, designing a traffic classifier for PDP is also challenging owing to lack of support for complex operations in the P4 language and limited memory at its target (Sharma et al., 2017). Ideally, the PDP should store only a limited number of features to process thousands of flows concurrently, which sets an upper bound on the traffic classifier performance. Therefore, it is necessary to design a classification algorithm that fits the constraints of the P4 language (i.e., no floating points and loop-control) and its target (i.e., limited memory). Thus, the classifier must use minimal features and identify flows at the earliest stage while respecting the data plane constraints to optimize the performance.

Our classifier design is based on a novel and effective traffic classification method that can quickly and accurately classify various traffic classes. This is particularly

important for traffic management in latency-sensitive applications. Our approach relies solely on a single and stable feature, namely, sequential packet size information, which can be directly extracted from the header of the packets. This is in contrast to conventional methods that use time-related metrics such as min/max/mean packet size and inter-arrival time, which are resource intensive and can be unstable and easily impacted by factors such as bursty heavy loads and traffic congestion (Erman et al., 2007).

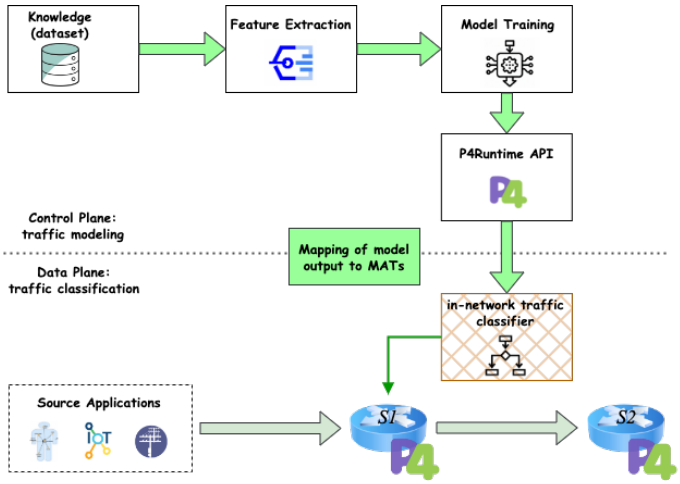


Figure 3.3 An overview of in-network traffic classification process.

The complete in-network classification process is illustrated in Fig. 3.3, mainly consisting of two phases: offline training of the ML model at the control plane and online inference in the data plane. In the first phase, the control plane trains the ML model on a given dataset. The resulting output is then translated into *MATs* of a programmable switch for online inference. An encoded-based approach is used because this method uses a relatively small number of stages and feature tables have no dependency and can share stages, requiring n parallel feature tables and one decision table.

There are many supervised learning approaches in the literature for traffic char-

acterization, but not all are appropriate for implementation at P4 (Xavier et al., 2021). In other words, because we aim to embed the ML model output into the data plane, the necessary operations in the targeted model must be readily available in P4. Therefore, we uses a classical DT algorithm to address the P4 limitations. Given the current primitives of P4 language (Bosshart, Daly, et al., 2014), a DT based classifier is more suitable for such tasks. Only a comparison operation is required to classify element x, which can be easily expressed in *MATs*.

3.3.3.2 Packet with extended header

In addition to the simultaneous processing of heterogeneous traffic flows, a well-defined *SLO*, i.e., network latency, is required to guarantee certain levels of user experience. However, simply optimizing a network to minimize latency and performing measurements to determine what latency is delivered is no longer sufficient. Instead, it is necessary to quantify the specific target latency. This can refer to ensuring a latency that should not be exceeded, known as an *in-time guarantee*. In certain cases, it can even involve achieving a specific target latency window, referred to as an *on-time guarantee* (Clemm, Zhani, et al., 2020).

We employ the capabilities of the PDP to incorporate service objectives within the packet's header, ensuring precise adherence to service quality requirements while aligning with system design considerations. Utilizing an In-Network Telemetry (INT) (Tan et al., 2021) header as a QoS header, as illustrated in Fig. 3.4, facilitates the embedding of QoS requirements. This extended header encompasses the $ClassID$ and associated latency, dynamically updating over traversing nodes by subtracting the packet's experienced latency. Such a design strategy empowers applications or end-users to specify desired QoS from the network.

The in-network traffic classifier, positioned at the edge of the network, is pivotal

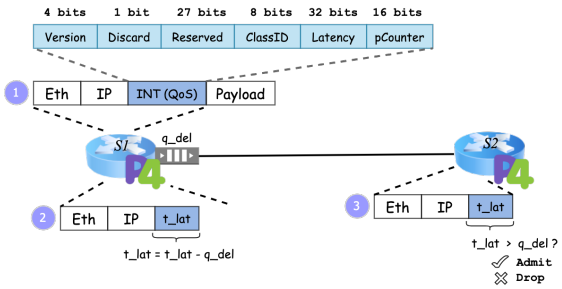


Figure 3.4 Extended QoS header for carrying *SLO*.

in incorporating corresponding QoS requirements within classified packets. By including service objectives as an integral part of the packet, the network gains the ability to tailor flow behavior according to the specified QoS, offering a customizable approach to meet diverse application needs.

3.3.3.3 Admission control

Despite implementing current QoS mechanisms, some packets may still experience higher latency than the specified threshold, resulting in unnecessary consumption of resources and negatively impacting the competing flows. Therefore, validating packets as they traverse the network is essential, prioritizing them based on service objectives or dropping them when necessary to ensure resource utilization.

Our design approach involves implementing admission-control logic in the ingress pipeline of the switches. When incoming packets with QoS parameters arrive, the switches validate their service objectives and take appropriate action. In our case, the extended header fields utilize the class priority and maximum end-to-end latency as indicators for queues to execute differentiated actions during resource contention at the output port. This approach enables us to admit and prioritize packets that satisfy specified latency requirements while dropping or rejecting packets that exceed their latency deadline. By doing so, the switch schedules packets

per objective, ensures network resource utilization and frees up the bandwidth for competing flows to minimize the risk of resource contention.

In the case of TCP traffic, the dropping mechanism of the admission control could lead to loads of retransmission. While standard congestion control mechanisms handle retransmissions in TCP — a well-studied area in networking — our focus remains on the intelligent management of traffic demands to minimize the risk of congestion due to resource contention rather than addressing TCP-specific congestion issues.

3.3.3.4 PIFO-accelerated priority queuing

Low latency is a critical requirement for emerging network services, making delay performance one of the most important measures to be focused on. When discussing network delay, we refer to the interaction time between a user sending a request packet and receiving a reply packet. The delay can be classified into transmission, processing, and queuing delays, with queuing delay being the most significant factor. To reduce the queuing delay, the network employs Active Queue Management (AQM) mechanisms (Adams, 2012). These mechanisms use a variety of priority queuing algorithms to control the delay of flows by adjusting the forwarding order of packets. The programmability of the data plane renders it an ideal platform for deploying AQM algorithms (Papagianni and De Schepper, 2019; Qiao and Gao, 2022). However, implementing classical AQM algorithms on programmable pipeline-based ASICs is challenging because of the architectural constraints of these ASICs (Kunze et al., 2021).

A Push-In First-Out (PIFO) queue is a popular abstraction for programmable packet scheduling (Sivaraman et al., 2016). PIFO associates a rank with each packet and maintains a sorted queue to buffer packets. Newly arrived packets are

inserted into the queue based on their ranks, and the packets are dequeued from the head. Different packet scheduling algorithms can be implemented on top of PIFO by changing the rank computation function, that is, scheduling based on the Shortest Remaining Processing Time (SRPT) (Schrage and Miller, 1966) to minimize the flow completion times.

Our proposed design for the differentiated transmission of packets is primarily based on the utilization of PIFO-accelerated priority queues. To prioritize each traffic class, we use multi-priority queues that schedule the most critical packet first, even within the same priority level. The programmability of PIFO lies in its rank computation function, which allows us to customize the sorting order of the packets. We achieve this customization using an extended packet header with an admission control component that decides packet admission and ranking based on the criticality level of the packet, such as its class priority and remaining latency budget. Hence, this approach enables better QoS with a more efficient utilization of network resources.

3.3.4 Implementation

This subsection presents the necessary steps to implement the complete system design, from traffic-type identification to QoS provisioning. The implementation of each component is discussed in detail.

3.3.4.1 In-network traffic classifier

A high-level classification process is shown in Fig. 3.3, where the control plane trains the ML model on the dataset and translates the resulting outputs to the data plane for online inference. The given dataset S consists of packets p_j of subflows $(f_i(1:j))$ and is further split into training S_T and testing S_P samples.

After preparing the dataset, a DT algorithm is used to characterize the traffic, and the resulting outputs are translated into *M/A* rules. A control plane API called P4Runtime embeds *M/A* rules into the *MATs* of the switch for online inference.

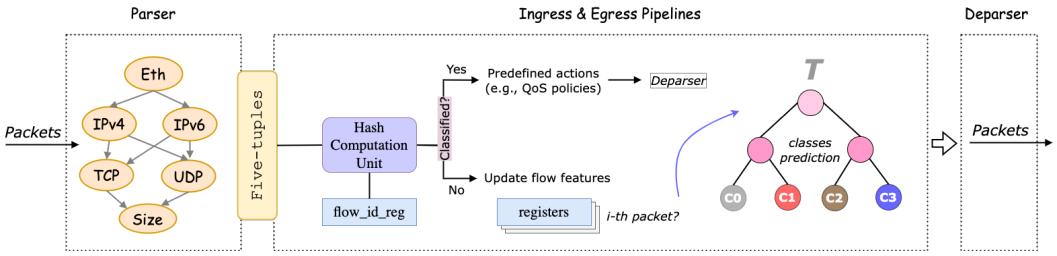


Figure 3.5 Online inference process

The next step is to predict the *flow class* for the given packets as input, that is, the sequential packet size information of the first few packets from each flow. The general process of online inference is shown in Fig. 3.5. The switch maintains a few registers to record the *flow_id*, *packet_sizes*, and *packet_counter* for the first few packets of each flow. The diagram depicts how each packet traversing the device is handled. For incoming packets, the parser module extracts the relevant features (*i.e., five tuples and packet size*) from the header and keeps these feature values in the pipeline's metadata. The next step is to calculate the hash value for each flow based on the *five tuples* field from the packet header. The *flow_id* register tracks all classified flows to treat the belonging packets accordingly. The switch applies corresponding actions to the incoming packets belonging to the identified classes, that is, QoS policies. In other words, the classified flow packets do not necessarily pass through the decision tree process and are processed at a line rate. When the flow is not classified, the switch verifies *packet_counter* for the corresponding flow. Until the *packet_counter* reaches the *threshold*, the parser extracts the *packet_size* and stores it in a *size_vector*. Once the *packet_counter* meets the *threshold*, a classification occurs on the *size_vector* following *MATs*. The flow class is eventually saved in the *meta.class* variable. Detailed steps are

Algorithm 1: In-network traffic classification

Input: *packets, thr: max # of packets, class_latency = [], class_latency_thr = []*

Output: *classes_vector*

class_id = 0; class_vector = []; flow_id=[]; size_vector=[]; queue_delay = []

```
Function InferClass(packets):
    while pkt do
        flow_id = hash(five_tuple);
        if isClassified(flow_id) then
            ApplyAction(flow_id) ;                                  // Function call
        else
            if pkt_counter <thr then
                size_vector[flow_id * thr + pkt_counter]= pkt_payload_len;   pkt_-
                 counter++;
            if pkt_counter == thr then
                class_id = Classes_Prediction(flow_id, size_vector, thr)     pkt_-
                counter++;
End Function
Function isClassified(flow_id):
    if class_vector[flow_id] != 0 then
        return True;
End Function
Function ApplyAction(flow_id):
    pkt_hdr_latency = class_latency[class_id];
    if class_id != 0 then
        meta.rank = pkt_hdr_latency;
        if meta.rank <class_latency_thr[class_id] then
            pkt_queue_priority = high;
        else
            pkt_queue_priority = low;
    if pkt_hdr_latency >queue_delay[class_id] then
        pkt_forward();
        queue_delay[class_id] = current_queue_delay;
End Function
Function Classes_Prediction(flow_id, size_vector, thr):
    size = [];
     for i = 0; i <thr; i ++ do
        size[i] = read_register(flow_id * thr + i);
    for j = 0; j <thr; j ++ do
        match_action_table[j].key = size[j];
    match_action_table[j+1].keys = size[ids];
     class_id = match_action_table[j+1].action[val];
    return class_id;
End Function
```

presented in Algorithm 1.

3.3.4.2 Extending packet header

After receiving and classifying the packets by the ingress switch, it adds an 11-byte INT header as an extended QoS header. The extended header carries the necessary QoS parameters within the packet, enabling the traversing nodes over the path to validate the *SLO* and perform appropriately differentiated actions.

3.3.4.3 Admission control

The QoS parameters carried within the packet's header facilitate the validation of the *SLO* across the network. As illustrated in Fig. 3.4, the switch residing at the edge of the network (left side) updates, for example, the targeted latency t_{lat} of a specific packet p_j belonging to a particular class c by subtracting the experienced queuing delay q_{del} from the current latency budget. When the packet reaches the second switch (right side), it validates the service objective by comparing the remaining latency budget with the current queuing delay of a particular class. It then decides whether to drop or schedule the packet per objective.

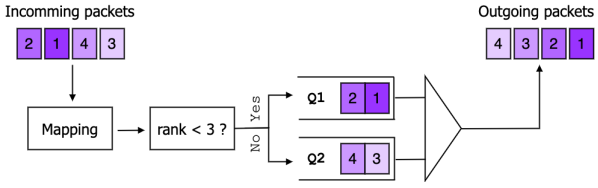


Figure 3.6 Priority queues for a single class of traffic.

3.3.4.4 PIFO-accelerated priority queuing

PIFO-enabled multi-priority queues are maintained for each traffic class to minimize *deadline violation* by prioritizing packets with low latency budgets first, even having the same priority. As shown in Fig. 3.6, multi-priority queues are maintained for a single traffic class. The rank computation function, located at the top of the priority queues, calculates the rank of a packet based on its remaining latency budget and class priority and schedules the packet to the appropriate priority queue.

3.4 Validation methodology

The validation process involves two parts: traffic classification and QoS provisioning. We evaluate the performance of the traffic classifier using two IoT datasets. We then move to the validation of the QoS provisioning part. We describe the datasets we use, the features involved in the classification process and the choice of the ML model selection, as well as our experimental setup in Sections 3.4.1, 3.4.2, and 3.4.3, respectively.

3.4.1 Datasets description

We utilize two IoT datasets, UNSW-IoT (Sivanathan et al., 2018), and Consumer-IoT (Ren et al., 2019), containing traffic from different applications to evaluate the performance of the classifier. We have selected these datasets for evaluation due to their extensive utilization in the existing literature for traffic classification.

We grouped individual devices/applications into broader categories based on their similar QoS requirements. Each dataset is classified into five categories, representing distinct classes with varying QoS parameters such as latency-sensitive traffic or

bandwidth-intensive applications. These categories facilitate the demonstration of the effectiveness of our smart traffic management framework in handling flows with diverse service requirements, rather than exhaustively evaluating the classification accuracy of every class in the datasets. The approach aligns with the primary goal of focusing on QoS provisioning and efficient traffic management.

Dataset	**Class**	**# of flows**	**# of packets**
UNSW-IoT (Sivanathan et al., 2018)	Appliance	218	11,631
	Controller	1,228	46,801
	Home Automation	1,330	21,745
	Sensor	3,259	66,340
	Streaming	3,392	348,125
Consumer-IoT (Ren et al., 2019)	Appliance	4,077	88,269
	Audio	3,401	160,603
	Controller	995	33,794
	Home Automation	479	13,346
	Streaming	25,798	220,599

Table 3.2 Summary of the datasets

The given datasets comprised Packet Capture (PCAP) files collected over several days. To manage the evaluation workload, we select a portion of the datasets. For UNSW-IoT, we considered the interval from Sep. 22nd to Sep. 30th. Consumer-IoT data from 'Interaction experiments' is used. Utilizing all data samples for model development is often impractical and unnecessary. We further apply the well-known k-means cluster sampling method to obtain a representative subset of data samples. Table 3.2 summarizes the obtained subset of the datasets, their classes, and volume of packets/flows in each dataset. It is not our goal to dig into the details about the characteristics of each type of flow, how they were generated, or how they were used for classification purposes. We refer the interested reader to (Sivanathan et al., 2018; Ren et al., 2019) for more information.

3.4.2 Features extraction & ML algorithm selection

Section 3.2.1 discusses the benefits of working with flow-based ML models because of their dynamic features (e.g., duration and cumulative packet size), which cannot be captured using individual packets. However, working with flows has the drawback of not being able to compute better descriptive measures directly (e.g., average, variance, and standard deviation) for time-varying features, because certain operations, such as division and square root, are not supported in P4. Therefore, we followed the feature selection choice proposed by (Saqib, Hmitti, et al., 2022; W. Chen et al., 2021), which uses sequential packet size information as input features to train ML models. Because the extraction process focused on a single future, we applied the same method to both datasets.

Parameter	**KNN**	**XGB**	**RF**	**DT**
n_neighbors	3,**5**,7,9,11	-	-	-
learning_rate	-	0.1, 0.2, **0.3**	-	-
n_estimators	-	7,**9**,11,13	**7**,9,11,13	-
max_depth	-	7,9,**11**,13	7,**9**,11,13	7,9,**11**,13

Table 3.3 Hyperparameters of ML models

The choice of model selection is difficult because of the ML model deployment limitations in P4. Therefore, we first explored the potential of both supervised and unsupervised learning algorithms: DT, Random Forest (RF), XGBoost (XGB), and K-Nearest Neighbors (KNN). We then select the algorithm based on its performance on the given data and deployment feasibility in the data plane.

The hyperparameter settings for each ML model are shown in Table 3.3. The selected values for each hyperparameter are chosen based on specific objectives. For the number of neighbors in KNN, the choice is made to balance bias and variance, with 5 providing optimal results. For XGB, the learning rate is set to 0.3 to ensure robust training and convergence. The number of estimators is set to 9 for

XGB and 7 for RF to optimize model performance while managing computational efficiency. The maximum depth of trees is chosen to control model complexity and avoid overfitting, with values of 11, 9, and 11 being optimal for XGB, RF, and DT, respectively.

3.4.3 Experimental setup

We organized our experiments in three steps. First, we explain our simulation setup in Section 3.4.3.1. Secondly, we describe how we trained our models on the given datasets and deployed them into the data plane in Section 3.4.3.2. Finally, we outline the performance measures for traffic classification and QoS provisioning in Section 3.4.3.3.

3.4.3.1 Simulation setup

The logical components of the simulation setup are shown in Fig. 3.7. The Measurement Component (on the left) is responsible for generating, collecting, and analyzing the network traffic, whereas the Data Plane Component (on the right) is the target to be evaluated. The data plane is implemented in P4 and compiled with the target of the behavioral-model version s (BMv2). The structure of the simulation setup is as follows:

- *Host H1:* the host sends and also receives the packets back already classified and timestamped. This component reports the classifier prediction and network performance of packets.
- *Switch S1:* contains the traffic classification algorithm and QoS provisioning mechanism that classify the incoming traffic and apply QoS policies.

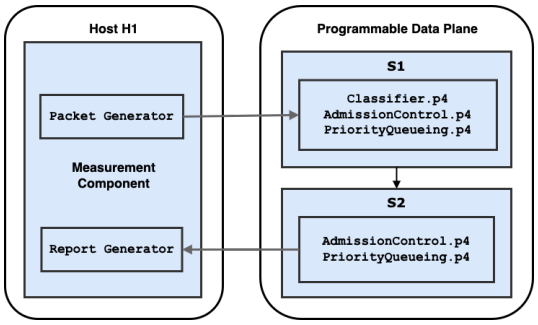


Figure 3.7 Simulation setup

- *Switch S2:* only contains the QoS provisioning part and receives the packets with embedded QoS parameters.

Host 1 generates the traffic with Pktgen-DPDK[1] and *tcpdump* is used to collect the packets to be analyzed at the end of the process. The output of the ML model is deployed into the *MATs* of *S1* for traffic class prediction of incoming packets. The *firmware* of *S1* also contains the QoS provisioning part, that is, the logic to embed QoS parameters into the packet header, validates the *SLO*, and applies priority queuing as per the service objectives. The *S2* receives classified packets with embedded *SLO*; therefore, in *S2*, the *firmware* only contains the QoS provisioning part.

3.4.3.2 ML model training & deployment

For ML models training, we relied on the classifier implemented in Python's *scikit-learn*[2]. The selection of optimal hyperparameters, such as tree height and minimum number of items per leaf, is accomplished through cross-validation to mitigate

[1]https://pktgen-dpdk.readthedocs.io/en/latest/

[2]https://scikit-learn.org/

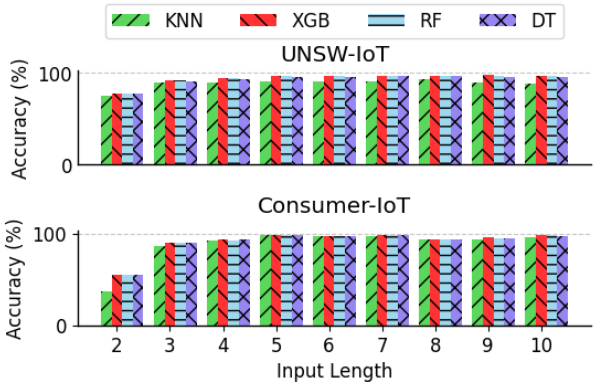


Figure 3.8 Accuracy of ML-models over the chosen datasets.

overfitting. The training process utilized 70% of the data, and the remaining 30% is allocated for validation and testing. The average accuracy of the selected models (DT, RF, XGB and KNN) on the given datasets (UNSW-IoT and Consumer-IoT) can be seen in Fig. 3.8. The *x-axis* represents the *input length* (i.e., number of packets to take classification decision) and the *y-axis* represents the obtained average accuracy of ML models. It can be seen that the *accuracy* of all models increases with an increase in *input length*. Regarding comparing the performance of the algorithms, the decision tree algorithms (i.e., DT and RF) outperformed KNN while competing with each other. RF appears to exhibit a slight (i.e., 1%) superiority over DT in the UNSW-IoT dataset, particularly when the *input length* is set to 4 and 5. Nonetheless, this improvement is accompanied by additional deployment costs in the data plane, leading to noticeable latency degradation in the models, as anticipated (Xavier et al., 2022). Therefore, we decided to use DT to satisfy the properties listed in Section 3.3.2. For more details on these algorithms to justify our choice, we refer the interested reader to the manuscripts (Xiong and Zilberman, 2019; Xavier et al., 2021).

The DT classifier produces an output translated into *MATs* in the form of *M/A* rules. A single table handles the size range of a single packet from the first few

packet sequences of an individual flow. The number of tables is equal to the *input length* plus a class prediction table. In our case, the *threshold* for the *input length* is selected based on the performance of the ML model on the data. We measured the impact of the *input length* on the flow performance by classifying the network traffic based on different *input lengths* . Before applying *MATs*, the P4 code extracts the payload sizes of the sequential packets from an individual flow and stores them as a *size vector* in *SRAM*. Once the threshold of the textitinput length is reached, the switch traverses the *size vector* through the *MATs* to predict the traffic class.

3.4.3.3 Performance measures

This part explains the performance measures of the network traffic classifier and QoS provisioning.

Classification: the classification experiment consists of two parts: measuring the performance of the traffic classifier over the datasets in offline mode and evaluating its performance during online inference. The offline phase uses the conventional performance indicators to test the ability of a multi-class classifier (Grandini et al., 2020). To evaluate the performance of the classifier in the data plane, we use the *True Positive Rate (TPR)* as a performance indicator. *TPR* is the ratio of the total number of correctly classified positive samples to the total number of positive samples:

$$TPR = \frac{TP}{(TP + FN)} \tag{3.1}$$

True Positive (TP) means the observation is positive, and the sample is predicted to be positive. *False Negative (FN)* is when the observation is positive, but the sample is predicted to be negative. To measure the accuracy of per-class flows, we used *TPR*, which represents the percentage of correctly predicted per-class

flows (Poupart et al., 2016). The objective is to achieve a high *TPR* for all classes, indicating accurate predictions.

The generation of results from the offline training phase is straightforward; however, for online inference, experiments are conducted in a controlled environment. After sending packets through the data plane and collecting the classification results, we evaluated the performance of the model using the aforementioned metric.

QoS provisioning: the following metrics are used to measure the effectiveness of the QoS provisioning mechanism:

- *Deadline violation (DV):* the percentage of packets that exceed their predefined latency threshold. This occurs when packets from a particular class experience latency higher than the defined threshold.
- *Outdated received packets (OP):* the percentage of packets that violate the deadline but are transmitted in the network is useless and unnecessarily consumes the network resources.
- *Bandwidth utilization (BU):* the ratio of received bandwidth by a particular application/traffic class to its total generated data rate.

3.5 Experimental results

In this section, we present the results of our classification approach and demonstrate that the extended packet header enables efficient packet processing. First, we highlight the strength of our single-feature-based traffic classifier across two different datasets: UNSW-IoT and consumer-IoT. We then demonstrate the effectiveness of carrying the QoS parameters inside the packet header for provisioning. The results of our QoS provisioning approach illustrate the contribution of this extended packet header to effective and efficient QoS provisioning.

3.5.1 Classification results

The classification results consist of ML-model training and online inference.

Class	Precision	Recall	F1-Score
Sensor	0.99	0.99	0.99
Streaming	0.98	0.82	0.89
Home Auto.	0.99	0.99	0.99
Controller	0.84	1.00	0.91
Appliance	1.00	0.98	0.99

Table 3.4 Classification results from UNSW-IoT dataset

Class	Precision	Recall	F1-Score
Audio	0.97	0.98	0.98
Streaming	0.98	0.96	0.97
Home Auto.	0.99	1.00	0.99
Controller	0.98	1.00	0.99
Appliance	0.99	0.98	0.98

Table 3.5 Classification results from Consumer-IoT dataset

Model training: we first present the classification results obtained from offline model training on the selected datasets. Since traffic classification at the earliest stage of flow creation is highly desirable, we use *input length* as a performance indicator. Fig. 3.8 illustrates the total *accuracy* of the selected algorithms across the datasets. The *accuracy* improves as the *input length* increases, with all algorithms achieving over 90% *accuracy* at a length of *four*. In addition, increasing the input length increases the class identification time. Therefore, we set the *input length* threshold to *four*.

We consider the classical DT as our traffic classification algorithm because of its competitive performance on data and its feasibility for deployment in the data plane. As shown in Fig. 3.8, performance of classical DT is on par with advanced ensemble methods such as RF and XGB. To support the competitiveness of DT, we visualized the similarities in the input data and presented them for both datasets

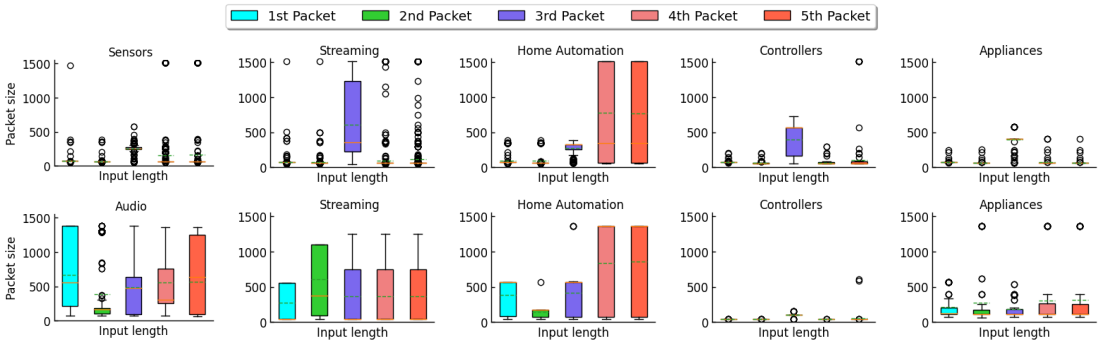


Figure 3.9 Input similarities between sequential packet sizes from UNSW-IoT (top) and Consumer-IoT (bottom) datasets

in Fig. 3.9. As sequential packet sizes are input features, the size ranges differ over *input length* and between traffic classes, allowing the traffic classifier to identify the traffic classes properly. Table 3.4 and 3.5 presents the complete classification results from offline model training using the performance metrics of *precision*, *recall*, and *F1-Score*.

Online inference: the next step of the experiment is about testing the ML-model in the data plane for online inference. Here, we discuss the results obtained from the data plane, including *TPR*, and the resource overhead added by each ML model to the switch.

TPR: we sequentially deployed the output of ML model to the data plane to evaluate the prediction performance of the classifier. Following deployment of the models, we used packet generator to send packets from the corresponding dataset. Table 3.6 lists the classification results from the data plane. The *TPR* of the classifier obtained from the data plane and the *recall* metric of the corresponding dataset are closely related (see Table 3.4 and 3.5). Hence, the offline trained model provides similar results during online inference in the data plane.

Efficiency: the last set of results concerns the *cost* or *processing efficiency* of the

UNSW-IoT		Consumer-IoT	
Class	**TPR**	**Class**	**TPR**
Sensor	97%	Audio	96%
Streaming	82%	Streaming	96%
Home Auto.	99%	Home Auto.	98%
Controllers	99%	Controllers	99%
Appliances	96%	Appliances	98%

Table 3.6 Classification results from data plane

network traffic classifier within the data plane. As stated earlier in Section 3.2.1, the trade-off between using per-packet and per-flow-based ML models raises the question of maintaining classification accuracy and processing efficiency within the data plane. Although the per-flow model is computationally expensive, we rely on it as it captures traffic characteristics effectively. However, instead of using statistical measures that could impact the processing efficiency of the classifier, our approach relies on using sequential packet payload size information, which can be directly extracted from the packet header. Hence, we use a single feature (i.e., packet size information) and feed it to the model sequentially to capture the flow characteristics.

We compare the processing efficiency of sequential packet size information, called single-feature (SF), to features relying on statistical measures, referred to as multi-features (MF). In the MF case, we use the minimum, maximum, average, and total payload size for the first few, i.e., four flow packets. Simple arithmetic and comparison operations can be directly applied to calculate these statistical measures in the data plane. However, the lack of support for crucial operations (e.g., division and square root) in P4 necessitates using an external function in P4 called *extern* (Silva et al., 2018). Calling the *extern* function for every single flow makes the classification process computationally expensive for the MF model.

A side-by-side comparison of the processing efficiency of SF and MF models under

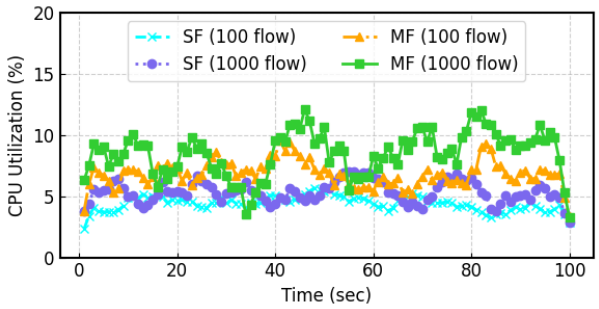


Figure 3.10 CPU utilization over time

different input traffic workloads (i.e., number of flows per second) is shown in Fig. 3.10. The simulation runs for 100 seconds for each model under each input workload (i.e., sending 100 and 1000 flows per second). The CPU utilization for the SF model under both workloads is around 5% and 6%, lower than the MF model under both workloads. The slight increase in the CPU utilization for using the SF model with a higher workload is logical due to the overall traffic load on the host system. In contrast, the utilization for MF model is somehow proportional to SF for both input workloads. The MF model with 100 flows, mostly remains between 6.5% and 8%, while with 1000 flows it approaches 10%.

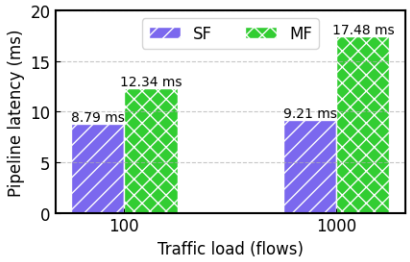


Figure 3.11 Flow identification time

The processing load of the switch impacts the processing delay or pipeline latency for the traversing packets. The average pipeline latency added by both types of models under different input workloads is shown in Fig. 3.11. The differences in

pipeline latencies indicate the impact of computational operations on the processing delay of the packets. The increase in processing load and its effect on the pipeline latency for the MF model, compared to SF, is evident due to the computational operations, such as taking the average, that are not natively supported by the P4 language but require calling *extern*, which affects the processing efficiency of the system.

3.5.2 Evaluating the effectiveness of QoS provisioning

This subsection evaluates the effectiveness of the proposed QoS-provisioning mechanism. In particular, we present how carrying *SLO* as part of a packet allows stateless QoS provisioning to manage heterogeneous traffic demands efficiently. We first present the technical details for the performance evaluation and then move on to the obtained results.

Application	Latency (ms)	Update time (ms)	Data size (bytes)	Device density	Data rate (Mbps)	Class ID	Priority
Class 1: Process automation	50-100	1	40-100	250/km^2	0.2	1	Low
Class 2: Factory automation	0.25-10	1	10-300	250/km^2	0.6	3	Medium
Class 3: Smart grid	3-20	1	80-1000	250/km^2	2	2	High

Table 3.7 Service quality requirements of a few latency critical IoT applications (Schulz et al., 2017)

Operating System (OS)	**Ubuntu 20.04**
Software	BMv2
Packets generated	10,000 packets each class
Generation rate	250 packets each class/second
Packet size	300, 1000, 100 bytes
Priority queues	Low, Medium, and High
Bandwidth	in: 3.1 Mbps, out: 2.3 Mbps

Table 3.8 Technical details of simulation setup

3.5.2.1 Technical details

To evaluate the performance, we employed the network topology detailed in Section 3.4.3.1, featuring one host and two switches. The host generated multi-priority traffic aligned with the service quality needs of three emerging latency-critical applications, as outlined in Table 3.7. To ensure consistent evaluation, we maintained uniform update times and device densities across all classes.

We fixed the packet sizes at 100, 300, and 1000 bytes to maintain consistency in the QoS evaluation and provide a clear comparison across different traffic classes. These sizes correspond to traffic with varying priority levels and allow for a controlled assessment of how the QoS framework handles different service requirements under known conditions. While the packet sizes are fixed for QoS evaluation, the traffic classification still relies on sequential packet sizes. The ML-based classifier dynamically assigns traffic flows to the appropriate QoS groups, ensuring that the system remains flexible and adaptable to various traffic patterns, even beyond the specific conditions used in the QoS evaluation.

The priority factor of the classes is defined based on the criticality levels, which are directly proportional to the required data rates and inversely proportional to latency (De Alwis et al., 2021; Saqib, Elbiaze, et al., 2024). To assess how the data plane would handle this heterogeneous traffic, we deliberately reduced the bandwidth at the outgoing port of *Switch S1* by 25%, thereby causing congestion in the network. The technical details of the simulation setup are listed in Table 3.8.

At *Switch S1*, an in-network traffic classifier operating in the *ingress* pipeline to identify the traffic class and embed the corresponding QoS parameter (i.e., targeted latency) into the packet header. The switch then processes packets in the *egress*

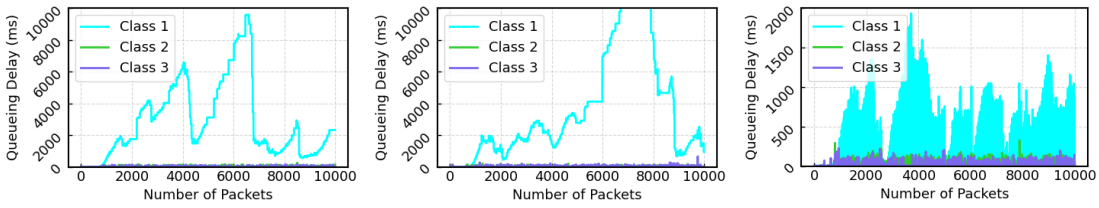
pipeline by utilizing priority queuing to prioritize packets based on their class priority level and latency deadline. If the incoming traffic exceeds the outbound rate limit, packets may experience delays owing to congestion at the outbound link. In such cases, the data plane processes packets per class priority and latency requirements, ensuring that packets with a higher priority and tighter latency constraints are given precedence. The performance metrics defined in Section 3.3.3.3 are used for the evaluation.

To assess system performance thoroughly, we create three different evaluation scenarios by adjusting QoS requirements and network bandwidth to simulate congestion effects. QoS parameters like latency and data rate, detailed in Table 3.7, are varied as minimum, average, and maximum in each scenario to represent diverse workloads. We reduce the bandwidth at outgoing links by 25% in each scenario to induce congestion, facilitating a comprehensive evaluation across priority classes.

We further define three cases to compare the proposed QoS mechanism with state-of-the-art. The first case uses classical priority queuing (Froes et al., 2020) for multi-priority traffic. The second case employs PIFO queuing (Alcoz et al., 2020; Gran Alcoz and Vanbever, 2023) which rank the packet based on latency deadline and dynamically adjust the mapping of ranked packets to strict priority queues. The final case is our proposed admission-control-enabled PIFO queuing mechanism, which validates individual packet *SLOs* before mapping the packets to priority queues.

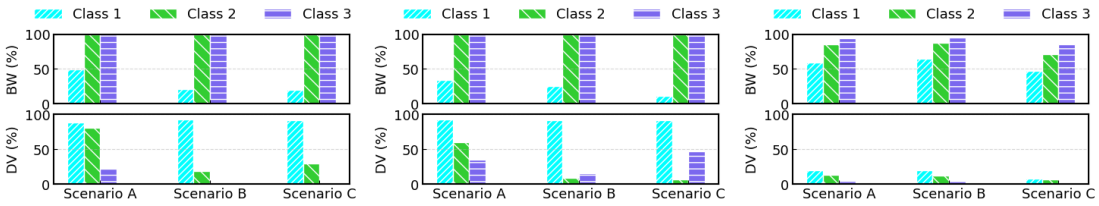
3.5.2.2 Obtained results

Since we target low-latency communications, we consider network latency the primary comparison metric. The evaluation simulates the defined scenarios using existing and proposed QoS mechanisms. In any case, network congestion at the



justification=centering

Figure 3.12 Queuing delay at the outgoing congested link of the switch for scenario C: classical priority queuing (left), PIFO queuing (middle), and proposed AC-PIFO queuing (right).



justification=centering

Figure 3.13 QoS measures obtained over the following switch for scenario C: classical priority queuing (left), PIFO queuing (middle), and proposed AC-PIFO queuing (right).

outgoing link of a switch leads to increased queuing delay for multi-priority packets. However, queuing mechanism handles multi-priority traffic demands during network congestion differently.

The queuing delay of packets using existing and proposed method is shown in Fig. 3.12. During classical priority queuing, high-priority packets (i.e., Class 2 and 3) took precedence, while lower-priority (i.e., Class 1) packets experienced significant delays. Similarly, in the PIFO queuing case, lower-priority packets experienced even worse delays at certain points compared to the first case. In contrast, the proposed QoS mechanism better handles heterogeneous traffic by dropping outdated packets to free up resources for lower-priority traffic, thus improving the overall utilization

of network bandwidth. The queuing delay for low priority traffic is significantly reduced.

Fig. 3.13 specifically shows the performance measures obtained by each QoS mechanism under different evaluation scenarios. The *x-axis* in each subfigure represents the evaluation scenario and the *y-axis* represents the performance measures, i.e., bandwidth utilization and deadline violation rates in percentage for three priority classes. The bars indicate the priority class ID: *cyan* for low priority, *green* for medium, and *blue* for high priority.

In the first case, the higher priority classes obtained the full share of bandwidth while low priority suffered. Due to the precedence of higher priority classes, the low priority classes experienced significantly higher deadline violations due to longer queuing delays. The second case shows a similar behavior toward low-priority traffic. However, the deadline violation for higher priority classes shows a different trend; Class 3 packets received a higher deadline violation rate than Class 2. As PIFO ranks the packets based on the objective, i.e., latency, it dynamically adapts the mapping between packet rank and available strict priority queues. Hence, Class 2 packets took precedence due to their shorter latency deadline, subject to class priority.

In both cases, the higher priority classes receive a full share of bandwidth during congestion but also forward outdated packets whose latency deadlines are violated and may be useless. As a result, the low-priority traffic suffers during all evaluation scenarios. In the last case, validating *SLOs* allows the switch to drop outdated packets while admitting valid packets based on their remaining latency budget. The dropping mechanism for outdated packets leads to better utilization of network resources. As seen in the rightmost subfigure, the higher priority classes still obtain higher bandwidth. Simultaneously, the dropping mechanism for outdated packets

allows the low-priority class to use available bandwidth during resource contention. Additionally, the number of deadline violations is considerably reduced for all classes. Notably, the evaluation show similar trends in the obtained performance measures for each traffic class over each scenario, demonstrating robustness of the proposed system in managing multi-priority traffic during network congestion.

		Classical Priority Queuing (Froes et al., 2020)			PIFO Queuing (Alcoz et al., 2020; Gran Alcoz and Vanbever, 2023)			AC-PIFO Queuing		
Scenario	Class ID	BW (%)	DV (%)	OP (%)	BW (%)	DV (%)	OP (%)	BW (%)	DV (%)	OP (%)
3*A	1	49.12	92.36	41.48	99.97	05.02	05.02	65.31	12.70	0.00
	2	99.97	09.64	09.64	99.51	17.47	17.11	91.61	8.31	0.00
	3	99.97	11.33	11.33	29.25	94.29	23.54	90.04	09.93	0.00
3*B	1	60.08	85.86	45.00	99.97	02.45	02.45	94.98	04.99	0.00
	2	99.97	11.51	11.51	99.97	07.38	07.38	93.10	06.87	0.00
	3	99.97	09.80	09.80	70.36	88.82	59.18	92.96	07.01	0.00
3*C	1	71.58	75.90	47.48	99.97	00.33	00.33	98.86	01.11	0.00
	2	99.97	06.49	06.49	99.97	06.60	06.60	95.36	04.69	0.00
	3	99.97	05.95	05.95	82.51	86.44	69.00	94.11	05.55	0.00

Table 3.9 Comparative analysis of a proposed QoS provisioning method versus state-of-the-art approaches across diverse evaluation scenarios.

A more quantified comparison of the proposed QoS mechanism with existing work under different evaluation scenarios is shown in Table 3.9. The first case makes decisions based on class priority, while the second prefers the objective (i.e., latency) over priority. Both cases lead to over/under-provisioning of network resources. The proposed SLO-enabled admission-controlled queuing mechanism operates with high efficiency, effectively utilizing priority queues to manage diverse traffic demands without the need for network-wide flow-specific rules. Our approach ensures there is no over/under-provisioning of network resources, allowing competing flows (i.e., low priority) to take the use of available resources during network congestion. It is important to note that the proportions of the obtained results are based on the simulation settings.

While extending the packet header holds significance in prioritizing latency-critical packets and enhancing network efficiency, carrying extra bytes in individual packets has a negligible impact on network throughput. As depicted in Fig. 3.14, the ob-

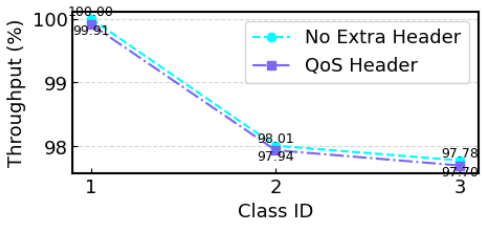


Figure 3.14 Throughput overhead by QoS header

served impact on throughput is marginal when viewed relative to the allocated data rate for each class of traffic. While *Class One* generates packets with significantly smaller sizes, the drop in throughput is expressed as a percentage and remains proportional to its respective data rate, which is configured according to its specific requirements.

3.5.3 Advantages over conventional approaches

3.5.3.1 In-network traffic classification

The obtained classification results reveal that instead of relying on time-related metrics, we can accurately and efficiently identify the network traffic based on the sequential packet size information. The time-related features add up extra resource overhead at the device and may not be consistent and stable enough to serve as classifiers when the network dynamics, which has been verified in many previous studies (Erman et al., 2007). Sequential packet size information is a more stable feature that can be obtained directly from the header of the packet. Because we use packet size information as an input, applications with large messages, such as high-quality video streaming, will cause fragmentation at the network layer, affecting the overall class identification time. However, as shown in Table 3.7, the payload size in emerging and latency-critical IoT applications is less than that of the Maximum Transmission Unit (MTU). As a result, our proposed

approach is potentially applicable to high-precision networks, where latency-critical applications will benefit significantly.

3.5.3.2 QoS provisioning

Services with very strict QoS requirements (e.g., loss, bandwidth, and latency) have emerged and must be simultaneously supported by the network. However, accommodating these varying requirements by deploying per-flow rules would cause significant management and resource overhead (e.g., queues, memory) (Karakus and Durresi, 2017). In contrast, our approach is stateless, enabling applications to request a specific QoS from the network, whereas the networking nodes do not need to keep track of every flow. In particular, the in-network traffic classifier embeds *SLO* (i.e., the maximum allowed latency a packet can experience) in the packet itself and allows the network to process the packets based on the specified objective. The QoS evaluation part shows that having *SLO* as part of the packet allows the traversing nodes to take QoS-aware actions (i.e., processing or forwarding the packets) to meet service-specific QoS demands. At the same time, it also improves the utilization of network resources by reducing the number of marked (outdated) packets.

3.5.4 Discussion

3.5.4.1 Hardware implementation

In addition to the superior performance of the proposed in-network solution in managing heterogeneous traffic demands, the framework can be effectively implemented in hardware known for its high throughput (up to 12.8 Tbps) and low latency (sub-microsecond) (Intel, 2024), far surpassing the capabilities of BMv2. BMv2 is a tool for developing, testing, and debugging P4 data planes, not

a production-grade software switch (*Performance of bmv2* n.d.). Therefore, the throughput and latency performance of BMv2 is significantly less than that of production-grade hardware.

In particular, loading the P4 programs for the tasks mentioned above, would little change the throughput, i.e., the receive throughput (Rx) and transmit throughput (Tx) of the switch (Xie, Q. Li, Dong, et al., 2022). The packet processing delay is expected to remain close to the baseline performance of the device since no complex operations are involved in the classification and QoS provisioning logic (Xavier et al., 2021). The additional QoS header may decrease throughput and increase transmission delay on each output link. However, this overhead is generally constant and typically negligible when using switch Gbps output links (Turkovic et al., 2021).

3.5.4.2 Potential threats of embedding QoS objectives in packets

Embedding QoS objectives or *SLOs* in packet headers introduces potential security threats. Malicious actors could attempt to manipulate or forge *SLOs* to gain unauthorized priority access to network resources or disrupt traffic management protocols, leading to unfair bandwidth allocation or intentional network congestion.

To mitigate these risks, it is essential to implement *SLO* verification mechanisms and packet integrity checks. These mechanisms ensure only authorized entities can modify or interpret the *SLO* data embedded in packets, preventing unauthorized tampering. Moreover, the incorporation of cryptographic techniques such as digital signatures plays a significant role in enhancing security. These techniques ensure the authenticity and integrity of the *SLOs* carried in packets, providing an additional layer of protection.

Despite valid security concerns, careful design and the deployment of appropriate

verification and integrity mechanisms can minimize the risks, ensuring that the benefits of embedding QoS objectives in packets outweigh the potential vulnerabilities.

3.6 Conclusion

This study introduces an intelligent and programmable traffic management framework that effectively manages heterogeneous traffic demands using lightweight processing logic. We present an accurate and efficient early-stage traffic classification algorithm based on a single feature integrated with a stateless QoS-aware packet processing mechanism. The proposed framework significantly reduces the number of marked (outdated) packets and prioritizes traffic based on class priority and service objectives, resulting in improved network resource utilization. In addition, the reconfigurable *MATs* allow for the dynamic deployment of ML model outputs and corresponding QoS actions, ensuring seamless adaptation to any changes in data or model generation steps. The trained and optimized ML models within the framework can be safely reused for traffic classification, enabling QoS-driven packet processing within the network.

The current framework validation is limited to a single-bottleneck link scenario. A key question arises: how can a switch effectively decide between a packet with a lower latency budget and a shorter journey ahead versus a packet with a higher latency budget but a longer journey? The framework requires integration of the global network state to enable network-aware optimal forwarding decisions. Moreover, using a controlled simulation environment affects the validity of the insights; it is well-suited for our primary objectives of reproducibility, ease of debugging, and simulating various network conditions. The mentioned limitations provide valuable insights for future development.

REFERENCES

Adams, Richelle (2012). "Active queue management: A survey". In: *IEEE communications surveys & tutorials* 15.3, pp. 1425–1476.

Alcoz, Albert Gran, Alexander Dietmüller, and Laurent Vanbever (2020). "SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues." In: *NSDI*, pp. 59–76.

Black, D and Paul Jones (2015). *Differentiated services (DiffServ) and real-time communication*. Tech. rep.

Bosshart, Pat, Dan Daly, et al. (2014). "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3, pp. 87–95.

Bosshart, Pat, Glen Gibb, et al. (2013). "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN". In: *ACM SIGCOMM Computer Communication Review* 43.4, pp. 99–110.

Busse-Grawitz, Coralie et al. (2019). "pforest: In-network inference with random forests". In: *arXiv preprint arXiv:1909.05680*.

Chakraborty, Biswadeep et al. (2021). "Cost-aware feature selection for IoT device classification". In: *IEEE Internet of Things Journal* 8.14, pp. 11052–11064.

Chen, Wenxiong et al. (2021). “Sequential message characterization for early classification of encrypted internet traffic”. In: *IEEE Transactions on Vehicular Technology* 70.4, pp. 3746–3760.

Chen, Xiang et al. (2022). “Empowering DDoS Attack Mitigation with Programmable Switches”. In: *IEEE Network*.

Clemm, Alexander and Toerless Eckert (2020). “High-precision latency forwarding over packet-programmable networks”. In: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, pp. 1–8.

Clemm, Alexander, Mohamed Faten Zhani, and Raouf Boutaba (2020). “Network management 2030: Operations and control of network 2030 services”. In: *Journal of Network and Systems Management* 28.4, pp. 721–750.

Dang, Huynh Tu et al. (2020). “P4xos: Consensus as a network service”. In: *IEEE/ACM Transactions on Networking* 28.4, pp. 1726–1738.

De Alwis, Chamitha et al. (2021). “Survey on 6G frontiers: Trends, applications, requirements, technologies and future research”. In: *IEEE Open Journal of the Communications Society* 2, pp. 836–886.

Erman, Jeffrey et al. (2007). “Offline/realtime traffic classification using semi-supervised learning”. In: *Performance Evaluation* 64.9-12, pp. 1194–1213.

Finn, Norman et al. (2019). “Deterministic networking architecture”. In: *RFC 8655*.

Froes, Wallas et al. (2020). “ProgLab: Programmable labels for QoS provisioning on software defined networks”. In: *Computer Communications* 161, pp. 99–108.

Gran Alcoz, Albert and Laurent Vanbever (2023). "QVISOR: Virtualizing Packet Scheduling Policies". In: *ACM HotNets*.

Grandini, Margherita, Enrico Bagli, and Giorgio Visani (2020). "Metrics for multi-class classification: an overview". In: *arXiv preprint arXiv:2008.05756*.

Hardegen, Christoph and Sebastian Rieger (2020). "Prediction-based flow routing in programmable networks with P4". In: *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, pp. 1–5.

Ibanez, Stephen et al. (2019). "The p4-> netfpga workflow for line-rate packet processing". In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 1–9.

Intel (2024). *Intel® Tofino™: Intelligent Fabric Processors*. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html. [Online; accessed 04-June-2024].

James, Gareth et al. (2023). "Tree-based methods". In: *An Introduction to Statistical Learning: with Applications in Python*. Springer, pp. 331–366.

Karakus, Murat and Arjan Durresi (2017). "Quality of service (QoS) in software defined networking (SDN): A survey". In: *Journal of Network and Computer Applications* 80, pp. 200–218.

Kfoury, Elie F, Jorge Crichigno, and Elias Bou-Harb (2021). "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends". In: *IEEE Access* 9, pp. 87094–87155.

Khan, Shuraia, Farookh Khadeer Hussain, and Omar K Hussain (2021). "Guaranteeing end-to-end QoS provisioning in SOA based SDN architecture: A survey and Open Issues". In: *Future Generation Computer Systems* 119, pp. 176–187.

Kianpisheh, Somayeh and Tarik Taleb (2022). "A Survey on In-network Computing: Programmable Data Plane And Technology Specific Applications". In: *IEEE Communications Surveys & Tutorials*.

Kunze, Ike et al. (2021). "Tofino+ P4: A strong compound for AQM on high-speed networks?" In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, pp. 72–80.

Lee, Jong-Hyouk and Kamal Singh (2020). "Switchtree: in-network computing and traffic analyses with random forests". In: *Neural Computing and Applications*, pp. 1–12.

Li, Richard et al. (2018). "A new framework and protocol for future networking applications". In: *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, pp. 21–26.

Liatifis, Athanasios et al. (2023). "Advancing sdn from openflow to p4: A survey". In: *ACM Computing Surveys* 55.9, pp. 1–37.

Monge, Antonio Sanchez and Krzysztof Grzegorz Szarkowicz (2015). *MPLS in the SDN Era: Interoperable Scenarios to Make Networks Scale to New Services*. " O'Reilly Media, Inc."

Musumeci, Francesco et al. (2022). "Machine-learning-enabled ddos attacks detection in p4 programmable networks". In: *Journal of Network and Systems Management* 30, pp. 1–27.

Papagianni, Chrysa and Koen De Schepper (2019). "Pi2 for p4: An active queue management scheme for programmable data planes". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments and Technologies*, pp. 84–86.

Performance of bmv2 (n.d.). https://github.com/p4lang/behavioral-model/blob/main/docs/performance.md#what-impacts-performance. Online; accessed 213 February 2022.

Poupart, Pascal et al. (2016). "Online flow size prediction for improved network routing". In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, pp. 1–6.

Qiao, Mai and Deyun Gao (2022). "Fine-Grained Active Queue Management in the Data Plane with P4". In: *2022 7th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, pp. 174–179.

Quan, Wei et al. (2022). "AI-driven Packet Forwarding with Programmable Data Plane: A Survey". In: *IEEE Communications Surveys & Tutorials*.

Ren, Jingjing et al. (2019). "Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach". In: *Proceedings of the Internet Measurement Conference*, pp. 267–279.

Saqib, Muhammad, Halima Elbiaze, and Roch Glitho (2024). "Adaptive In-Network Traffic Classifier: Bridging the Gap for Improved QoS by Minimizing Misclassification". In: *IEEE Open Journal of the Communications Society*.

Saqib, Muhammad, Zakaria Ait Hmitti, et al. (2022). "An Accurate & Efficient Approach for Traffic Classification Inside Programmable Data Plane". In: *GLOBE-*

COM 2022-2022 IEEE Global Communications Conference. IEEE, pp. 6331–6336.

Schrage, Linus E and Louis W Miller (1966). "The queue M/G/1 with the shortest remaining processing time discipline". In: *Operations Research* 14.4, pp. 670–684.

Schulz, Philipp et al. (2017). "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture". In: *IEEE Communications Magazine* 55.2, pp. 70–78.

Sharma, Naveen Kr et al. (2017). "Evaluating the Power of Flexible Packet Processing for Network Resource Allocation." In: *NSDI*, pp. 67–82.

Shenker, Scott, Craig Partridge, and Roch Guerin (1997). *Specification of guaranteed quality of service*. Tech. rep.

Silva, Jeferson Santiago da et al. (2018). "Extern objects in p4: an rohc header compression scheme case study". In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, pp. 517–522.

Sivanathan, Arunan et al. (2018). "Classifying IoT devices in smart environments using network traffic characteristics". In: *IEEE Transactions on Mobile Computing* 18.8, pp. 1745–1759.

Sivaraman, Anirudh et al. (2016). "Programmable packet scheduling at line rate". In: *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 44–57.

Tamura, Hitomi et al. (2004). "Performance analysis for QoS provisioning in MPLS networks". In: *Telecommunication Systems* 25, pp. 209–230.

Tan, Lizhuang et al. (2021). “In-band network telemetry: A survey”. In: *Computer Networks* 186, p. 107763.

Turkovic, Belma et al. (2021). “P4qos: Qos-based packet processing with p4”. In: *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, pp. 216–220.

Wroclawski, John (1997). *Specification of the controlled-load network element service*. Tech. rep.

Wu, Yulei et al. (2022). “A survey of intelligent network slicing management for industrial IoT: integrated approaches for smart transportation, smart energy, and smart factory”. In: *IEEE Communications Surveys & Tutorials* 24.2, pp. 1175–1211.

Xavier, Bruno Missi et al. (2021). “Programmable switches for in-networking classification”. In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, pp. 1–10.

– (2022). “MAP4: A Pragmatic Framework for In-Network Machine Learning Traffic Classification”. In: *IEEE Transactions on Network and Service Management*.

Xie, Guorui, Qing Li, Yutao Dong, et al. (2022). “Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation”. In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, pp. 1938–1947.

Xie, Guorui, Qing Li, Guanglin Duan, et al. (2023). “Empowering In-Network Classification in Programmable Switches by Binary Decision Tree and Knowledge Distillation”. In: *IEEE/ACM Transactions on Networking*.

Xiong, Zhaoqi and Noa Zilberman (2019). "Do switches dream of machine learning? toward in-network classification". In: *Proceedings of the 18th ACM workshop on hot topics in networks*, pp. 25–33.

Zhang, Xiaoquan et al. (2021). "pHeavy: Predicting heavy flows in the programmable data plane". In: *IEEE Transactions on Network and Service Management* 18.4, pp. 4353–4364.

Zheng, Changgang, Xinpeng Hong, et al. (2023). "In-Network Machine Learning Using Programmable Network Devices: A Survey". In: *IEEE Communications Surveys & Tutorials*.

Zheng, Changgang, Zhaoqi Xiong, et al. (2024). "IIsy: Hybrid In-Network Classification Using Programmable Switches". In: *IEEE/ACM Transactions on Networking*.

Zheng, Changgang, Mingyuan Zang, et al. (2024). "Planter: Rapid prototyping of in-network machine learning inference". In: *ACM SIGCOMM Computer Communication Review* 54.1, pp. 2–21.

Zheng, Changgang and Noa Zilberman (2021). "Planter: seeding trees within switches". In: *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, pp. 12–14.

Zhou, Guangmeng et al. (2023). "An efficient design of intelligent network data plane". In: *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 6203–6220.

CHAPTER IV

ARTICLE 2 - ADAPTIVE IN-NETWORK TRAFFIC CLASSIFIER: BRIDGING THE GAP FOR IMPROVED QOS BY MINIMIZING MISCLASSIFICATION

Muhammad Saqib
Université du Québec à Montréal
Montréal (Québec), Canada

Halima Elbiaze, Professeur titulaire
Université du Québec à Montréal
Montréal (Québec), Canada

Roch Glitho, Professeur titulaire
Concordia University
Montréal (Québec), Canada

FOREWORD TO ARTICLE 2

The first paper of this thesis addresses the intelligent handling of heterogeneous traffic demands in the network using an early-stage network traffic classifier. This paper investigates and quantifies the impact of traffic misclassification on QoS and InP profitability. This paper was published in the scientific journal **IEEE Open Journal of the Communications Society (IEEE OJ-COMS)**, Volume 5, Jan 2024.

- M. Saqib, H. Elbiaze and R. H. Glitho, **Adaptive In-Network Traffic Classifier: Bridging the Gap for Improved QoS by Minimizing Misclassification**, in IEEE Open Journal of the Communications Society, vol. 5, pp. 677-689, 2024, doi: 10.1109/OJCOMS.2024.3351706.

A related article to the above article was published in the scientific journal (conference) **IEEE International Conference on Communication (ICC)**, May 2023.

- M. Saqib, H. Elbiaze and R. Glitho, **A Profit-Aware Adaptive Approach for In-Network Traffic Classification**, ICC 2023 - IEEE International Conference on Communications, Rome, Italy, 2023, pp. 3351-3356, doi: 10.1109/ICC45041.2023.10278593.

Papers published in IEEE OJ-COMS and IEEE ICC are peer-reviewed in accordance with the requirements set forth in the IEEE PSPB Operations Manual (Sections 8.2.1.C & 8.2.2.A). Each published paper has been reviewed by a minimum of two independent reviewers using a single-blind peer review process, where the identity of the reviewers is not known to the authors, but the reviewers know the identity of the authors. Papers are subject to plagiarism check before acceptance.

ABSTRACT

In-network traffic classification presents an innovative approach to developing early-stage and accurate traffic classification solutions. However, despite its initial accuracy, the one-size-fits-all Machine Learning (ML) model becomes obsolete as traffic patterns evolve. This evolution in traffic patterns inevitably leads to misclassification, resulting in the erroneous mapping of traffic flows to Quality of Service (QoS) classes. Consequently, misclassification may lead to service quality violations, imposing penalties on Infrastructure Providers (InPs). The impact, however, is not solely tied to misclassification rates, as a multi-path network with paths of varying capacities can redirect traffic from low data rate classes to high data rate paths and vice versa, thereby influencing the overall outcome. Therefore, precisely quantifying the impact of misclassification on network performance, i.e., QoS, is paramount. This research aims to investigate and address the effects of traffic misclassification on Service Level Agreement (SLA) violations within multi-class, multi-path networks. To achieve this, we propose a novel framework to quantify SLA violations caused by misclassification, an economic model to assess its impact on provider profitability, and adaptive ML techniques to enhance traffic classification accuracy continually. The evaluation results reveal that the optimal path allocation for various traffic classes determines the targeted revenue. Meanwhile, the adaptivity of the classifier maintains prediction accuracy, ensuring the integrity of SLA through precise QoS class assignments. Hence, implementing an adaptive traffic classifier can mitigate penalties and sustain profitability. This work provides valuable insights for network operators, enabling effective misclassification management, resource optimization, and the maintenance of SLA integrity.

Keywords : Programmable data plane, in-network traffic classification, traffic misclassification, QoS, network economics.

4.1 Introduction

With the proliferation of latency-sensitive services and applications within the Internet of Things (IoT) domain, the application of diverse Quality of Service (QoS) policies and efficient utilization of network resources becomes paramount. Network Slicing (NS) has emerged as a promising solution for resource orchestration, enabling QoS isolation through the overlay of multiple virtual networks on a shared network domain (Wu et al., 2022). NS facilitates the efficient utilization and management of network resources while offering differentiated services at scale, allowing specific services to leverage dedicated network slices to meet their QoS requirements (Esteves et al., 2020).

In network management, network traffic classification plays a pivotal role. It serves as the foundation for mapping incoming traffic flows to their appropriate QoS slices, thereby ensuring the provision of application-specific QoS guarantees (Yao et al., 2019). This simplifies the enforcement of *Service Level Agreement (SLA)*. Given the dynamic nature of traffic patterns and the growing diversity of IoT behaviors, early-stage traffic classification is indispensable for timely and accurate QoS provisioning. The programmability of the data plane allows for the definition of customized matching criteria for traffic type identification (Kfoury et al., 2021), enabling fine-grain classification at a near-line rate. This concept is known as in-network traffic classification, which involves implementing rule-based Machine Learning (ML) models, such as Decision Trees (DTs), within switches to achieve high-speed processing (Xiong and Zilberman, 2019; Xavier et al., 2021; Xie et al., 2022).

In-network traffic classification turned out to be the key enabler in accurate and early-stage traffic classification solutions (Saqib, Hmitti, et al., 2022). However, in the IoT domain, the traffic presently includes a variety of behaviours such as

communication types, events, sources, patterns and volumes, etc(Sivanathan et al., 2018; Žliobaitė et al., 2016). These behaviours considerably impact traffic patterns, management, and control. Based on the number and type of active devices in the network, the devices' behaviour might generate a variety of characteristics, i.e., variability in data transmission period and payload size. Learning-based models have rapidly become a viable option for identifying the source devices and application types from Internet traffic (Azab et al., 2022). Despite having a learning model with good accuracy, a single-time trained model becomes outdated as the traffic pattern changes over time (Lu et al., 2018). This changing traffic pattern leads to *misclassification*, i.e., incorrect mapping of traffic flows to the QoS classes.

The researchers approached misclassification from a risk-free or cost-aware model training perspective, with minimal inaccuracy risk. For example, the authors in (Chakraborty et al., 2021) argue that a solution for device identification (or classification) should priorly consider features extraction cost (computational and memory). Similarly, another work in (Telikani et al., 2021) considers a cost-sensitive learning strategy to ensure the robustness of traffic classifiers against unbalanced datasets. They consider the misclassification cost during training and minimize the training model's cost. However, despite having a cost-effective model with good classification accuracy, the one-fit ML model loses its relevance over time as the traffic pattern changes. This loss of accuracy leads to incorrect mapping of traffic flows to the QoS classes, which further results in *SLA* violations and affects customer satisfaction in return. As a result, on the one hand, the service provider tries to increase revenue through priority-based traffic scheduling (Y. Xu and D. Xu, 2018). On the other hand, incorrect QoS class mapping by the classifier may lead to *SLA* violations, resulting in the addition of penalties that negatively impact the provider's profit. A significant challenge for Infrastructure Providers (InPs) is

ensuring multi-priority traffic demands while maintaining maximum profit. To the best of the authors' knowledge, no existing works investigate the implications of traffic misclassification on network performance or network operators.

In our prior work (Saqib, Elbiaze, et al., 2023), we examined the effect of traffic misclassification on InP profitability. We developed an economic model that directly calculates penalties based on class priority and misclassification rate. We assumed that *SLA* violations are directly proportional to the misclassification rates of traffic classes. However, this is only sometimes the case in multi-class, multi-path networks. In other words, the misclassification rate is not the sole metric for *SLA* violation, as misclassification does not necessarily result in *SLA* breaches. Instead, there is a need for precise measurement of the QoS status of network paths to calculate *SLA* violations. Hence, there remain unanswered questions. For instance, *what if traffic flows from a high data rate class are incorrectly directed to a low data rate path, or vice versa? What if traffic from multiple classes is poorly mapped to a single network path?* These open questions underscore the importance of measuring the misclassification impact on network performance and quantifying *SLA* violations to establish equitable penalties for the InP.

This research aims to investigate and mitigate the impact of traffic misclassification on *SLA* violations within multi-class, multi-path network environments. Our contributions encompass a novel framework for quantifying *SLA* violations caused by misclassification, an economic model to evaluate its impact on provider profitability, and adaptive ML techniques for continuous improvement of traffic classification accuracy. This work provides valuable guidance to network operators for effectively managing misclassification, optimizing resources, and ensuring the integrity of *SLAs*.

The proposed research investigates and mitigates the impact of traffic misclassifica-

tion through a two-phase system design: the control plane and the data plane. In the control plane, we determine optimal routing paths to maximize revenue and employ adaptive ML techniques to generate updated classification rules for the data plane. The data plane features an in-network traffic classifier and performance monitoring for the quantification of *SLA* violations. By jointly monitoring classifier performance and network path utilization, we precisely measure the impact of misclassification on the QoS of network paths and enable adaptive model updates to improve accuracy, thereby reducing the negative impact of classifier.

The remainder of this paper is organized as follows. Section 4.2 provides an overview of the related work. Section 4.3 presents the problem definition and network model. Section 4.4 introduces the proposed solution. The validation plan is discussed in Section 4.5, followed by the presentation of experimental results in Section 4.6. Finally, Section 4.7 contains the concluding remarks.

4.2 Related work

This section provides an overview of the most relevant approaches in the literature concerning the traffic misclassification problem, exploring how researchers have addressed this crucial issue from various perspectives.

In (Nechay et al., 2009), the authors present two novel types of online Internet traffic classifiers designed to ensure performance guarantees for false alarm and false discovery rates. These classifiers aim to minimize overall misclassification rates while meeting specific constraints, with one classifier focused on reducing false alarm rates and the other on reducing false discovery rates. The proposed techniques enhance network monitoring, service quality, and security measures.

Some researchers have approached misclassification as an optimal feature selection problem, primarily focusing on minimizing inaccuracy risks. For instance,

in (Wanode et al., 2022), the authors select the optimal set of features for IoT device fingerprinting on edge infrastructure to reduce the feature set's size while maintaining classification accuracy. This approach enables efficient device identification on resource-constrained edge nodes. Similarly, another study (Chakraborty et al., 2021) addresses misclassification in an IoT environment by introducing a new variable called *risk* into the classification algorithm. They emphasize the importance of identifying whether misclassification occurred and determining the misclassified class, as this information can have significant implications for actions and costs. Incorporating the notion of risk aims to improve the accuracy and effectiveness of IoT device classification.

Another perspective on misclassification treats it as an imbalanced class problem. The authors tackle misclassification in (Telikani et al., 2021) by proposing a cost-sensitive deep learning approach. They divide the dataset into partitions and create a cost matrix for each partition based on the data distribution. These costs are applied to the cost function layer to penalize classification errors, ensuring diverse costs for each type of misclassification. By incorporating these costs into the deep learning classifiers, the proposed approach aims to enhance the robustness of classifiers against the imbalanced class problem in network traffic classification.

Similarly, a different approach is presented in (Zhu et al., 2023), where a deep learning model named the Cost Matrix Time-Space Neural Network (CMTSNN) is introduced. The CMTSNN model utilizes a cost penalty matrix and an improved cross-entropy loss function to enhance the classification accuracy of minority categories and overall multi-classification performance. The cost penalty matrix is applied to the cost penalty layer, and the improved cross-entropy loss function is used to calculate the loss, reducing the impact of data imbalance on model classification. This approach helps mitigate the effects of misclassification and improves the identification of encrypted abnormal traffic in the IoT network.

It is worth noting that the term *misclassification* is not new but has been explored in the state-of-the-art literature from risk-free or cost-aware perspectives. Researchers consider the misclassification risk or cost during the training process, focusing on minimizing the risk of inaccuracy or the cost of the model training. However, even with a risk-free or cost-effective model with good classification accuracy, a one-size-fits-all ML model can lose relevance over time as traffic patterns change due to *concept* or *data drift*(Lu et al., 2018). This loss of accuracy leads to the incorrect mapping of traffic flows to QoS classes, resulting in *SLA* violations and reduced customer satisfaction. Considering the impact of post-classification processes becomes highly desirable in the context of multi-class, multi-path networks. Surprisingly, the reviewed work has not investigated the implications of incorrectly mapped traffic flows on network performance or InP.

4.3 Problem formulation and network model

4.3.1 Toward traffic characterization

We begin by discussing the motivation for categorizing Internet traffic. Instead of relying on a well-known QoS classifier identifier (QCI) table, which usually offers a predetermined set of values linked to specific QoS characteristics, we aim to address the diverse and evolving nature of emerging applications and services. These entities often have distinct data rates, latency, and priority needs, and the rigid structure of QCI tables may need to be revised to accommodate such variability effectively. Our approach seeks a more flexible and adaptive method for classifying traffic flows to better align with the dynamic requirements of contemporary applications. For example, the chosen applications in Table 4.1 exhibit varying sensitivity to service quality requirements. We translated these service quality metrics, such as latency φ and data rate γ, to various classes having different priorities. Each class is

characterized by specific traffic attributes, including high definition HD, real-time R_t or non-real-time traffic X_t - indicating the bandwidth and latency requirements; critical R_a or non-critical data rate X_a - to express the delay tolerance level(Afzal et al., 2017).

Table 4.1 Service quality requirements of a few emerging applications (De Alwis et al., 2021)

Use case / Applications	**Traffic Class**	**Data rate** (γ)	**E2E Latency** (φ)	**Priority** (α)
Personalized BAN	Non-real-time/Critical rate (X_tR_a)	1-100 Mbps	$\leq$ 1000 ms	α_1
Internet of Everything	Non-real-time/Non-critical rate (X_tX_a)	100-1000 Mbps	$\leq$ 100 ms	α_2
Smart Grid 2.0	Real-time/Critical rate (R_tR_a)	100-2000 Mbps	$\leq$ 10 ms	α_3
Live streaming	HD real-time/Highly critical rate (HR_tHR_a)	2000-4000 Mbps	$\leq$ 1 ms	α_4

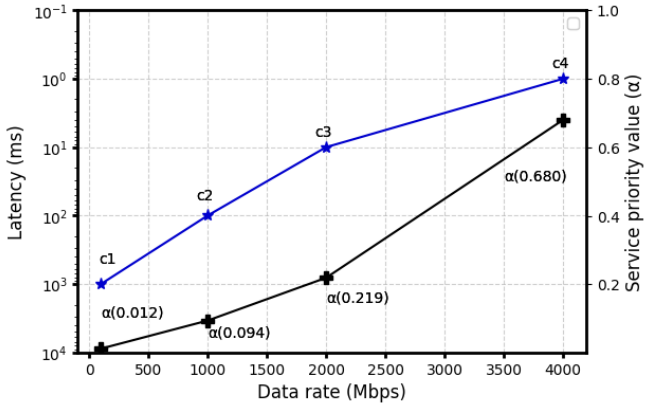


Figure 4.1 Service priority factor calculation

In Fig. 4.1, we illustrate the given classes' latency and data rate requirements to establish their priorities. The requirements become more critical as the data rate increases while latency decreases. The higher the data rate and the lower the latency, the more critical these requirements become. Consequently, the priority of each class is determined by its specific service quality requirements. A service priority factor, denoted as α, is assigned to each class based on these requirements. The α value increases as the required data rate and latency become more stringent. Hence, this α is intricately linked to the criticality level, proportionally related

to the necessary data rate while inversely associated with latency. The blue line visually emphasizes the expansion of the criticality level for each class, while the black line represents the α value. Given that α reflects the relative importance of the classes, it is utilized to define penalty and revenue generation functions in the proposed method. By delineating unique traffic classes and corresponding parameters for each application, our solution aims to optimize resource allocation based on the specific requirements of these diverse use cases.

Table 4.2 Notations

Notation	Description
N	Set of nodes
L	Set of links
$\mathcal{S}$	Set of network slices
K	Set of paths from source and destinations
C	Set of priority class (i.e., QoS group)
$F^{(c)}$	Set of traffic flows belonging to class c
$f_i^{(c)}$	Traffic flow i from class c
$\tau^{(c)}$	Generated traffic load by class c
$\varphi_{p,i}^{(c)}$	Delay of packet p belonging to $f_i^{(c)}$
$\gamma_{l,i}^{(c)}$	Data rate of link l on flow i belonging to class c
b_l, b_k	Bandwidth of link l and path k
$x_k^{(s)}$	Traffic volume of slice s on path k
$\alpha_i^{(c)}$	Service priority of flow i belonging to class c
$\lambda_k^{(c)}$	Traffic portion from class c to path k
$\mu_k^{(c)}$	BW utilization of path k by class c traffic
δ	The selling price for a unit of bandwidth
β	Monetary penalty unit.

4.3.2 Problem definition

Let $c \in C$ be the set of priority classes, each with different service quality parameters, namely the latency φ and data rate γ. Here, the latency φ is defined in terms of end-to-end packet delay on the chosen route, and the data rate γ is given in bits per second (bps) as specified by the user capacity requirement. As a result, each class c is bounded by threshold values of latency $\varphi_{max}^{(c)}$ and data rate $\gamma_{min}^{(c)}$. All the defined variables are summarized in Table 4.2.

Substrate network and constraints: We represent the underlying physical infrastructure as a directed graph $G = (N, L)$ with a set of nodes $n \in N$ and links $l \in L$, each link with a bandwidth $b_l > 0$ (measured in bps). On top of the substrate network, we consider the co-existence of multiple network slices indexed by $s \in \mathcal{S} = \{1, ..., S\}$. For clarity, we consider a scenario where each slice s uniquely serves a single traffic class c, defined by a source-destination pair (u, v) and associated service parameters φ (latency) and γ (data rate) (Leconte et al., 2018). Let $x_k^{(c)}$ denote the bandwidth allocated to class c on path $k \in K$, where K is the set of all possible paths. The following constraints must be satisfied for the traffic flows between any such pair:

$$\varphi_{p,i}^{(c)} \leq \varphi_{\max}^{(c)}, \quad \forall p \in f_i^{(c)} \in F^{(c)} \tag{4.1}$$

$$\gamma_{l,i}^{(c)} \geq \gamma_{\min}^{(c)}, \quad \forall l \in L, \quad \forall f_i^{(c)} \in F^{(c)} \tag{4.2}$$

The above constraint (1) ensures an end-to-end delay threshold along the multi-hop route. The associated end-to-end delay φ with a packet p across the multi-hop path between source node u and destination node v, in particular, shall not exceed the maximum latency limit for a given class c, i.e., $\varphi_{max}^{(c)}$. Furthermore, constraint (2) assures that the data rate γ at any link $l \in L$ should be sufficient to meet the

capacity demand of passing flow i belongs to class c.

In addition, the paths between (u, v) pairs over links L are indexed by $k \in K = \{1., , , K\}$. We denote the traffic volume each class c generates as $x_k^{(c)}$ going through path k. Since multiple slices—each dedicated to a specific class of traffic—may share the same physical network, the total bandwidth consumed on each path k must not exceed the available bandwidth b_k. Thus, we also define the paths' bandwidth constraint:

$$\sum_{c \in \mathcal{C}} x_k^{(c)} \leq b_k, \quad \forall k \in K \tag{4.3}$$

Provider's revenue: The service provider generates revenue by optimal resource (i.e., bandwidth) allocation to heterogeneous traffic demands. The revenue mainly depends on the resources requested by the service. That is the product of the selling price of a bandwidth unit and a class's service priority factor. The pricing policy determines the charge per unit bandwidth for each substrate link $l \in L$. A differential pricing policy is considered based on the class's criticality level. Thus, the revenue gained by the provider at the time t by selling the bandwidth resource can be expressed as:

$$\sigma(t) = \sum_{c \in C} \sum_{k \in K} \delta * \alpha^{(c)} * x_k^{(c)}(t) \tag{4.4}$$

The selling price for a bandwidth unit is δ. The priority factor of class c (i.e., service quality priority) is represented as $\alpha^{(c)}$, and $x_k^{(s)}(t)$ stands for satisfying bandwidth over a path k between (u, v) pairs of classes $c \in C$.

Network traffic classifier: The InP increases revenue by allocating resources to various traffic classes. Meanwhile, a traffic classifier at the network's edge assigns the incoming traffic flows to the correct traffic class, i.e., QoS slice. In the case of

incorrect traffic class mapping, the InP could not implement the appropriate QoS policies, which would impact the Customer Satisfaction Level (CSL). In order to maintain good classification accuracy, it is crucial to regularly check the classifier's prediction and penalize the classifier for inaccurate mappings.

Let $\mathcal{C}$ be the classifier that classifies the incoming traffic flows f_i to the corresponding class. Any misclassification, i.e., incorrect QoS slice mapping, might result in an *SLA* violation, adding a penalty that can be calculated as the per-class criticality level.

Misclassification rate: We use *f1-score* as a performance metric to measure the per-class misclassification rate. *f1-score* assesses the classification model's performance starting from the confusion matrix; it aggregates *Precision* and *Recall* measures under the concept of harmonic mean (Grandini et al., 2020). The formula of *f1-score* can be interpreted as a weighted average between *Precision* and *Recall*:

$$f1 = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \tag{4.5}$$

where

$$Precision = \frac{TP}{(TP + FP)} \tag{4.6}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{4.7}$$

TP means the observation is positive, and the sample is predicted to be positive. FN is that the observation is positive, but the sample is predicted to be negative. TN describes that observation is negative, and the sample is predicted to be negative. And FP represents that the observation is negative, but the sample is predicted to be positive.

The *f1-score* reaches its best value at one and the worst score at 0. Hence, the misclassification rate of a particular class c can be defined as:

$$\lambda^{(c)} = 1 - f1^{(c)}, \ \forall\, c \in C \tag{4.8}$$

Penalty: The provider must return the penalty incurred due to the misclassification. The penalty ρ for a particular class c over time t can be calculated as a product of the monetary penalty unit, the class's priority, the misclassification rate, and the path utilization rate of the class. The total penalty ρ associated with each class c at the time t can be computed as follows:

$$\rho(t) = \sum_{c \in C} \sum_{k \in K} \beta \cdot \alpha^{(c)} \cdot \lambda_k^{(c)} \cdot \mu_k(t) \tag{4.9}$$

where β is the monetary penalty unit, $\alpha^{(c)}$ is the class priority, $\lambda^{(c)}$ is the misclassification rate of class c over path k and μ_k is the bandwidth utilization rate of path k.

Objective function: The objectives are to maximize the provider's profit and minimize *SLA* violations. Maximizing the provider's profit can be achieved by maximizing σ and minimizing ρ. The objective function $\mathcal{P}$ can therefore be written as follows:

$$\max_{\mathcal{P}} \sum_{t \in T} (\sigma(t) - \rho(t)) \tag{4.10}$$

Subject to constraint **(4.1)**, **(4.2)** and **(4.3)**.

Where $\mathcal{P} = \{x_k^{(c)}, \lambda^{(c)}\}$ is the set of decision variables, with $x_k^{(s)}$ denoting the bandwidth allocated to class c over path k, and $\lambda^{(c)}$ representing the *f1-score* of traffic class c.

4.4 Proposed solution

In this section, we introduce a framework designed to investigate the impact of traffic misclassification and an adaptive learning-based approach to mitigate this impact. Fig. 4.2 shows the high-level system design, encompassing two primary phases: (i) optimizing routing paths and offline ML model training within the control plane and (ii) identifying traffic classes for network slice allocation and performance monitoring within the data plane. In the following subsections, we explore the details of our integrated and smart design for addressing the traffic misclassification problem.

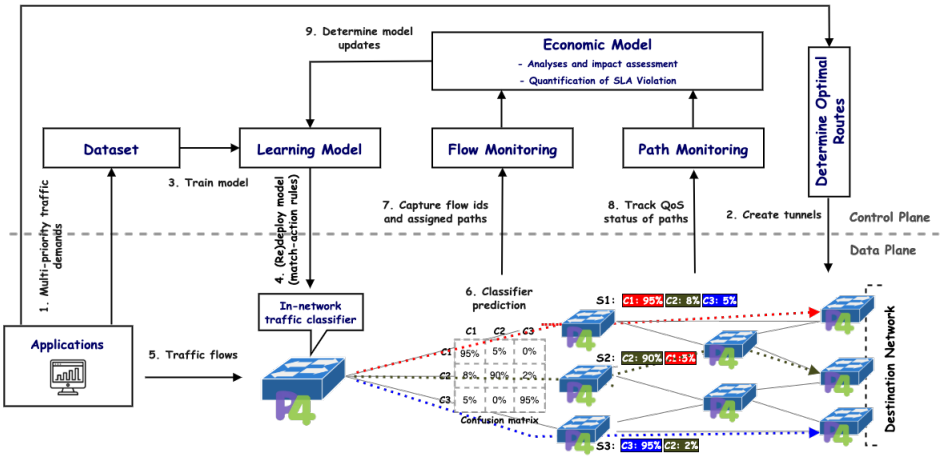


Figure 4.2 System design

4.4.1 Control plane

The control plane comprises three essential logical components: route optimization, ML model training, and an economic model for quantifying *SLA* violations and guiding updates to the ML model.

4.4.1.1 Determine optimal routes

The control component first utilizes a routing module to identify the most efficient paths for accommodating heterogeneous traffic demands. Subsequently, it sets the targeted revenue, determined by the revenue generation function, as defined in Eq. (4.4). The function considers traffic demands and available routes with bandwidth capacity as inputs, resulting in the calculation of the maximum achievable revenue. To solve the allocation problem for traffic demands across multiple classes and paths in the network, we formulate this optimization problem as an Integer Linear Program (ILP). Our ILP can be generalized as a 0/1 Multi-Knapsack Problem (MKP) (Zhang and Geng, 1986). The complete formulation is detailed below:

$$\max_{\sigma} \sum_{c \in C} \sum_{k \in K} \delta \cdot \alpha^{(c)} \cdot x_k^{(c)} \tag{4.11}$$

subject to

$$\sum_{k \in K} x_k^{(c)} \leq 1 \quad \forall c \tag{4.12}$$

$$\sum_{c \in C} \gamma^{(c)} \cdot x_k^{(c)} \leq b_k \quad \forall k \tag{4.13}$$

$$\sum_{c \in C} \varphi_k \cdot x_k^{(c)} \leq \varphi_{max}^{(c)} \quad \forall k \tag{4.14}$$

$$x_k^{(c)} \in \{0, 1\} \quad \forall c, k \tag{4.15}$$

where

$$x_k^{(c)} = \begin{cases} 1 & \text{if class } c \text{ allocated to path } k \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

(4.12) guarantees that traffic demand from a specific class c is accommodated by being assigned to at most one path k. Constraint **(4.13)** ensures that the allocation adheres to the capacity of each designated path, while constraint **(4.14)** ensures that the cumulative latency on the allocated path does not surpass the predetermined limit for each traffic class. Finally, the binary decision variables represent the pivotal choices in allocating traffic demands to the network paths. We utilized the Gurobi optimizer to solve the ILP (see Section 4.5.2).

4.4.1.2 Learning module

Additionally, the control plane employs a learning module to gain insights into multi-priority traffic patterns from a given dataset. The dataset, denoted as S, comprises packets p_j from sub-flows ($f_i(1:j)$) and is divided into two subsets: a training set, S_T, and a testing set, S_P. The learning module initially trains the ML model on S_T. Subsequently, it translates the resulting output, represented as *if-else* conditions, into *Match-Action Tables (MATs)* within a P4-enabled programmable switch for real-time inference. This translation process is facilitated through a control plane API called P4Runtime.

Numerous supervised learning approaches are available in the literature for traffic characterization, but not all are suitable for implementation in P4 (Xavier et al., 2021). Our objective is to seamlessly integrate the ML model's output into the data plane, necessitating compatibility with the available operations in P4. Consequently, we opted for a classical decision tree algorithm to circumvent the limitations of P4. Given the current primitives in the P4 language (Bosshart et al., 2014), a *Decision Tree Classifier (DTC)* proves to be the more suitable choice for this task. It requires a comparison operation to classify an element x, which can be readily expressed in P4 using *match-action* rules.

4.4.2 Data plane

The data plane comprises an in-network traffic classifier and performance monitoring logic responsible for detecting *SLA* violations.

4.4.2.1 In-network traffic classifier

After embedding the learning-based model's output into the data plane, the next step is predicting the incoming packet class. Our in-network traffic classifier is designed using a novel and effective method capable of quickly and accurately classifying diverse traffic classes. What makes our approach unique is its reliance on a single stable feature, specifically the sequential packet size information that can be directly extracted from the packet's header.

The switch maintains registers to track flow IDs, packet sizes, and packet counters for the initial packets of each flow. For incoming packets, the switch's parser extracts the flow ID and relevant features, such as the five-tuples (i.e., IPs, Ports and protocol) and packet size, from the header and stores these feature values in the pipeline's metadata. The flow ID register records all classified flows, allowing the switch to handle packets belonging to these classes efficiently. This means that packets from the classified flows do not need to undergo the decision tree process and are processed at a line rate, according to the identified class or QoS policy.

When a flow is not classified, the switch checks the packet counter for that flow. The parser extracts packet sizes and stores them in a size vector until the packet counter reaches a certain threshold. Once the packet counter reaches the threshold, the *MATs* are used for classification based on the size vector. Subsequently, the switch directs subsequent flow packets to the appropriate slice (i.e., QoS group). We refer to (Saqib, Hmitti, et al., 2022) for more details about the traffic flow

classification process inside a programmable data plane.

4.4.2.2 Performance monitoring

The next step involves calculating the misclassification rate for each class based on the classifier's predictions. We continuously monitor the classifier's performance by computing the *f1-score* from the data plane's predictions. Due to evolving traffic patterns, the deployed model's accuracy diminishes over time, resulting in flows being incorrectly mapped to the wrong QoS classes.

In our design, traffic flows from a set of classes, denoted as C, are mapped to a corresponding set of slices, denoted as S. As illustrated in Fig. 2, the colored arrows, such as *red, green,* and *blue,* represent distinct slices with varying bandwidth capacities and propagation delays. For instance, misclassification causes approximately 8% of flows from class $c2$ and 5% of flows from class $c3$ to be routed through slice $s1$. Similarly, about 5% of flows from class $c1$ and 2% of flows from class $c2$ are routed through $s2$ and $s3$, respectively.

Due to these misclassifications, the intended QoS policies may not be applied to a fraction of the flows, resulting in degrading service quality. The switch captures the flow IDs and associated routing paths to address this issue. Based on the generated traffic load $T^{(c)}$ for each class and the classifier's predictions, we calculate the traffic load on each network path k and determine the bandwidth utilization rate u_k for each path.

Since the available bandwidth capacity varies across network paths, some misclassified traffic flows from a particular class may be directed to paths with either higher or lower capacity. This can lead to underutilization or overutilization of certain paths, affecting service quality. Measuring the precise impact on service quality, such as how each traffic class receives its allocated resources, is a complex task

in a multi-class and multi-path network. Therefore, we employ a joint approach, periodically monitoring the classifier's predictions and the QoS status of the paths to assess the impact of traffic misclassification on service quality.

4.4.3 Integrated and adaptive design

The entire process for assessing and mitigating the impact of traffic misclassification is outlined in Algorithms 2-4. Algorithm 2 iterates T times, invoking path scheduling and penalty calculation algorithms at lines 7 and 8, respectively. Algorithm 3 aims to maximize revenue through optimal path allocation, while Algorithm 4 calculates the misclassification rate for each traffic class, the utilization rate of each network path, and the associated penalties. The InP profit is periodically determined by deducting the estimated penalties from the generated revenue at line 9.

Algorithm 2: Maximizing InP's profit

Input: C, F, T, K

C: A list of classes with varying QoS requirements

$F^{(c)}$*: Network traffic flows of class c*

$T^{(c)}$*: Generated traffic load for each class c*

K: A list of capacity-varying routes

Output: $\mathcal{P}$*: Profit*

$\mathcal{P} = 0$;

for $t \in T$ do

    $\sigma(t)$ = *PathScheduling(C, K)* ; // Algorithm 3

    $\rho(t)$ = *PenaltyCalculation(C,F,T,K)* ; // Algorithm 4

    $\mathcal{P}(t) = \sigma(t)$ - $\rho(t)$; // Profit over time t

Algorithm 3 utilizes a solver to iterate over traffic classes and network paths. For each item (i.e., traffic demand from a class c), we have K choices (paths) for allocation. Therefore, we must allocate C items to K choices over each period t. For each feasible allocation, where a network path can adequately meet the service

quality requirements of a traffic class, the revenue is calculated by multiplying the class's priority factor with the price for a bandwidth unit and the satisfiable bandwidth. The generated revenue is temporarily assigned to a variable σ at line 5. The solver aims to maximize the revenue through optimal allocation and returns the maximum generated revenue as a result of the optimal assignment of C to K. The algorithm's computational complexity is inherently tied to the performance characteristics and algorithms implemented within the solver. However, in the worst case, the algorithm may need to explore all possible combinations of C and K. Hence, the worst-case complexity can be approximated as $\mathcal{O}(K^C)$.

Algorithm 3: PathScheduling

Input: C, K

Output: $max\_revenue$

$max\_revenue = 0$;

for $c \in C$ do

  for $k \in K$ do

    for *each feasible solution* do

      $\sigma = \sum_{c \in C} \sum_{k \in K} \delta \cdot \alpha^{(c)} \cdot x_k^{(c)}$;

      if $\sigma > max\_revenue$ then

        $max\_revenue = \sigma$;

return $max\_revenue$

Algorithm 4 is designed to monitor network performance, specifically path utilization rates, and precisely calculate the penalties for InP as a result of *SLA* violations. The first nested loop from lines 1 to 8 runs for $K \times C$, calculating the network path utilization rates. Initially, the classifier's predictions, denoted as $\lambda_k^{(c)}$, are derived from classification results where columns represent paths, and rows represent classes. The network path utilization rate, i.e., μ_k at line 4, is then determined by multiplying the generated traffic load $T^{(c)}$ for a specific class c by the classifier's prediction $\lambda_k^{(c)}$ and dividing it by the network path's capacity b_k. This utilization rate reflects how efficiently each path is being used. Furthermore, to ensure fairness

Algorithm 4: PenaltyCalculation

Input: C, F, K, T

Output: ρ

```
// Calculate path utilization
```

for $k \in K$ do

  for $c \in C$ do

    $\lambda_k^{(c)} = pr(F_k^{(c)})$;

    $\mu_k += \left(\frac{T^{(c)} \cdot \lambda_k^{(c)}}{b_k} \right) \cdot 100$;

  if $\mu_k > 100$ then

    $\mu_k = \frac{\mu_k}{100}$;

```
// Calculate penalty
```

for $c \in C$ do

  for $k \in K$ do

    if $c \neq k$ then

      $\rho^{(c)} += \beta \cdot \alpha^{(c)} \cdot \mu_k \cdot \lambda_k^{(c)}$;

$\rho = \sum_{c \in C} \rho^{(c)}$;

return ρ

and uniformity in the penalty calculation, we scale the utilization rate for each path by dividing the over-utilization by 100 at line 7, allowing for a consistent assessment.

The second nested loop from lines 9 to 14 runs for $C \times K$, calculating penalties for each traffic class based on traffic misclassification and path utilization rates. The sub-loop iterates over network paths, assessing penalties for traffic incorrectly mapped to paths other than its intended route. Line 11 indicates that if the traffic is directed to a path other than its planned route, then calculate the penalty for that portion of traffic on the assigned path. The penalty is computed as the product of a penalty unit, class priority, path utilization rate, and misclassification

rate at line 12. At the end of the iterations, the algorithm calculates and returns the accumulated penalty for each time (i.e., days).

The combined performance monitoring of the classifier and network paths enables us to measure and quantify the misclassification effects accurately. Based on this quantification, the economic model determines the appropriate updates to adapt to newly obtained traffic patterns and improve the classifier's performance. Consequently, we incrementally regularly incorporate freshly acquired data into the existing model to keep an up-to-date ML model within the data plane. This approach ensures that flows are predicted correctly and misclassification rates are minimized. The complete computational complexity of the proposed algorithms can be approximated as:

$$\mathcal{O}\left(a_1(T) \times \left(a_2(K^C) + a_3(K \times C + C \times K)\right)\right)$$

- $a_1(T)$: Represents the computational complexity of running the solver for periods.
- $a_2(K^C)$: Signifies the solver's complexity, linked to its internal logic. In the worst case, the algorithm explores all possible combinations, resulting in an approximate worst-case complexity of $\mathcal{O}(K^C)$.
- $a_3(K \times C + C \times K)$: Represents the complexity of calculating the path utilization rates and penalty for each class on the associated path.

The overall worst-case complexity depends on the number of traffic classes C and network paths K.

4.5 Validation methodology

This section presents the dataset employed, describes the experimental setup, and outlines the distinct performance analysis cases covered in Sections 4.5.1, 4.5.2, and 4.5.3, respectively.

4.5.1 Dataset description

We have utilized packet capture (PCAP) traces of IoT devices from (Sivanathan et al., 2018) as our dataset. Among the available instances in the dataset, we consider the PCAP files spanning seven days, from September 23 to September 29, 2016. These files contain flows associated with four distinct applications and involve various IoT devices.

Table 4.3 Dataset summary

Class	Device type	# of flows	# of packets
C1(XR_a)	Sensor	6411	233329
C2(XX)	Appliance	5439	23561
C3(R_tR_a)	Controller	16788	270840
C4(HR_tHR_a)	Camera	1601	144187

These IoT devices have been categorized into different classes, each with varying degrees of priority. To ensure diversity, we selected devices capable of being assigned to various QoS groups, ranging from high bandwidth and low latency to best effort. Devices within the same class exhibit similar traffic characteristics. Consequently, for validation, we select a single device from each class. An overview of the dataset about these chosen devices is shown in Table 4.3.

4.5.2 Experimental setup

We organized our experiments into three steps. First, we define the network topology for finding optimal paths using a solver in Section 4.5.2.1. Then, we discuss the model training and deployment in Section 4.5.2.2. Finally, we present the simulation setup for the in-network traffic classifier and performance monitoring in Section 4.5.2.3.

4.5.2.1 Network topology

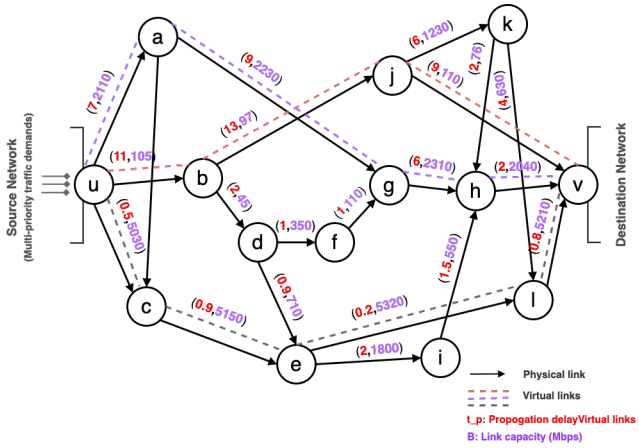


Figure 4.3 Network topology

Since we consider the performance requirements for various classes of traffic defined in Table 4.1, the InP generates revenue by satisfying traffic demands with varying service quality parameters. We define a network topology with diverse paths (see Fig. 4.3) and employ Gurobi solver to identify the optimal paths for fulfilling the traffic demands.

4.5.2.2 Model training and deployment

We relied on Python's *scikit-learn*[1] implementation for model training, and we used cross-validation to choose the best hyperparameters (e.g., tree height and the minimum number of items per leaf) to avoid overfitting. The *DTC* classifier produces output translated into *MATs* in the form of *match-action* rules using P4Runtime API. A single table handles a single packet's size range from the first few packets' sequences of an individual flow. The number of tables equals the input length plus a class prediction table. Before applying *MATs*, the P4 code extracts the payload sizes of the sequential packets from an individual flow and stores them as a size vector in *SRAM*. Once the input length's threshold is reached, the switch traverses the size vector through the *MATs* to predict the traffic class.

4.5.2.3 Simulating data plane

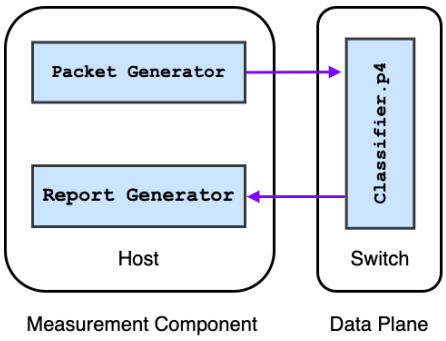


Figure 4.4 Simulation setup

The logical components of the simulation setup are shown in Fig. 4.4. The Measurement Component (on the left) is responsible for generating, collecting, and analyzing the network traffic, whilst the Data Plane Component (on the right) is

[1]https://scikit-learn.org/

the target to be evaluated. The data plane is implemented in P4, compiled with a target of behavioural model version 2 (BMv2). The structure of the simulation setup is organized as follows:

- *Host* send the packets with Pktgen-DPDK[2] and also receives the packets back already classified and timestamped. This component reports the classifier's predictions and network performance for the packets.
- *Switch* contains the traffic classification algorithm responsible for classifying and mapping incoming traffic to the appropriate QoS slices.

4.5.3 Cases for performance analyses

Traffic misclassification doesn't solely equate to *SLA* violations. In other words, misclassification doesn't necessarily lead to *SLA* breaches. For instance, traffic from a low-priority and low-data-rate class can be erroneously mapped to a high-priority and high-data-rate path without impacting that class significantly. Conversely, misclassifying traffic from a high-priority, high-rate class to a low-priority, low-data-rate path can severely affect *SLA* compliance. Therefore, *SLA* violations for a specific class may or may not be linked to misclassification.

In a multi-class, multi-path network, numerous possibilities exist, necessitating precise measurement of *SLA* violations for equitable calculation of per-class penalties. We've defined three validation and performance analysis cases to simplify our approach, facilitating a foundational understanding of *SLA* violation quantification.

- *Case 1:* When traffic from a low-priority class is incorrectly mapped to multiple high-priority paths;

[2]https://pktgen-dpdk.readthedocs.io/en/latest/

- *Case 2:* When traffic from a high-priority class is incorrectly mapped to multiple low-priority paths;
- *Case 3:* When traffic from multi-priority classes is incorrectly mapped to a single path.

4.6 Experimental results

The experimental results are divided into three main parts: firstly, determining optimal paths to achieve the targeted revenue by satisfying diverse traffic demands; secondly, assessing the impact of traffic misclassification; and thirdly, evaluating the adaptivity of the machine ML to mitigate the effects of misclassification. These results are presented in Sections 4.6.1, 4.6.2, and 4.6.3.

4.6.1 Satisfying multi-priority traffic demands

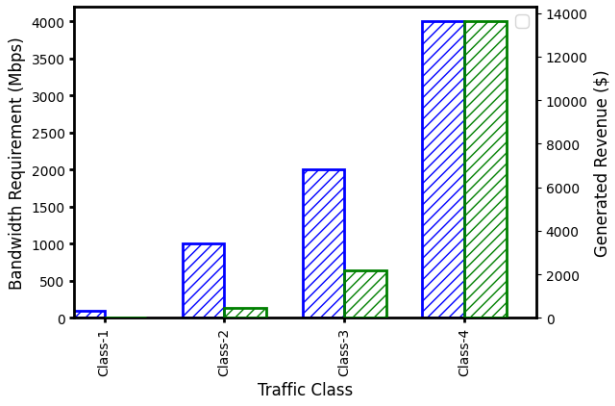


Figure 4.5 Per-class revenue determined by optimal path allocation

The InP generates revenue by meeting traffic demands while considering distinct service quality parameters. The selling price for a single unit of bandwidth is set at 5$. The provider allocates the optimal routing paths the solver determines to meet

diverse traffic demands. Based on these satisfiable traffic demands, the provider calculates the targeted revenue using the revenue generation function defined in (Eq. 4.4). The generated revenue, as depicted in Fig. 4.5, is shown for the specified classes from Table 4.1. The bar graph illustrates the correlation between bandwidth demand and revenue. This correlation is particularly robust for classes with a high criticality level, such as *class-4*, while *Class-2* exhibits the opposite behavior due to its lower criticality level. Consequently, the InP identifies the optimal route for each traffic class to maximize revenue while meeting the demands.

4.6.2 Impact of traffic misclassification

The next step involves assessing the impact of misclassification on the provider's profit. Since network traffic patterns evolve rather than remain constant, an ML model's accuracy naturally degrades over time as traffic patterns change. We employ three distinct cases to measure the impact of misclassification and quantify *SLA* violations (see Section 4.5.3).

We divided the chosen dataset into multiple chunks for analysis, each representing daily streaming data. The initial data chunk was dedicated to training, while subsequent fragments were used for testing. We examined new data patterns on the *2nd, 3rd, and 4th day* to identify the ML model drift, validating the cases mentioned earlier. We then periodically monitored the model's performance on contemporary data patterns.

4.6.2.1 Classification results

The following day's confusion matrix for each case can be seen in Fig. 4.6. In the first case, a substantial portion of traffic from a low-priority class (i.e., class 1) is erroneously mapped to high-priority classes. In contrast, the second case involves

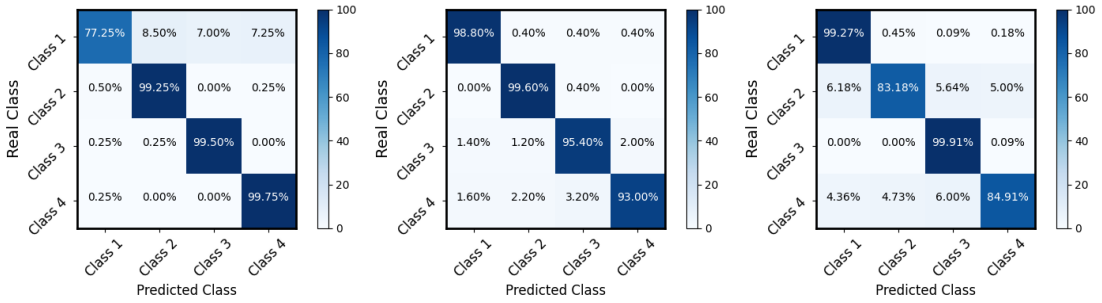


Figure 4.6 Confusion matrix for case 1 (left), 2 (middle), and 3 (right).

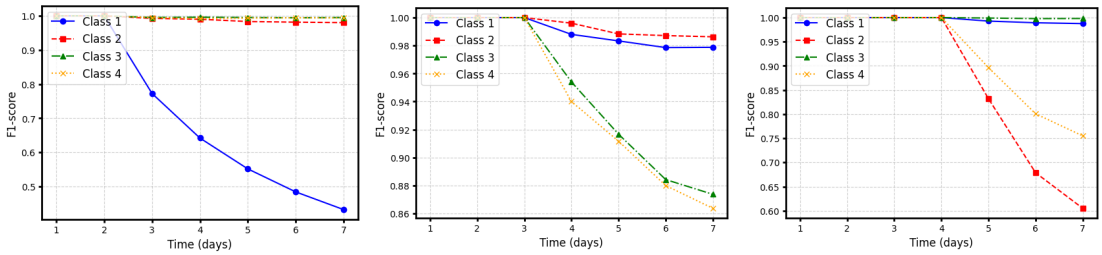


Figure 4.7 Classification result (i.e., f1-score) from complete simulation for cases 1 (left), 2 (middle), and 3 (right).

a minor portion of traffic from a higher-priority class being inaccurately assigned to low-priority classes.

The third case illustrates a scenario where there is no misclassification within specific classes (i.e., classes 1 and 3), but traffic from other classes is directed towards their paths due to misclassification. Fig. 4.7 draws the *f1-score* for the complete simulation encompassing all these cases. Given that incoming traffic patterns are unknown to the existing model, the *f1-score* of the classifier's predictions diminishes in the subsequent days. This underscores the necessity for model adaptivity to address the evolving traffic patterns and tackle the misclassification problem.

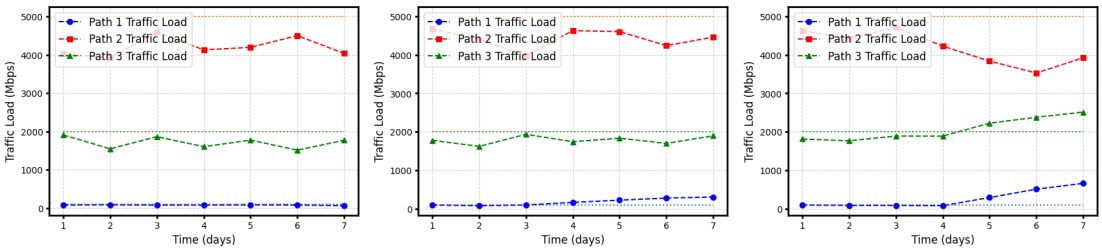


Figure 4.8 Network path capacities and traffic load for cases 1 (left), 2 (middle), and 3 (right).

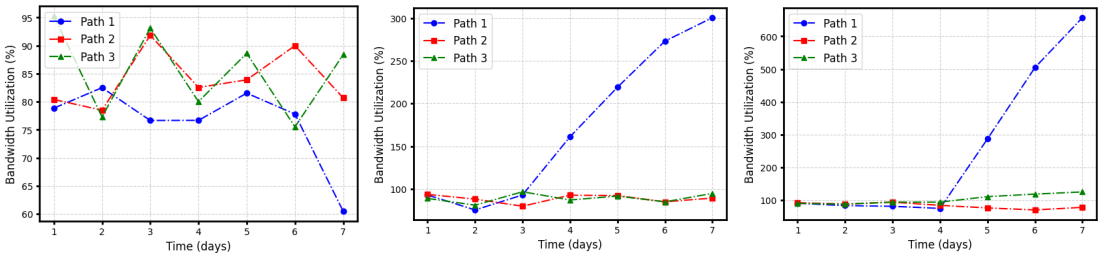


Figure 4.9 Network path bandwidth utilization rates for cases 1 (left), 2 (middle), and 3 (right).

4.6.2.2 Impact on network performance

Fig. 4.8 illustrates the network path capacities for each traffic class. Dotted lines without markers represent the path's capacity (i.e., available bandwidth), while dotted lines with markers indicate the network traffic load on each path. Fig. 4.9 presents the path utilization rates from traffic mapping to the predicted network paths.

In the first case, despite a higher misclassification rate from class 1, the traffic load on all network paths remains below their capacities. Consequently, the influence of misclassification in case 1 on network performance is almost negligible, as depicted

by the network traffic load on each corresponding path in Fig. 4.9. This is attributed to the incorrect mapping of traffic from low to high data rate classes, which imposes minimal overhead on high-capacity network paths and can be accommodated by the network.

Conversely, a minor portion of traffic flows (approximately 1.6%) from a high-priority class is mistakenly mapped to the low-priority class in case 2, resulting in a significant increase in the utilization rate of the corresponding path (i.e., path 1).

The final case visualizes the impact of traffic misclassification from another perspective. In this scenario, there may be no misclassification within specific classes (e.g., class 1 or 3). Still, traffic from other classes may be erroneously mapped due to misclassification from other classes (e.g., 2 and 4). This can lead to over-utilization of the associated paths.

The three presented cases demonstrate the impact of traffic misclassification on network performance, which results from the incorrect mapping of traffic to designated network paths. It's important to note that this impact isn't solely determined by the rate of traffic misclassification but also hinges on the specific QoS requirements of the traffic classes and the QoS status of the associated network paths. Therefore, in addition to considering factors such as the penalty unit, class priority, and misclassification rate (Saqib, Elbiaze, et al., 2023), the final penalty calculation for traffic classes also incorporates network path utilization.

4.6.2.3 Impact on penalty

The impact of traffic misclassification on penalties is illustrated in Fig. 4.10. In the initial scenario, despite a heightened rate of traffic misclassification, no supplementary penalty is imposed, as the network path utilization rate stays below 100%. Conversely, a marginal portion of erroneously mapped traffic from class 4

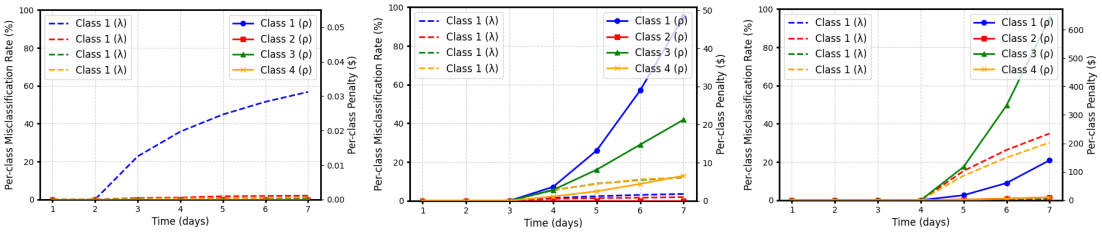


Figure 4.10 Improved penalty calculation for cases 1 (left), 2 (middle), and 3 (right).

leads to the imposition of a penalty, even for class 1, owing to QoS degradation on the path of class 1 caused by traffic congestion. Similarly, class 3 traffic was correctly mapped to its designated path, but the associated path became congested due to misclassification from other classes, leading to a higher penalty for class 3.

Hence, the impact of traffic misclassification on the final penalty is not solely determined by the misclassification rate or class priority but also depends on the QoS status of the network paths.

4.6.2.4 Results from adaptive classifier

The final set of results offers valuable insights into the impact of adapting to evolving traffic patterns and how this adaptation diminishes the influence of misclassification on network performance. We periodically incorporated newly received traffic patterns into the existing ML model to achieve this. The adaptation process was designed to alleviate the impact of misclassification on the network's QoS.

The results, as illustrated in Fig. 4.11, clearly demonstrate that enhancing the accuracy of our classifier has a direct and positive effect on the InP profit. Specifically, we observe a reduction in the total penalty as accuracy improves. In simpler terms, as we enhance the classifier's accuracy, it becomes more proficient at precisely

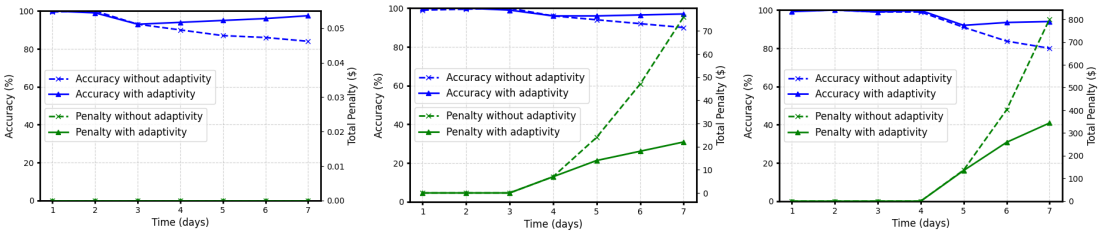


Figure 4.11 Adaptive classifier results for cases 1 (left), 2 (middle), and 3 (right).

assigning incoming traffic to the appropriate QoS classes. This, in turn, leads to diminished penalties while sustaining higher profits.

In summary, adapting to changing traffic patterns empowers the proposed approach to minimize the adverse effects of misclassification, ultimately maximizing the InP's profit and overall service quality.

4.7 Conclusion

This paper presents an approach that investigates the network performance and economic implications of traffic misclassification, introducing an adaptive classification method to address this challenge effectively. Our validation process involved characterizing multi-priority traffic, optimizing path allocation, and mapping incoming flows to various QoS classes. We found that in a multi-class, multi-path network, the impact of misclassification on penalty charges varies due to several factors, including the criticality level of each class, the misclassification rate, and the associated paths' utilization rate. Furthermore, we observed that the classifier's adaptivity significantly improves prediction accuracy, ensuring precise QoS class assignments and thus reducing penalties while increasing profits. This research contributes to a better understanding of the complex nature of traffic misclassification and its impact on *SLA* violations within intricate network environments.

REFERENCES

Afzal, Bilal et al. (2017). "Energy efficient context aware traffic scheduling for IoT applications". In: *Ad Hoc Networks* 62, pp. 101–115.

Azab, Ahmad et al. (2022). "Network traffic classification: Techniques, datasets, and challenges". In: *Digital Communications and Networks*.

Bosshart, Pat et al. (2014). "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3, pp. 87–95.

Chakraborty, Biswadeep et al. (2021). "Cost-aware feature selection for IoT device classification". In: *IEEE Internet of Things Journal* 8.14, pp. 11052–11064.

De Alwis, Chamitha et al. (2021). "Survey on 6G frontiers: Trends, applications, requirements, technologies and future research". In: *IEEE Open Journal of the Communications Society* 2, pp. 836–886.

Esteves, Jose Jurandir Alves et al. (2020). "Optimized network slicing proof-of-concept with interactive gaming use case". In: *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, pp. 150–152.

Grandini, Margherita, Enrico Bagli, and Giorgio Visani (2020). "Metrics for multi-class classification: an overview". In: *arXiv preprint arXiv:2008.05756*.

Kfoury, Elie F, Jorge Crichigno, and Elias Bou-Harb (2021). "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends". In: *IEEE Access* 9, pp. 87094–87155.

Leconte, Mathieu et al. (2018). "A resource allocation framework for network slicing". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, pp. 2177–2185.

Lu, Jie et al. (2018). "Learning under concept drift: A review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12, pp. 2346–2363.

Nechay, Daniel, Yvan Pointurier, and Mark Coates (2009). "Controlling false alarm/discovery rates in online internet traffic flow classification". In: *IEEE INFOCOM 2009*. IEEE, pp. 684–692.

Saqib, Muhammad, Halima Elbiaze, and Roch Glitho (2023). "A Profit-aware Adaptive Approach for In-Network Traffic Classification". In: *IEEE International Conference on Communications*.

Saqib, Muhammad, Zakaria Ait Hmitti, et al. (2022). "An Accurate & Efficient Approach for Traffic Classification Inside Programmable Data Plane". In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, pp. 6331–6336.

Sivanathan, Arunan et al. (2018). "Classifying IoT devices in smart environments using network traffic characteristics". In: *IEEE Transactions on Mobile Computing* 18.8, pp. 1745–1759.

Telikani, Akbar et al. (2021). "A Cost-Sensitive Deep Learning-Based Approach for Network Traffic Classification". In: *IEEE Transactions on Network and Service Management* 19.1, pp. 661–670.

Wanode, Sarvesh Sanjay, Milind Anand, and Barsha Mitra (2022). "Optimal Feature Set Selection for IoT Device Fingerprinting on Edge Infrastructure using Machine Intelligence". In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, pp. 1–6.

Wu, Yulei et al. (2022). "A survey of intelligent network slicing management for industrial IoT: integrated approaches for smart transportation, smart energy, and smart factory". In: *IEEE Communications Surveys & Tutorials* 24.2, pp. 1175–1211.

Xavier, Bruno Missi et al. (2021). "Programmable switches for in-networking classification". In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, pp. 1–10.

Xie, Guorui et al. (2022). "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation". In: *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, pp. 1938–1947.

Xiong, Zhaoqi and Noa Zilberman (2019). "Do switches dream of machine learning? toward in-network classification". In: *Proceedings of the 18th ACM workshop on hot topics in networks*, pp. 25–33.

Xu, Yi and Du Xu (2018). "Maximizing profit of network inp by cross-priority traffic engineering". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, pp. 1–6.

Yao, Haipeng et al. (2019). “Capsule network assisted IoT traffic classification mechanism for smart cities”. In: *IEEE Internet of Things Journal* 6.5, pp. 7515–7525.

Zhang, Li’ang and Suyun Geng (1986). “The complexity of the 0/1 multi-knapsack problem”. In: *Journal of Computer Science and Technology* 1.1, pp. 46–50.

Zhu, Shizhou et al. (2023). “CMTSNN: A deep learning model for multi-classification of abnormal and encrypted traffic of Internet of Things”. In: *IEEE Internet of Things Journal*.

Žliobaitė, Indrė, Mykola Pechenizkiy, and Joao Gama (2016). “An overview of concept drift applications”. In: *Big data analysis: new algorithms for a new society*, pp. 91–114.

CHAPTER V

ARTICLE 3 - A DATA-DRIVEN APPROACH TO MITIGATE EVOLVING VOLUMETRIC ATTACKS IN PROGRAMMABLE NETWORKS

Muhammad Saqib
Université du Québec à Montréal
Montréal (Québec), Canada

Halima Elbiaze, Professeur titulaire
Université du Québec à Montréal
Montréal (Québec), Canada

Roch Glitho, Professeur titulaire
Concordia University
Montréal (Québec), Canada

FOREWORD TO ARTICLE 3

The first two articles of this thesis leverage a network traffic classifier for QoS management. The third article addresses the security aspect, using an adaptive network traffic classifier to protect the network against evolving volumetric attacks. This paper has been submitted to **IEEE Transactions on Machine Learning in Communication and Networking (TMLCN)** for possible publication.

- M. Saqib, H. Elbiaze and R. H. Glitho, **A Data-Driven Approach to Mitigate Evolving Volumetric Attacks in Programmable Networks**, in IEEE Transactions on Machine Learning in Communication and Networking (SUBMITTED).

A early version of the above article is accepted for presentation in the scientific journal (conference) **IEEE Global Communication Conference (GLOBECOM)**, Dec 2024.

- M. Saqib, H. Elbiaze and R. Glitho, **In-Network Defense: Safeguarding the Network Against Evolving DDoS Attacks**, GLOBECOM 2024 - IEEE Global Communication Conference, Cape town, South Africa (ACCEPTED).

Papers submitted to IEEE TMLCN and accepted in IEEE GLOBECOM are peer-reviewed in accordance with the requirements set forth in the IEEE PSPB Operations Manual (Sections 8.2.1.C & 8.2.2.A). Each published paper has been reviewed by a minimum of two independent reviewers using a single-blind peer review process, where the identity of the reviewers is not known to the authors, but the reviewers know the identity of the authors. Papers are subject to plagiarism check before acceptance.

ABSTRACT

In-network machine learning (ML) offers a cutting-edge approach for promptly detecting malicious traffic. Existing methods often rely on one-size-fits-all ML models that fail to adapt to evolving attack traffic patterns, leading to a time-consuming and labor-intensive process for updating ML model from the control to the data plane. To address these limitations, we propose an automated, data-driven method for identifying novel malicious traffic patterns and updating ML model seamlessly in programmable networks. The proposed method sets drift detection thresholds based on baseline performance from historical (i.e., training) data and uses these thresholds to detect anomalies in unseen (i.e., testing) data. We continuously adjust the thresholds to accommodate data distribution changes and in-network inference results while minimizing sensitivity to minor fluctuations. We evaluate the proposed method using two intrusion detection datasets, CICIDS2017 and UNSW-NB15. The experimental results demonstrate its efficacy in safeguarding against evolving volumetric attacks. Additionally, we compare the conventional model performance-based drift detection method with an adaptive monitoring window-based approach, highlighting the latter's advantage in balancing drift detection efficacy and minimizing its adaptation impact, i.e., disruptions to normal network traffic are reduced by an average of 20%. The adaptive method dynamically adjusts the drift monitoring window size to adapt to the characteristics of the unseen traffic patterns.

Keywords : Network intrusion detection, machine learning, programmable data plane

5.1 Introduction

Machine learning (ML) has proven to be a valuable tool for traffic classification and network security (Pacheco et al., 2018). Network-oriented ML tasks are traditionally executed on servers or middleboxes (Doshi et al., 2018). However, recent advancements in programmable data planes (PDP) have introduced new possibilities for cost-effective, flexible, high-performance network security (Chen, H. Liu, et al., 2022). Network administrators can directly program packet processing logic using network programming languages, such as P4 (Bosshart, Daly, et al., 2014), enabling the offloading of ML operations to the network data plane. This approach allows for in-network ML (Chen, Wu, et al., 2023), which strategically divides the training and inference phases between the control and data planes for efficient network security.

In-network inference is a key enabler for developing efficient and accurate attack detection solutions (Zhou et al., 2023; Xavier et al., 2022). However, the evolving landscape of malicious activities introduces variations in the network traffic patterns using different protocols, attack vectors, and the data distribution of malicious traffic (Tgavalekos et al., 2018). Such dynamic movement of network traffic properties causes *concept drift* (Amin et al., 2023), which can be a sign of anomalies because it represents a departure from the established patterns and statistical properties on which an ML model has been trained (Tan et al., 2020). As the data distribution shifts over time, it may indicate the emergence of new or unusual patterns, affecting the relevance of input features (i.e., port number and statistical properties of packet size) of a trained ML model and their classification decision boundaries. This shift in data distribution may ultimately lead to decreased accuracy because the model must adapt to evolving patterns in the data for accurate prediction (Gu, 2019). Therefore, adaptive ML models that recognize and respond to changing traffic

patterns are crucial for an effective threat defense (Kuppa and Le-Khac, 2022).

One of the major limitations of existing in-network attack detection methods is the one-shot learning of ML models (Xie et al., 2023; Xavier et al., 2021; Xavier et al., 2022; X. Zhang et al., 2021; Musumeci et al., 2022; Coelho and Schaeffer-Filho, 2022; Zhou et al., 2023). The mapping rules of these models are initially defined based on historical data at the control counterpart following preprocessing and hyperparameter tuning. The resulting rules are then mapped onto the data plane for online inference. Because the entire ML operation is strategically divided into control and data planes, monitoring individual instances of malicious activity, identifying previously unseen malicious traffic patterns using preprocessing and hyperparameter tuning, and updating the ML model from the control to the data plane are infeasible in such a pragmatic framework. These results in frequent updates to the ML model, which can cause significant network overhead and disrupt normal network traffic. In addition, in-network ML updates are more complex than server-based ML, and conducting a hitless update process with minimal impact on normal packet forwarding performance is challenging (Zheng, Hong, et al., 2023). Therefore, it is essential to integrate the functionalities of the control and data planes to execute in-network ML updates that address emerging anomalies with minimal disruption caused by the updates. However, existing works still need to address the challenge of detecting and seamlessly adapting to unseen malicious traffic patterns in programmable networks to protect against evolving attacks.

This study aims to automate the drift detection and ML model updating process in a programmable network to seamlessly adapt to changes in traffic patterns for safeguarding the network against evolving volumetric attacks. Our contribution encompasses a novel framework that defines the baseline performance from historical (i.e., *training*) data to establish drift detection thresholds and continuously updates them based on unseen (i.e., *testing*) data, that is, based on the changing data

distribution and in-network inference results. In particular, drift detection and adaptation logic are distributed over the control and data planes. The control plane initially trains the ML model using historical data and updates it based on unknown traffic patterns from unseen data. The data plane maintains a customized monitoring sketch for extracting features from traffic flows and employs the updated rules of ML model for online inference. The inference results are regularly monitored to detect drifts in unseen patterns, triggering updates to the ML model as needed.

We first validate the existence of drifts and their implications using two intrusion detection datasets: CICIDS2017(Sharafaldin et al., 2018) and UNSW-NB15(Moustafa and Slay, 2015). Subsequently, we evaluate the efficacy - the ability to detect drifts under defined conditions - of the proposed data-driven method. The experimental results reveal the efficacy of the adaptive method for safeguarding against evolving volumetric attacks. We further compare the conventional model performance-based drift detection method with that enabled by an adaptive monitoring window, emphasizing the latter's superiority in balancing the efficacy of drift detection and its adaptation impact on the disruption of normal network traffic. This balance is achieved by dynamically adjusting the drift monitoring window size to better adapt to the characteristics of the unseen data.

The key contributions of this work are as follows.

- We propose a self-adaptive traffic classification method that detects and responds to evolving volumetric attacks. By continuously monitoring traffic feature distributions and detecting concept drift, the system updates the ML model to maintain classification effectiveness in programmable networks.
- The framework leverages the interplay between the control and data planes for efficient adaptation. The data plane extracts traffic features and logs performance degradation, while the control plane verifies drift using statistical

methods and updates the ML model to refine classification rules. This design helps balance adaptation with operational overhead.

- To further optimize adaptation, we introduce an adaptive windowing strategy that dynamically adjusts the the frequency of retraining. This mechanism ensures that updates occur only when necessary, reducing unnecessary retraining while maintaining detection performance.

The remainder of this paper is organized as follows. Section 5.2 provides an overview of the related work. Section 5.3 introduces the proposed method. Section 5.4 discusses the validation methodology, and Section 5.5 presents the experimental results. Finally, Section 5.6 concludes the study.

Table 5.1 Summary of the related work

Category	Reference	Learning based	Line speed	Drift detection	Runtime updates	Comment
1st	(Barradas et al., 2021; Hireche et al., 2022)	✓	x	x	x	Classification in control plane
2nd	(M. Zhang et al., 2020; Z. Liu et al., 2021)	x	✓	x	✓	Threshold driven traffic filters
3rd	(Ding et al., 2021)	x	✓	x	x	Predefined threshold based rules
	(Musumeci et al., 2022)	✓	✓	x	x	-
	(X. Zhang et al., 2021; Chen, H. Liu, et al., 2022; Coelho and Schaeffer-Filho, 2022; Zhou et al., 2023)	✓	✓	x	✓	-
	This study	✓	✓	✓	✓	Learning based threshold adjustment

1st: collect flow information in the data plane and traffic classification on the control plane; 2nd: like 1st category with classification logic based on threshold-driven traffic filters; 3rd: embed the learning-based models in the data plane.

5.2 Related work

This section comprehensively provides an overview of the current volumetric attack detection approaches in programmable networks.

We position our approach with state-of-the-art in Table 5.1. Our method is designed to meet the following key requirements: learning-based traffic classification, line-rate processing, drift detection, and runtime updates. These design requirements

are essential for building an adaptive method that addresses the dynamic nature of volumetric attacks in programmable networks.

The first category of prior art consists of methods that extract valuable flow information in the data plane to support broader applications. For example, Othmane et al. (Hireche et al., 2022) and FlowLens (Barradas et al., 2021) use programmable switches to collect flow-distribution information, which is subsequently used by the control plane for learning-based traffic classification. While these methods meet the need for learning-based classification, they rely on the control plane for decision-making, limiting their ability to process data at line speed and respond to real-time network conditions.

The second category includes Poseidon (M. Zhang et al., 2020) and Jaqen (Z. Liu et al., 2021), both of which adopt designs similar to FlowLens (Barradas et al., 2021). However, the collected flow information is directly processed in the data plane to identify volumetric attacks at the line rate. These approaches fulfill the line-rate processing requirement but rely on threshold-driven filters rather than adaptable learning-based models, making them less effective in scenarios where handcrafted filters cannot accurately represent traffic analysis logic.

The third category of prior art takes a step further to realize intelligence in the data plane. The concept is referred to as in-switch inference, which brings remarkable benefits to the network, i.e., reducing latency and increasing throughput (Chen, H. Liu, et al., 2022). As a result, there has been growing interest in integrating the output of ML algorithms into the data plane for the classification of network traffic at an early stage. Ding et al. (Ding et al., 2021) propose an in-network DDoS victim identification method using a sketch-based data structure that estimates the per-destination flow cardinality using a fixed threshold-based rule to directly identify victims in the data plane of the network. Although this approach supports

line-rate processing, it lacks learning-based traffic classification, limiting its ability to adapt to evolving traffic patterns.

More advanced methods, such as those proposed by Musumeci et al. (Musumeci et al., 2022), BACKORDERS (Coelho and Schaeffer-Filho, 2022), and NetBeacon (Zhou et al., 2023), attempt to overcome these limitations by incorporating learning-based models directly into the data plane for early-satge volumetric attack detection. Musumeci et al. [17] demonstrate how different ML classifiers can improve accuracy and reduce inference time. Similarly, BACKORDERS [18] focuses on increasing classification accuracy and efficiency. NetBeacon [7] advances the state-of-the-art by introducing a multi-phase sequential model architecture that performs dynamic packet analysis at near-line rate, meeting both learning-based classification and line-rate processing requirements. Furthermore, these methods enable real-time updates of ML model directly within the data plane, enhancing adaptability to evolving traffic patterns.

Other recent works on in-switch inference include IIsy (Zheng, Xiong, et al., 2024), a hybrid approach that maps ML-based classification models to programmable switches using tree-based classifiers and a backend-assisted decision-making process to reduce computational overhead on the switch. Additionally, a learning-based ransomware mitigation system (Friday, Bou-Harb, et al., 2022) leverages Random Forest models within programmable switches for detecting and mitigating ransomware activity without relying on deep packet inspection. The In-Network Classification (INC) method (Friday, Kfoury, et al., 2022) extends this paradigm by incorporating a bagging ensemble to detect botnet infections in real-time within Tbps traffic flows, forwarding identified threats to a controller for further clustering and inference.

One of the primary areas for improvement of the current art is the one-shot learning

of ML models. Existing approaches involve experimentally defining mapping rules for substantial historical data through preprocessing and hyperparameter tuning, after which the learned rules are mapped to the data plane for online inference. However, the attacking traffic landscape is continually evolving, which can compromise the relevance of the input features and their classification decision boundaries (Khamassi et al., 2018; Wang and Jones, 2021). As a result, even though a one-size-fits-all ML model may exhibit initial accuracy, it can quickly become outdated as traffic patterns change. Therefore, it is crucial to implement a learning-based model that can efficaciously adjust to fluctuations in traffic patterns and continuously classify traffic flows with the utmost accuracy.

The ability to update ML models in the *Match-Action Tables (MATs)* of programmable network devices is considered in several studies (Xie et al., 2023; X. Zhang et al., 2021; Chen, H. Liu, et al., 2022; Coelho and Schaeffer-Filho, 2022; Zhou et al., 2023). However, the effectiveness of the inference process relies on the manual tuning of ML models, from control to the plane, which requires a predefined set of mapping rules for specific scenarios. For example, techniques for detecting anomalies require defining an optimal set of features and their classification decision boundaries (i.e., threshold values). Consequently, administrators and security researchers must invest significant time and effort in investigating various parameters and determining the most suitable ones for evolving attack types. This process is time-consuming and labor-intensive, hindering the adoption of these techniques in modern production networks (Chen, Wu, et al., 2023). Therefore, it is essential to achieve automatic parameter tuning of entire ML operations in programmable networks to seamlessly adapt to changing traffic patterns and maintain flow identification accuracy. However, existing works have not addressed the need to detect and seamlessly adjust to unseen malicious traffic patterns in programmable networks to protect against evolving attacks.

Our work falls into the third category (in-switch inference) but extends it further by introducing adaptivity to traffic classification. Unlike existing studies, which rely on static ML models, our framework automates drift detection and adaptive ML model updates. By integrating automated drift detection and ML model adaptation, our study addresses the critical limitations of existing in-switch ML methods, making it more adaptive, efficient, and suitable for evolving volumetric attack detection in programmable networks.

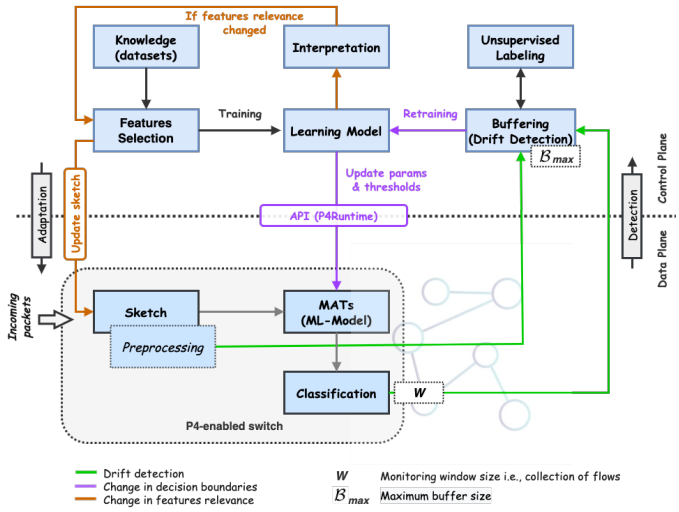


Figure 5.1 System design

5.3 System design

This section introduces the proposed adaptive approach to identify and seamlessly adapt to unknown traffic patterns in programmable networks. A high-level overview of the proposed framework is presented in Fig. 5.1. The control plane trains the ML model using historical data and updates it based on unknown traffic patterns from unseen (i.e., future) data. The data plane maintains a customized monitoring sketch for extracting features from traffic flows and uses the updated ML model rules for online inference. In the following subsections, we present the details of

the distributed entities across the control and data planes, and the integrated data-driven method used to automate drift detection and adaptation processes.

5.3.1 Control plane

The control plane consists three essential logical components of offline learning: *knowledge*, *learning*, and *interpretation*. The following subsections provide a detailed discussion of each component.

5.3.1.1 Knowledge

The datasets used in this study contain both *benign* and *malicious* traffic patterns. The knowledge is derived from two primary sources: historical data (the initial data) and future data, representing unseen traffic patterns. The historical data is utilized for the initial learning phase, while the unseen data supports the continuous learning of emerging traffic patterns.

To learn from the unseen data, the control plane processes the encapsulated messages generated by the data plane, which contains the extracted features information of the newly received traffic flows. Specifically, when the system detects a drift warning, the control plane begins to *buffer* the extracted feature information and applies an *unsupervised learning* algorithm to label the buffered instances for model retraining. Feature selection is applied to ensure only the most relevant features are retained for classification, optimizing model performance. Meanwhile, unsupervised labeling assigns labels to the buffered instances without requiring human intervention, making the retraining process more efficient.

5.3.1.2 Learning

The control plane uses a *learning module* to gain insights into benign and malicious traffic patterns from historical and unseen data. The dataset, denoted as D, consists of packets from subflows and is divided into two subsets: a training set, D_{tr}, and a testing set, D_{ts}. The *learning module* initially trains the ML model on D_{tr}. Subsequently, it embeds the learned rules, represented as *Match-Action (MA)* rules, into *MATs* within a P4-enabled programmable switch (Bosshart, Gibb, et al., 2013) for real-time inference. The embedding process is facilitated by a control plane application programming interface (API) called P4Runtime.

5.3.1.3 Interpretation

The attackers aim to increase discrepancies between the training and testing data by modifying traffic characteristics, which can shift the relevance of input features and classification decision boundaries. Given limited memory constraints and the inability to support complex operations in PDP (Sharma et al., 2017), using a classifier with a minimal set of features is preferable. Therefore, it is important to establish and maintain an optimal set of features to enable efficient and precise classification. In light of these considerations, we consider it crucial to employ *explainability* technique with a focus on feature importance to enhance the transparency and comprehension of the decision-making processes of ML model. This helps to preserve the most pertinent features for online inference.

5.3.2 Data plane

The data plane phase involves extracting features and applying inference logic to detect and mitigate attacks.

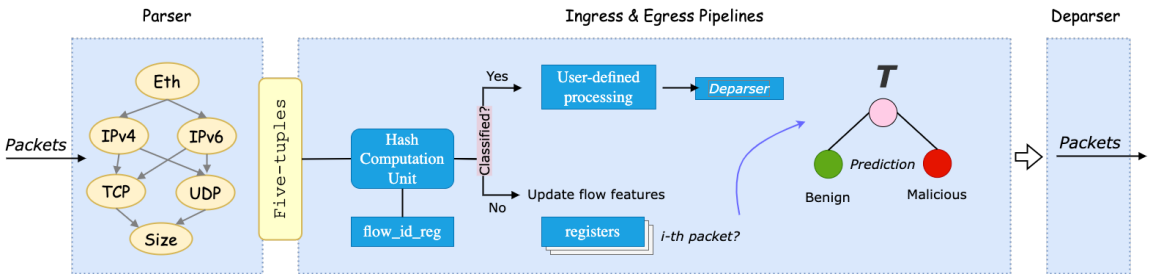


Figure 5.2 Online inference process

The general process of complete online inference is illustrated in Fig. 5.2. The data plane first uses a monitoring sketch to extract a selected set of feature values from incoming traffic flows through in-band feature extraction. This method provides flexibility and immediacy by extracting features directly from different layers in the data plane rather than from dumped captures. In-band feature extraction is accomplished through protocol-independent processing pipelines in the PDP, which programmatically process packet headers. The P4 language defines the packet header fields, enabling the parser to identify protocols and extract header information (Bosshart, Daly, et al., 2014). The parser operates as a state machine, parsing the headers layer by layer.

The parser module extracts relevant features for incoming packets, such as the five tuples (i.e., IPs, ports, and protocol type) and packet size from the header. These values are stored in the metadata of the switch pipeline. The next step is calculating a hash value for each flow based on the five tuples from the packet header. The monitoring sketch is a data structure implemented using registers in the *SRAM* of the switch. The *flow ID* register tracks all identified flows to apply appropriate actions to the corresponding packets. The switch performs specified actions on incoming packets belonging to classified flows, such as forwarding normal packets and dropping anomalies. In other words, packets belonging to the classified flows are immediately dropped or processed at a line rate.

The parser extracts the flow information and stores it to registers in *SRAM* until the packet counter reaches the threshold for classification. Once the packet counter meets the threshold, the switch calculates statistics such as the minimum, maximum, and average packet size of the flow and stores these values in the associated registers. Classification occurs on the resulting feature values following *MATs*, and the flow class is ultimately saved in the flow register. When the classification accuracy of the inference results begins to degrade, the data plane starts logging the preprocessed feature information to the control plane for further analysis and necessary updates to adapt to new traffic patterns.

5.3.3 Integrated data-driven method

The control plane offers ample storage and computational resources, but its use introduces latency and throughput overheads. In contrast, the data plane excels in providing near line-rate inference capabilities for swift decision-making (Chen, H. Liu, et al., 2022); however, it has limitations in supporting computational resources and operations (Sharma et al., 2017). We propose an integrated method that merges the strengths of both planes to safeguard the network against evolving volumetric attacks. Specifically, we leveraged the capability of the control plane to determine drift detection thresholds based on the baseline performance derived from historical data, continuously updating the thresholds during the online inference phase in the data plane. The mathematical formulation of the drift detection parameter definition and algorithms for continuous learning are discussed in the following subsections.

5.3.3.1 Defining baseline performance

The dataset D comprises n labeled data instances, each characterized by a k-dimensional feature vector x_i and a binary label y_i. It is expressed as

$$D = \{(x_i, y_i)\}_{i=1}^{n},$$

where $x_i = (x_{i1}, x_{i2}, \ldots, x_{ik}) \in \mathbb{R}^k$ represents the feature vector of i-th instance, and $y_i \in \{0, 1\}$ signifies the corresponding label indicating whether the instance is benign (0) or malicious (1).

As new traffic patterns emerge, shifts in data distribution may occur. To analyze the drifting behavior of data instances, we partition the dataset D having n labeled data instances into set of segments $\{s_1, s_2, \ldots, s_m\}$, each of size η, where $\eta = \lfloor\sqrt{n}\rfloor$. The size of η is chosen to ensure sufficient data for reliable statistical measurements while keeping the segments small enough to detect changes in the data distribution. The segments containing malicious data instances are considered drifting segments. The subset of non-drifting segments is denoted as $S' \subseteq \{s_l, \ldots, s_m\}$, and the subset of drifting segments is denoted as $S \subseteq \{s_l, \ldots, s_m\} \setminus S'$. The learning algorithm $\mathcal{L}$ is presented with the labeled data instances. The performance of the algorithm in terms of accuracy, is represented by A. The segmentation of the dataset allows us to measure the change in data distribution and its impact on the performance of $\mathcal{L}$, denoted as Δd_l and ΔA_l, respectively, for each segment l.

To efficaciously monitor and respond to data drift, it is essential first to establish a baseline performance that reflects the expected behavior of the ML model under normal conditions. We begin by defining the baseline performance by comparing the accuracy A between S and S', thereby quantifying the impact of drift on A. Establishing this baseline helps to determine the drift detection thresholds, i.e., drift warning and drift alarm (Gama et al., 2004). In this study, drift warning refers

to when the inference result degrades from the most consistent level of accuracy obtained over existing data. A drift alarm refers to an observed drop in historical accuracy due to a data distribution shift. By establishing a baseline performance for the data, we create a benchmark for future predictions.

We continuously update the thresholds based on the newly received traffic patterns. The ongoing adjustment of thresholds aims to balance drift detection accuracy and its adaptation impact. This approach enables an in-intrusion detection system to efficiently adapt to emerging patterns and maintain accuracy in evolving network behavior. The mathematical formulation for defining baseline performance and determining drift detection thresholds is provided. The necessary mathematical notations and descriptions are presented in Table 5.2.

We measure the change in data distribution by calculating the statistical measures, i.e., mean μ and standard deviation σ, for each feature j of l-th segment.

For non-drifting segments S':

$$\mu'_{l,j} = \frac{1}{|s_l|} \sum_{i \in s_l} x_{i,j}, \quad \forall j \in \{1, 2, \ldots, j\}, \quad \forall l \in S' \tag{5.1}$$

$$\begin{aligned} \sigma'_{l,j} &= \sqrt{\frac{1}{|s_l|} \sum_{i \in s_l} (x_{i,j} - \mu'_{l,j})^2}, \\ &\forall j \in \{1, 2, \ldots, k\}, \quad \forall l \in S' \end{aligned} \tag{5.2}$$

For drifting segments S:

$$\mu_{l,j} = \frac{1}{|s_l|} \sum_{i \in s_l} x_{i,j}, \quad \forall j \in \{1, 2, \ldots, k\}, \quad \forall l \in S \tag{5.3}$$

Table 5.2 Notation Table

Notation	Description
D	Historical data: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
D_{tr}, D_{ts}	Training and testing data
$\mathcal{L}$	Learning algorithm trained on D_{tr}
S', S	Set of non-drifting and drifting segments
$\mu_{l,j}$, $\sigma_{l,j}$	Mean and standard deviation of feature j from l-th segment
$\Delta d_{l,j}$	Magnitude of change in feature j of l-th segment
$\Delta d_{l,j}$	Cumulative magnitude of change for all features of l-th segment
A_l	Accuracy of $\mathcal{L}$ at l-th segment
θ_d	Correlation coefficient represents how the change in data distribution affects the accuracy of $\mathcal{L}$
$\sigma^2(A)$	Variance of accuracy across drifting segments
α	Drift warning threshold
β	Drift alarm threshold
W	Batch size, i.e., collection of traffic flows
M	Moving average of accuracy of last M number of batches
δ	Adjustment factor for modifying the batch size W
$\mathcal{B}_{max}$	Maximum size of the buffer
$\mathcal{R}_{rate}$	Retraining rate of $\mathcal{L}$.

$$\sigma_{l,j} = \sqrt{\frac{1}{|s_l|}\sum_{i \in s_l}(x_{i,j} - \mu_{l,j})^2}, \quad \forall j \in \{1, 2, \ldots, k\}, \quad \forall l \in S \tag{5.4}$$

Next, we calculate the difference in statistical measures between drifting and non-drifting segments:

$$\Delta\mu_{l,j} = \mu_{l,j} - \mu'_{l,j}, \quad \forall j \in \{1, 2, \ldots, k\}, \quad \forall l \in S, \quad \forall l \in S' \tag{5.5}$$

$$\Delta\sigma_{l,j} = \sigma_{l,j} - \sigma'_{l,j}, \quad \forall j \in \{1, 2, \ldots, k\}, \quad \forall l \in S, \quad \forall l \in S' \tag{5.6}$$

We calculate the magnitude of change for each feature of l-th segment to quantify

the shift in data distribution. The magnitude of change for each feature j in segment l is defined as the Euclidean distance between the means and standard deviations of drifting and non-drifting segments:

$$\begin{aligned} \Delta d_{l,j} = \sqrt{(\Delta\mu_{l,j})^2 + (\Delta\sigma_{l,j})^2}, \\ \forall j \in \{1, 2, \ldots, k\}, \quad \forall l \in S \end{aligned} \tag{5.7}$$

We then aggregate the magnitudes of change across all features of l-th segment to obtain the cumulative magnitude of change for the entire segment:

$$\Delta d_l = \sqrt{\sum_{j=1}^{k} (\Delta d_{l,j})^2}, \quad \forall l \in S \tag{5.8}$$

To measure the change in accuracy, we first calculate the baseline accuracy for non-drifting and drifting segments:

$$A'_l = \frac{1}{|s_l|} \sum_{i \in s_l} \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}, \quad \forall l \in S' \tag{5.9}$$

$$A_l = \frac{1}{|s_l|} \sum_{i \in s_l} \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}, \quad \forall l \in S \tag{5.10}$$

Where TP indicates that the observation is positive, and the sample is predicted to be positive whereas TN reports that the observation is negative, and the sample is predicted to be negative. Similarly, FP represents that the observation is negative, but the sample is predicted to be positive. FN indicates that the observation is positive, but the sample is predicted to be negative.

We then calculate the change in accuracy for each drifting segment compared to non-drifting segment:

$$\Delta A_l = A_l - A'_l, \quad \forall l \in S \tag{5.11}$$

Next, we analyze the accuracy drops across drifting segments to determine the minimum and maximum accuracy changes observed:

$$A_{min} = \min(\Delta A_l), \quad \forall l \in S \tag{5.12}$$

$$A_{max} = \max(\Delta A_l), \quad \forall l \in S \tag{5.13}$$

To understand how changes in data distribution affect the accuracy of the model, we calculate the correlation coefficient between the magnitude of change in data distribution Δd_l and the change in accuracy ΔA_l for drifting segments S. This analysis helps in quantifying the relationship between the shift in data distribution and the observed accuracy variations, providing insight into the impact of data drift on model performance. First, we compute the average magnitude of change in data distribution and accuracy across drifting segments:

$$\overline{\Delta d} = \frac{1}{|S|} \sum_{l \in S} \Delta d_l \tag{5.14}$$

$$\overline{\Delta A} = \frac{1}{|S|} \sum_{l \in S} \Delta A_l \tag{5.15}$$

We then calculate the correlation coefficient θ_d that represents how the change in data distribution affects the obtained accuracy and is used to verify the drift occurrence before triggering the model updates:

$$\theta_d = \frac{\sum_{l \in S} (\Delta d_l - \overline{\Delta d})(\Delta A_l - \overline{\Delta A})}{\sqrt{\sum_{l \in S} (\Delta d_l - \overline{\Delta d})^2} \sqrt{\sum_{l \in S} (\Delta A_l - \overline{\Delta A})^2}} \tag{5.16}$$

Finally, we calculate the variance of the accuracy $\sigma^2(A)$ across drifting segments:

$$\sigma^2(A) = \frac{1}{|S|} \sum_{l \in S} (\Delta A_l - \overline{\Delta A})^2 \tag{5.17}$$

To determine the drift warning α and drift alarm β thresholds, we define them based on baseline performance, where

$$\alpha = A_{max} - 1 \tag{5.18}$$

$$\beta = \sigma^2(A) \tag{5.19}$$

The drift warning threshold α is set as $A_{\max} - 1$ to signal a one-point drop from the maximum observed accuracy, while the drift alarm threshold β is set as the variance $\sigma^2(A)$ to manage the tradeoff between detecting significant drifts promptly and smoothing short-term fluctuations in accuracy across drifting segments. We keep updating β on unseen data, that is, a collection of traffic flows W:

$$\beta = \frac{1}{M} \sum_{p=q-M+1}^{q} A_p(W) \tag{5.20}$$

Where:

$$A_{min} \leq \beta < \alpha < A_{max} \tag{5.21}$$

The continuous updates to the parameter β are based on the average accuracy of the last M batches, i.e., collections of traffic flows, each denoted as W. Let $M = 4$, where M represents the number of batches used to calculate the moving average. The value of M is chosen as a typical tradeoff between detecting drift responsively and smoothing short-term fluctuations in accuracy. This balance ensures that the system can react promptly while avoiding over-sensitivity to minor variations. Let q be the current index of W, and the index p iterates over the previous M flow

collections, from $q - M + 1$ to q. The denominator M normalizes the sum of the M collections. Additionally, we bound the parameters α and β using the minimum and maximum accuracy obtained from the baseline performance to prevent the system from continuously being in a drifting state.

The parameter W is dynamically adjusted using an adjustment factor δ to enhance adaptability to unseen traffic patterns:

$$\delta = (1.0 - A_q) \times W \tag{5.22}$$

$$W = \begin{cases} W + \delta & \text{if accuracy is increasing} \\ W - \delta & \text{if accuracy is decreasing} \\ W & \text{otherwise} \end{cases} \tag{5.23}$$

The adjustment factor δ is introduced to modify W based on the current accuracy level. The degree of adjustment depends on the disparity between the current accuracy and the desired level. A larger increment or decrement to W occurs when the accuracy significantly deviates from the desirable level. Consequently, the monitoring parameter W becomes more responsive to updates during severe accuracy drops and adjusts more conservatively for minor accuracy deviations.

5.3.3.2 Data-driven approach

The complete drift detection and adaptation processes are outlined in Algorithms 5 and 6, respectively. The detection algorithm uses historical and unseen data samples as input, along with the initially determined parameters and thresholds. A *STATE* variable initialized to *zero* at line 3 signifies the *normal, warning, and drifting* conditions of the system. The subsequent lines initialize the buffer and the adjustment factor for W.

Algorithm 5: Drift Detection

Input: D_{tr}, D_{ts}, $\mathcal{L}$, α, β, W, $\mathcal{B}_{max}$, M, θ_d, $\mathcal{U}$, X_{SHAP}, FS_{KBest}

Output: A[], $\mathcal{R}_{rate}$, decisionRules

$\mathcal{R}_{count} = 0$, $\mathcal{R}_{rate} = 0$, $A = []$;
shapValues = [], decisionRules = [];
STATE $\leftarrow 0$; $\mathcal{B} \leftarrow \theta$; $\delta \leftarrow \theta$;
while W_q *in* D_{ts} do
    $A_q \leftarrow \text{accuracy}(W_q)$;
    if *STATE == 0 AND* $A_q < \alpha$ then
        STATE $\leftarrow 1$; // drift warning
    if *STATE == 1 AND* $\mathcal{B} < \mathcal{B}_{max}$ then
        $\mathcal{B} \leftarrow W_q$;
        if $A_q < \beta$ *OR* $\mathcal{B} == \mathcal{B}_{max}$ then
            STATE $\leftarrow 2$; // drift alarm
        if $A_q > \alpha$ then
            $\mathcal{B} \leftarrow \theta$; STATE $\leftarrow 0$;
    if *STATE == 2* then
        $\Delta d_q = \sqrt{\sum_{j=1}^{k} (\Delta d_{l,j})^2}$;
        if $\Delta d_q > \theta_d$ then
            STATE $\leftarrow 0$; // non-drifting state
            decisionRules = DriftAdaptation(D_{tr}, $\mathcal{B}$, $\mathcal{L}$, $\mathcal{U}$, X_{SHAP}, FS_{KBest});
            $\mathcal{R}_{count} = \mathcal{R}_{count} + 1$;
    $A.append(A_q)$;
    $\beta = \frac{1}{M} \sum_{p=q-M+1}^{q} A_j$; $\delta = |1.00 - A_q| \cdot W_q$;
    if $A_q < A_{q-1}$ then
        $W_q = W_q - \delta$;
    if $A_q > A_{q-1}$ then
        $W_q = W_q + \delta$;
    $\mathcal{B}_{max} = M \cdot W_q$;
$\mathcal{R}_{rate} = \mathcal{R}_{count}/k$;
return *A[], $\mathcal{R}_{rate}$, decisionRules []*

Algorithm 5 iterates over unseen data D_{ts} from lines 4 to 28, continually adjusting the thresholds (β; W, which is initially set to the determined segment size η; and $\mathcal{B}_{max}$) to strike a balance between drift detection accuracy and its adaptation impact. For each W, which represents the monitoring flow count, the obtained accuracy is assigned to variable A_q at line 5. The state of the system is reflected by the current accuracy A_q. The first condition checks if the system is non-drifting (i.e., when the *STATE* is 0) but A_q has dropped from the determined α, then the *STATE* becomes *one* that indicates a drift warning. When there is a drift warning, buffer $\mathcal{B}$ begins to keep logs (i.e., in-band extracted features information) of incoming flows until the maximum buffer size is reached (see lines 8-13). We define $\mathcal{B}_{max}$ as the stopping condition for buffering. Meanwhile, if the accuracy increases back to the threshold α, the algorithm considers the drift warning to be a false alarm, resets the buffer at line 13, and reinitializes the *STATE* to *zero*. If A_q further degrades to the drift alarm threshold β or the buffer size reaches $\mathcal{B}_{max}$, the system transitions to the drift alarm state (see line 10), and the drift verification and adaptation process starts from line 14.

During the drift alarm, the system calculates the change in data distribution for all features $j \in \{1, 2, \ldots, k\}$ of the buffered instances. A predetermined threshold θd is used to detect changes in the data distribution. The threshold helps verify whether the distribution of buffered instances Δd_q has changed enough to indicate a drift. If this condition is true, the drift adaptation process begins by calling an adaptation algorithm at line 18. The adaptation algorithm takes the historical D_{tr} and buffered instances $\mathcal{B}$, along with the unsupervised learning algorithm and explainability methods as inputs, and returns updated decision rules for online inference in the data plane. Each time the adaptation algorithm is called, the model retraining count variable R_{count}, which represents the retraining rate, is incremented by one.

During drift adaptation, Algorithm 6 first applies an unsupervised learning algorithm to $\mathcal{B}$ to label instances for the model retraining. Next, the model is retrained on $\mathcal{B}$ and D_{tr} at line 2. From lines 3-7, explainability methods that focus on identifying important features are applied to the predictions of ML model to extract the top K features that contribute the most to the predictions of the classifier. The record of the top K features is maintained, triggering the need for sketch updates in the data plane if the relevance of the features changes over time. Finally, the updated decision rules are derived at line 11 and returned as the output of the algorithm.

Algorithm 6: Drift Adaptation

Input: D_{tr}, $\mathcal{B}$, $\mathcal{L}$, $\mathcal{U}$, X_{SHAP}, FS_{KBest}

Output: decisionRules []

1 $\mathcal{B} = \mathcal{U}(\mathcal{B})$ // Label buffered data using unsupervised learning

2 $\mathcal{L}$.fit(D_{tr}+B); // Retrain model with training and labeled buffer data

3 currentShap = $\mathcal{L}$.featureImportance(X_{SHAP});

4 // Compute SHAP values for feature importance

5 shapValues.append(currentShap);

6 // Select top K features

7 currentTopKFeatures = $FS_{\text{KBest}}(shapValues)$;

8 if *currentTopKFeatures* $\neq$ *prevTopKFeatures* then

9 Trigger: 'update data plane monitoring sketch for top K features';

10 prevTopKFeatures = currentTopKFeatures;

11 decisionRules [] = $\mathcal{L}$.getRules(); // Get updated decision rules

12 return *decisionRules []*

Because we opted for a data-driven approach, we continually adjusted the drift detection thresholds based on current classification accuracy. Algorithm 5 adjusts the thresholds at lines 21-26. A moving average parameter M is defined to reflect the current classification performance to β; hence, β is iteratively adjusted based

on the average accuracy of the last M flow collections (see line 21). Additionally, instead of using a constant W, we adjust W to reflect the dynamics of accuracy. For instance, when drift occurs, W becomes more aggressive and frequently triggers retraining by decreasing size and vice versa. A δ variable is used to adjust W, reflecting the change in the current accuracy A_q. Hence, the effect of δ on W is based on the current performance of ML model and size of W. The following conditions use δ to increment or decrement into W. Finally, the model retraining rate $\mathcal{R}_{rate}$ is calculated by dividing the number of retraining events i of W by R_{count}.

5.4 Validation methodology

This section provides an overview of the datasets used for validation, explains the rationale behind feature selection and the choice of ML algorithm, and presents the experimental setup in Sections 5.4.1, 5.4.2, and 5.4.3.

5.4.1 Description of the datasets

We use two network intrusion detection datasets to evaluate the proposed method for mitigating evolving volumetric attacks. The first dataset CICIDS2017, (Sharafaldin et al., 2018) provided by the Canadian Institute of Cybersecurity (CIC), which includes the most updated cyberattack scenarios. As different types of attacks are launched at various times to create the dataset, the attack patterns change over time, causing multiple drifts. The second dataset, UNSW-NB15 (Moustafa and Slay, 2015), is developed at the University of New South Wales and captures a wide range of attack types. Like CICIDS2017, UNSW-NB15 spans different periods, reflecting the dynamic landscape of cyber threats and introducing variations in attack patterns.

Table 5.3 Datasets summary

CICIDS2017		UNSW-NB15	
Label	Flows	Label	Flows
BENIGN	19101	BENIGN	40550
Hulk	34587	Exploits	4033
GoldenEye	1540	Fuzzers	2914
Slowhttptest	843	Generic	593
Slowloris	824	DoS	569

The datasets consist of traffic patterns gathered over several days. However, we focus on specific portions to facilitate the evaluation process. In the case of the CICIDS2017 dataset, we select the traffic portion generated specifically on Wednesday, July 5, 2017. For UNSW-NB15, we utilize a 16-hour simulated period on January 22, 2015. It is important to note that using all data samples for model development is often impractical and unnecessary. Therefore, an effective data-sampling method is not just beneficial, but essential for selecting highly representative data. A well-chosen sample can significantly reduce computational complexity, enhance model performance, and prevent overfitting by focusing on the most relevant data points. In our case, we employ the k-means cluster sampling method, a technique that groups data samples based on similarity and selects a proportion of samples from each cluster. Given the substantial size of the datasets, this method allows us to choose 10% of the original data from each dataset for evaluating the proposed framework. Importantly, the k-means cluster sampling method generates a highly representative and high-quality subset by discarding redundant data points, distinguishing it from other sampling methods.

Following the implementation of the k-means clustering sampling method, two representative subsets are obtained: the CICIDS2017 subset, which contains 56,895 instances, and the UNSW-NB15 subset, which includes 48,659 instances. The subsets serve as the basis for evaluating the proposed method. The datasets are listed in Table 5.3. Notably, both datasets exhibit considerable imbalance, with

normal/abnormal ratios of 34%/66% and 81%/19%, respectively. This ratio reflects the proportion of benign (normal) traffic to malicious (abnormal) traffic within the datasets. A higher percentage of abnormal traffic in CICIDS2017 (66%) and normal traffic in UNSW-NB15 (81%) demonstrates the inherent variability in the datasets, which is crucial for evaluating the robustness of the ML models on imbalanced datasets.

Since anomaly detection systems aim to distinguish cyber-attacks from normal states, dataset instances are treated as binary, with two labels: *benign* or *malicious*. For model evaluation, we employ both hold-out and prequential validations. In the hold-out evaluation, the initial model training uses the first 10% of the data, whereas the remaining 90% is used for testing. In prequential validation, also known as test-and-train validation, batches of instances (i.e., segments) in the inference test set are first used to test the learning model, and subsequently employed for model retraining and updating.

5.4.2 Features and ML algorithm selection

Given the constraints of the PDP, extracting a minimal set of features from network packets is crucial without requiring complex operations. Therefore, a feature extraction and selection module is employed. Table 5.4 lists selected features for both datasets. Five tuples (i.e., IPs, ports, and protocol) from each packet are used to identify the flow. The flow-level features for each unique flow are determined by calculating the aggregate summary: the maximum, minimum, and mean packet sizes. Since the relevance of features may change due to concept drift, we use explainability techniques following each model retraining to obtain the relevance information of the features. The SHAPley value-based explainability method (Sundararajan and Najmi, 2020) identifies and selects the top K features.

Table 5.4 Features definition

Feature Type		Description
Per-packet	-	Packet size, protocol, etc.
Flow-level	Aggregate	F = aggr(a, c, d)
	Summary	max, min, and mean, etc.

This explainability-driven feature selection approach enables the data plane to choose a minimal yet highly relevant set of features for online inference.

Since the relevance of features may change due to concept drift, we use explainability techniques after each model retraining to assess and retain only the most relevant features. The SHAPley value-based explainability method (Sundararajan and Najmi, 2020) identifies the top K features that contribute most to classification accuracy, ensuring that important flow characteristics remain included in the decision-making process. This explainability-driven feature selection approach enables the data plane to continuously adapt its feature set while maintaining a lightweight yet effective classification mechanism.

Although the feature such as source ports is often randomized, feature importance analysis (see Fig. 5.6) shows that they exhibit measurable relevance in certain traffic segments, particularly in attack patterns. This suggests that, in specific cases, automated attack tools or misconfigurations lead to non-random port usage.

The choice of an ML model depends on its deployability within a PDP. Although various supervised learning approaches exist for traffic characterization, not all are compatible with the implementation in P4 (Xavier et al., 2021). We aim to seamlessly integrate the ML model into the data plane, which requires alignment with the available operations in P4. Considering the current primitives in P4 (Bosshart, Daly, et al., 2014), a *Decision Tree Classifier (DTC)* emerges as the optimal choice for this task (Xiong and Zilberman, 2019; Saqib et al., 2022). The classification process of the DTC, involving comparison operations for element x,

aligns seamlessly with the *MATs* of the PDP.

5.4.3 Unsupervised Labeling

To autonomously classify network traffic, we employ k-Means clustering (Sinaga and Yang, 2020) and Isolation Forest (iForest) (F. T. Liu et al., 2008) as unsupervised learning techniques to identify malicious and benign instances from unseen traffic patterns. k-Means partitions traffic based on similarity, assuming that benign and attack traffic exhibit distinct clustering characteristics. iForest, on the other hand, detects anomalies based on the isolation of sparse patterns typically associated with attack traffic. The choice of unsupervised learning is motivated by the need to operate without labeled data, ensuring adaptability to evolving attack patterns. We evaluate both models across different datasets and analyze their efficacy in correctly identifying labels before the retraining processes.

5.4.4 Experimental setup

We structured our experiments in three steps. First, we detail our simulation setup in Section 5.4.3.1. Next, we explain the ML model training and deployment in the data plane in Section 5.4.3.2. Finally, we describe the performance measures for in-network attack detection in Section 5.4.3.3.

5.4.4.1 Simulation setup

The logical components of the simulation setup are shown in Fig. 5.3. The *measurement component* (on the left) generates, collects, and analyzes the network traffic. The *data plane component* (on the right) serves as the focal point of evaluation, implemented in P4 and compiled with the behavioral model version 2 (BMv2).

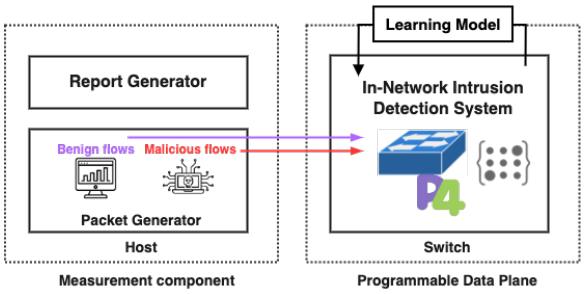


Figure 5.3 Simulatiing data plane

The structure of the simulation setup is as follows:

- The host uses the Python library *dpkt*[1] to send packets and receives them back with classification and timestamps. This component reports the classification results.
- The switch contains a traffic classification algorithm responsible for detecting anomalies and either forwarding or discarding packets.
- The learning module atop the data plane manages the ML model training and interpretation process on the provided data and is implemented in the control counterpart of the network.

5.4.4.2 Model training & deployment

In order to seamlessly integrate ML model with PDP, we require tools that support both model training and the interpretation of learned rules for real-time inference. We use the *scikit-learn*[1] implementation in Python for the ML model training. The learned rules of the *DTC* classifier are extracted and subsequently translated into

[1]https://pypi.org/project/dpkt/

[1]https://scikit-learn.org/

MATs as *MA* rules using the P4Runtime API. Each *MAT* corresponds to a single feature, with the total number of tables equal to the number of input features plus a class table.

Before applying the *MATs*, the P4 code extracts the port numbers and packet size from sequential packets of each flow and stores them in the memory registers in *SRAM*. Once the threshold is reached (i.e., the minimum number of packets required for a classification decision), the switch generates a summary of the input feature values. The extracted feature values are then processed through *MATs* to determine the traffic class.

5.4.4.3 Performance measures

The evaluation aimed to efficaciously address the detection and adaptation of concept drift with minimal disturbance to normal network traffic. As concept drift represents emerging attacks in our case, our goal is to enable a continuous learning approach to quickly detect and mitigate malicious traffic.

Quick detection and adaptation of concept drifts are desirable. However, performing such operations for every instance is infeasible in a pragmatic framework. ADaptive WINdowing (ADWIN) and Drift Detection Method (DDM) are two common drift detection techniques (Lu et al., 2018). ADWIN is a distribution-based method that effectively addresses gradual drift by increasing window size. On the other hand, DDM is a model performance-based method that defines warning and drift level thresholds to monitor the model error rate and standard deviation change for detection. Although the DDM can identify sudden drifts, its response time is often slow for gradual drifts. Motivated by the effectiveness of ADWIN, we implement an adaptive windowing-enabled DDM to monitor the accuracy of the collection of flows with a dynamic window size and then verify the drift by measuring the

change in data distribution. Hence, we opted to use an ADWIN-enabled DDM method to maintain the efficacy of drift detection and adaptation in programmable networks by preserving detection accuracy and controlling adaptation impact.

In our proposed framework, the ADWIN-enabled DDM continuously monitors the model predictions and adjusts the window size W (i.e., the batch of flows to be monitored) based on the current inference result. For instance, when the model undergoes drifts, W becomes smaller to adapt to new patterns quickly, while in the absence of drift, the window becomes larger to reduce the impact of adaptation.

Several performance metrics are used to evaluate the framework. One set of measures relates to drift detection accuracy, with two key metrics: True Positive Rate (TPR) and True Negative Rate (X. Zhang et al., 2021). TPR is the ratio of correctly classified positive samples to the total number of positive samples, while TNR is the ratio of correctly classified negative samples to the total number of negative samples. Hence, TPR and TNR represent the percentage of malicious flows that were correctly predicted and the percentage of benign flows that were correctly predicted, respectively.

$$TPR = \frac{TP}{TP + FN}$$
$$TNR = \frac{TN}{TN + FP}$$

The other set of metrics concerns the impact of drift adaptation, which is represented as packet loss caused by model retraining. Because the ML model is implemented in the form of *MA* rules in the *MATs* of the PDP, retraining causes model remapping to the existing *MATs*. In other words, when the control plane generates a fresh set of rules, these rules are written to the data plane to form new inference thresholds and complete the updating process. Such an update disturbs normal traffic processing of the network. Therefore, the framework should maintain the efficacy of concept

drift detection with a minimal adaptation impact. Hence, we defined the model retraining and packet loss rates as metrics to be minimized, with the former directly reflecting the latter.

Both sets of metrics are directly reflected by the monitoring window size W. A smaller W may result in quick drift detection but will lead to more frequent updates in the network. Conversely, a larger W will reduce the adaptation impact but might not efficaciously address the drift. Hence, maintaining the efficacy of drift detection and minimizing the effects of adaptation are equally important.

5.5 Experimental results

The experimental results are presented in two parts. First, we validate the existence of drifts and their implications in Section 5.5.1. Then, we evaluate the efficacy of the proposed data-driven method on unseen data in Section 5.5.2.

5.5.1 Validating drift in the datasets

The datasets selected for the analysis include evolving attack traffic patterns generated at different time intervals. We visualized the timelines of the attacking traffic instances from both datasets. Fig. 5.4 shows the emergence and evolution of attack patterns over time for CICIDS2017 on the left and and UNSW-NB15 on the right. The *x-axis* represents the timelines, and the *y-axis* denotes the attack type. Notably, benign traffic is consistently present, whereas various attacks are introduced at distinct points in time.

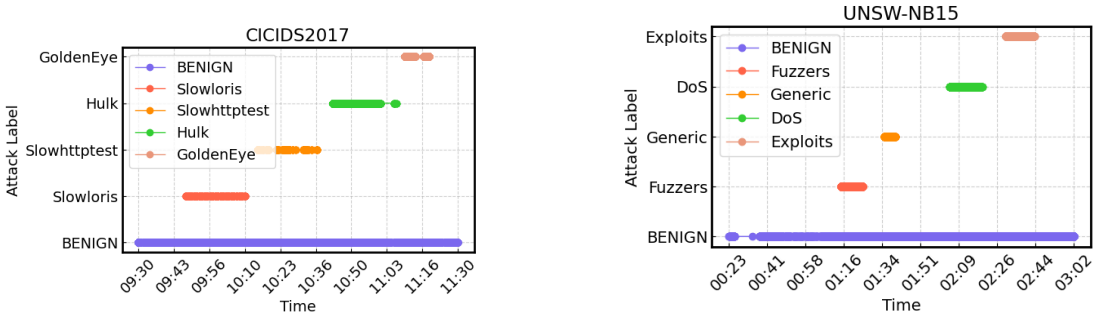


Figure 5.4 Attacking traffic timelines

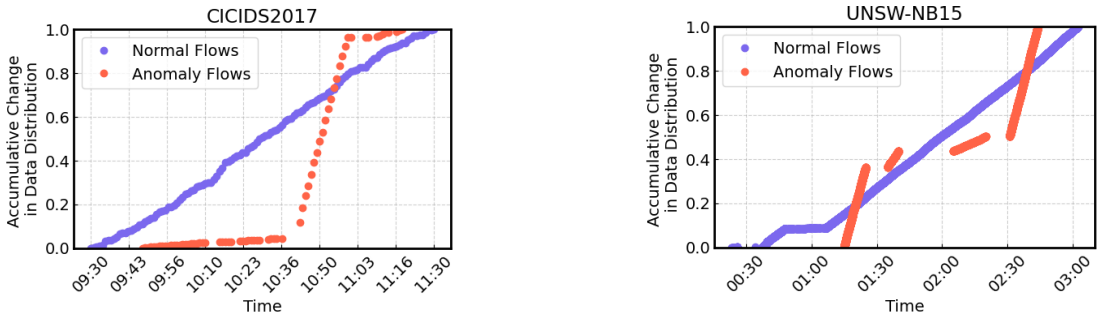


Figure 5.5 Effect of emerging traffic patterns on change in data distribution

5.5.1.1 Change in data distribution

We further examined variations in the traffic patterns that deviate from the norm. Attackers attempt to create traffic with diverse characteristics, such as using different ports and introducing randomness in packet sizes or inter-arrival times. The variations in traffic characteristics lead to changes in the distribution of malicious traffic and the traffic patterns become unknown to the existing ML model.

To better understand these deviations, we calculate the cumulative change for all features of benign and malicious traffic flows to investigate how the data distribution of anomalous flows deviates from normal flows. The sub-figures in Fig.

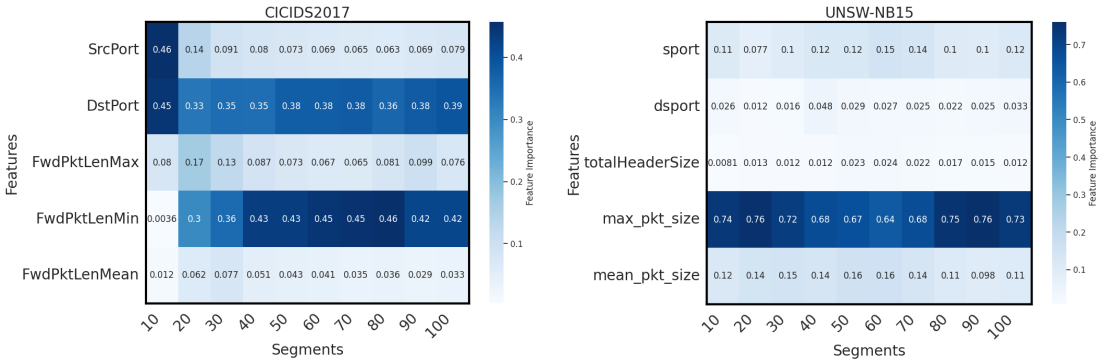


Figure 5.6 Effect of emerging traffic patterns on change in features relevance

5.5 show the cumulative change in the data distribution for the chosen datasets, CICIDS2017 on the left and UNSW-NB15 on the right. The *x-axis* represents the timeline, and *y-axis* shows the cumulative change within a range of 0 to 1. Notably, the distribution of the benign traffic exhibits a linear change while maintaining consistent characteristics. In contrast, the distribution of malicious traffic differs, occasionally showing an exponential increase and, at other times, a slower progression.

5.5.1.2 Change in features relevance

We extended our analysis to investigate the impact of emerging attacks on the relevance of input features. We divide the segmented datasets into ten subsets and conduct sequential testing by incorporating each subset into the existing model. The results of our analysis reveal the shifting dynamics in feature relevance across the test sets, as depicted in Fig. 5.6.

Notably, for CICIDS2017 (left), a marked and dynamic change in feature relevance is observed, indicating the evolving nature of the attack patterns. In contrast, UNSW-NB15 (right) exhibits subtle variations in feature relevance over time. The findings underscore the dynamic nature of feature importance, suggesting that

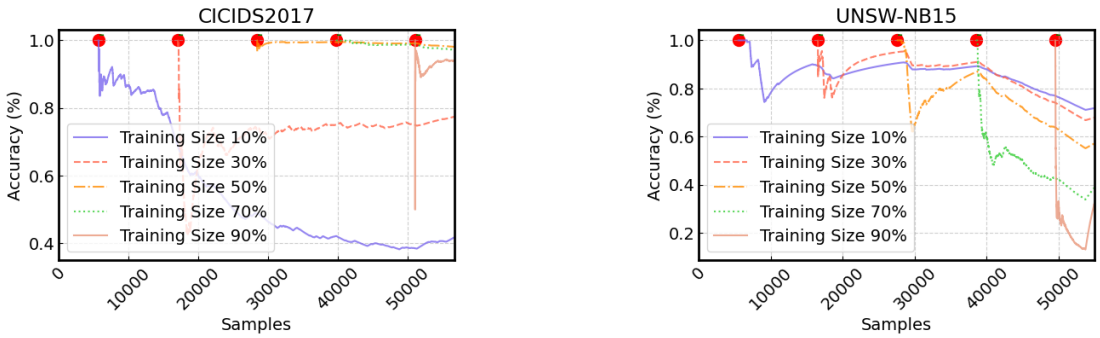


Figure 5.7 Accuracy over varying training sets

features may undergo changes in significance over time in certain cases.

5.5.1.3 Change in ML model accuracy

Fluctuations in feature relevance and their classification decision boundaries, caused by changes in data distribution, may lead to incorrect predictions by the ML model, thereby degrading its performance over time.

The performance degradation of the ML model is validated by varying the size of the training set from 10% to 90%. The decrease in accuracy across different training sets is shown in Fig. 5.7. The red dots indicate the instances from which the test set begins. As shown, the accuracy of the testing instances for the initial training set (i.e., 10% and 30%) decreases significantly for CICIDS2017 (on the left). In contrast, it deviates considerably for UNSW-NB15 (on the right). In the former case, the emerging patterns differ markedly from the existing ones, whereas the latter exhibits some relevance to the existing patterns. Over the subsequent training sets, the accuracy of the test sets improves as more anomalous patterns become known to the trained model. Therefore, the performance degradation of the model is directly reflected by the change in the relevance of the features and their classification decision boundaries when new traffic patterns emerge. Such

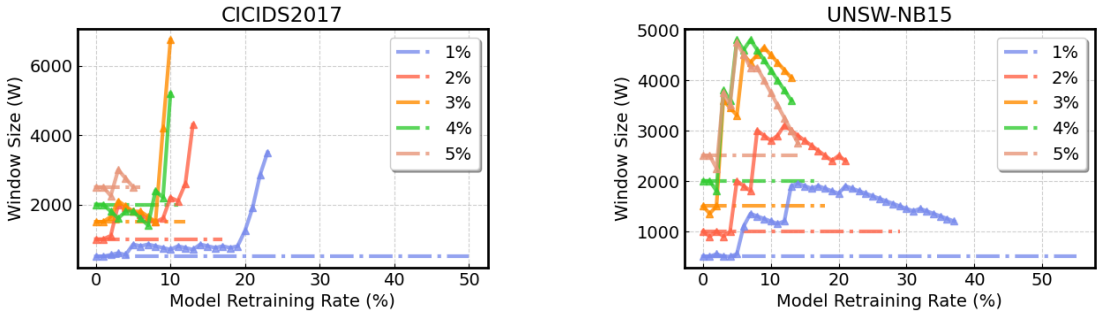


Figure 5.8 Effect of window size (W) on ML model retraining rate.

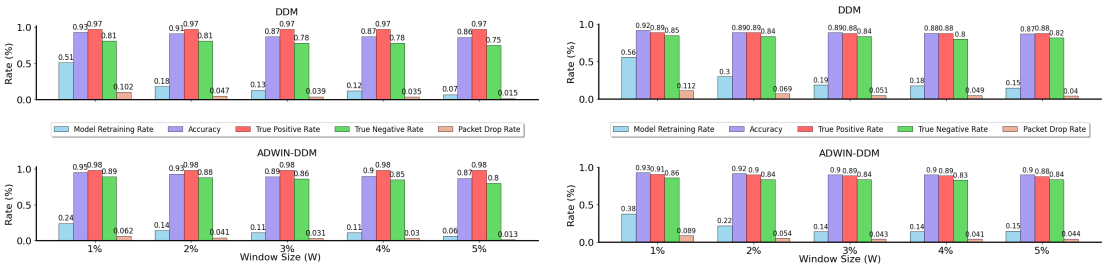


Figure 5.9 Performance measures: CICIDS2017 (left) and UNSW-NB15 (right).

results underscore the importance of introducing adaptivity to a single-fit ML model to sustain accuracy during evolving attacks. Additionally, this highlights the significance of employing drift detection and adaptation strategies to fortify the resilience of network intrusion detection systems.

5.5.2 Evaluating the effectiveness of data-driven approach

In this subsection, we evaluate the efficacy of the proposed data-driven method by demonstrating its ability to identify and mitigate emerging malicious traffic flow. To achieve this, we adopt drift detection thresholds based on the current performance of ML model. We then analyze the impact of varying monitoring window sizes on the efficacy of drift detection and its adaptation impact. Our results show that the proposed approach efficaciously detects and mitigates evolving malicious traffic.

5.5.2.1 Impact of varying window size

The term *window size (W)* refers to a batch of traffic flows representing the monitoring frequency in our pragmatic framework. Monitoring every instance is impractical and can result in a significant network overhead. A smaller W makes the system more aggressive in monitoring model performance, whereas a larger W reduces the monitoring frequency.

The efficacy of our design lies in using an adaptive windowing-enabled drift detection method, the ADWIN-DDM. We highlight the potential of ADWIN-DDM compared to DDM with static W. The subfigures in Fig. 5.8 depict the effect of W on the model retraining rate for the chosen datasets. The *x-axis* represents the size of W, and the *y-axis* represents the model retraining rate. The simulation is performed using the selected portion of the datasets, which is divided into 10% for training and 90% for testing. The initially trained ML model incrementally adapts incoming flow instances in batches of size W. The dashed-dotted lines represent the DDM with W, whereas the solid lines with arrow markers represent the DDM with dynamic W. Various starting sizes of W (ranging from 1% to 5% of the total flow instances) are represented by different colors proportionate to the chosen datasets.

The results show that the impact of the DDM with static W is directly proportional to the model retraining rate for both datasets. In contrast, DDM with an adaptive window size exhibits a different behavior. The solid lines demonstrate the dynamic adjustments of W based on the model performance. In the case of static W, a smaller window leads to more frequent adaptation, whereas a larger W slows adaptation. However, the dynamic W is adjusted based on the nature of the data and model performance. When the model performance decreases, W becomes more aggressive to address drift quickly. When the model performance improves, W becomes larger to avoid unnecessary updates and reduce or eliminate the negative

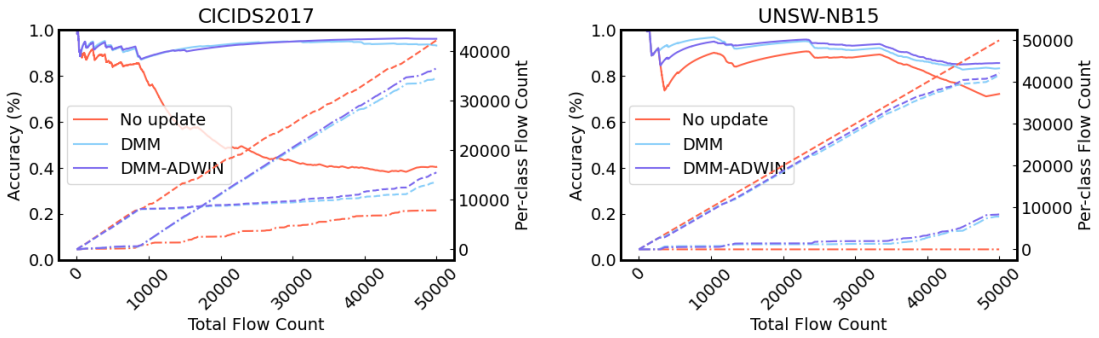


Figure 5.10 Data plane results

impact of adaptation. The variation generally shows similar trends regardless of the starting size, indicating an adjustment to W based on the nature of the data and model performance.

Overall, the adjustment of W is found to be more effective in capturing the characteristics of the unseen data from both datasets. In the case of the CICIDS2017 dataset, the size of W initially fluctuates but then increases considerably as the accuracy of the model improves, leading to a decrease in the retraining rate. In contrast, the unseen traffic patterns in the UNSW-NB15 dataset are largely unknown to the trained ML model, resulting in significant fluctuations in the size of W. Therefore, the adjustment of W is made based on the nature of the unseen data throughout the simulation, resulting in a better capture of unseen patterns with minimal adaptation impact.

Fig. 5.9 provides a more precise view of the effect of W on the drift detection accuracy and its adaptation impact for both datasets: CICIDS2017 on the left and UNSW-NB15 on the right. The grouped bars on the *x-axis* represent the performance metrics, and the *y-axis* represents the results in percentages. The first metric, the model retraining rate, is proportional to the initial window sizes in both cases (DDM with constant and dynamic W). The retraining rate directly influences the detection accuracy, as more frequent model adaptation of new patterns allows

quicker identification of emerging traffic patterns. However, this comes at a cost represented by the packet drop rate, which is directly affected by the model retraining rate. Hence, there is a trade-off between obtaining the classification accuracy and minimizing the model retraining impact, which is indicated in both cases for both datasets.

The results unequivocally demonstrate the efficacy of using a dynamically adjusted W over a static W. For CICIDS2017, the model retraining rate is significantly reduced, and the drift detection accuracy increases slightly when W started with 1% of the given instances. The subsequent sizes of W exhibit similar trends, with a trade-off between the accuracy and packet drop rate. The results for UNSW-NB15 on the right show similar trends over the complete simulation but with different rates due to the adjustment of W depending on the unseen data. Overall, the performance results are directly reflected by W in all cases for both datasets, indicating a negative correlation between W and model retraining. The ML model retraining has a positive impact on the efficacy of drift detection but a negative impact on drift adaptation. Therefore, the data-driven approach allows for the dynamic adjustment of W based on the nature of incoming data, which better manages the trade-off by maintaining accuracy while minimizing the impact of adaptation.

5.5.2.2 Mitigating evolving malicious traffic

The efficacy of the proposed approach in mitigating evolving malicious traffic is illustrated in Fig. 5.10. The 1^{st} *y-axis* on the left represents the average accuracy, and the 2^nd *y-axis* on the right depicts the number of predictions for each class, that is, the count of benign and malicious flows. The simulation runs over the chosen portion of datasets, with 10% of the data for training and 90% for testing,

initializes W at 1% of the total data. The subfigures display the results from the selected datasets using three cases distinguished by color (red, cyan, and blue): no adaptation, i.e., baseline; adaptation using static W (i..e, DDM); and adaptation using dynamic W (i.e., ADWIN-DDM). The solid lines represent the average accuracy, the dashed-dotted line represents the number of positive predictions (malicious flows), and the single dashed line indicates the number of negative predictions (benign flows).

The subfigure on the left for CICIDS2017 reveals that the average accuracy continuously decreases to 40% when the model was not retrained. Such a decrease is evident in the prediction counts, where malicious flows remain above 10,000, whereas the count of benign flows exceeds 40,000. By contrast, both model adaptation cases demonstrate how the model maintains accuracy by correctly identifying malicious and benign flows. In both cases, the model accurately identifies anomalies when new patterns emerge. The subfigure on the right presents the results for the UNSW-NB15. The average accuracy of the model without adaptation significantly decreases. In the adaptation cases, the accuracy is maintained by correctly mapping instances to their respective classes. However, there is a variation in the obtained accuracy attributed to the dissimilarities of the new patterns with the existing ones. Additionally, from both datasets, the precedence of ADWIN-DDM over DDM is indicated in obtaining better accuracy by more precisely identifying malicious traffic, facilitated by its dynamic adjustment to W based on model performance. In other words, ADWIN-DMM achieves better classification accuracy by efficaciously capturing and adapting the evolving shifts in malicious traffic patterns.

5.6 Conclusion

This paper introduces an adaptive in-network defense method designed to protect networks from evolving volumetric attacks. We employ a continuous learning approach that leverages a data-driven method to establish baseline performance thresholds from historical data, using these thresholds as benchmarks for detecting anomalies in unseen data. By continuously updating these thresholds to reflect changes in data distribution and in-network inference results, our method efficaciously adapts to evolving attack patterns.

We validate the presence of drift in intrusion detection datasets and demonstrate its effect on model performance. Our evaluation shows that the adaptive nature of the ML model maintains classification accuracy despite evolving attacks. We also compare static and dynamic monitoring window sizes, finding that dynamic windows achieve better accuracy while reducing disruptions to normal network traffic.

Our findings reveal a trade-off between drift detection accuracy and the effects of model adaptation. The adaptive adjustment of the monitoring window size helps manage this trade-off, providing a more responsive solution for detecting and adapting to new attack patterns.

REFERENCES

Amin, Muhammad et al. (2023). "Cyber security and beyond: Detecting malware and concept drift in AI-based sensor data streams using statistical techniques". In: *Computers and Electrical Engineering* 108, p. 108702.

Barradas, Diogo et al. (2021). "FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications." In: *NDSS*.

Bosshart, Pat, Dan Daly, et al. (2014). "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3, pp. 87–95.

Bosshart, Pat, Glen Gibb, et al. (2013). "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN". In: *ACM SIGCOMM Computer Communication Review* 43.4, pp. 99–110.

Chen, Xiang, Hongyan Liu, et al. (2022). "Empowering DDoS Attack Mitigation with Programmable Switches". In: *IEEE Network*.

Chen, Xiang, Chunming Wu, et al. (2023). "Empowering Network Security with Programmable Switches: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials*.

Coelho, Bruno and Alberto Schaeffer-Filho (2022). "BACKORDERS: using random forests to detect DDoS attacks in programmable data planes". In: *Proceedings of the 5th International Workshop on P4 in Europe*, pp. 1–7.

Ding, Damu et al. (2021). "In-network volumetric DDoS victim identification using programmable commodity switches". In: *IEEE Transactions on Network and Service Management* 18.2, pp. 1191–1202.

Doshi, Rohan, Noah Apthorpe, and Nick Feamster (2018). "Machine learning ddos detection for consumer internet of things devices". In: *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, pp. 29–35.

Friday, Kurt, Elias Bou-Harb, and Jorge Crichigno (2022). "A learning methodology for line-rate ransomware mitigation with p4 switches". In: *International Conference on Network and System Security*. Springer, pp. 120–139.

Friday, Kurt, Elie Kfoury, et al. (2022). "Inc: In-network classification of botnet propagation at line rate". In: *European Symposium on Research in Computer Security*. Springer, pp. 551–569.

Gama, Joao et al. (2004). "Learning with drift detection". In: *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer, pp. 286–295.

Gu, Feng (2019). "Concept drift detection for machine learning with stream data". PhD thesis.

Hireche, Othmane, Chafika Benzaıd, and Tarik Taleb (2022). "Deep data plane programming and AI for zero-trust self-driven networking in beyond 5G". In: *Computer Networks* 203, p. 108668.

Khamassi, Imen et al. (2018). "Discussion and review on evolving data streams and concept drift adapting". In: *Evolving systems* 9, pp. 1–23.

Kuppa, Aditya and Nhien-An Le-Khac (2022). "Learn to adapt: Robust drift detection in security domain". In: *Computers and Electrical Engineering* 102, p. 108239.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou (2008). "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE, pp. 413–422.

Liu, Zaoxing et al. (2021). "Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches". In: *30th USENIX Security Symposium (USENIX Security 21)*, pp. 3829–3846.

Lu, Jie et al. (2018). "Learning under concept drift: A review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12, pp. 2346–2363.

Moustafa, Nour and Jill Slay (2015). "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 military communications and information systems conference (MilCIS)*. IEEE, pp. 1–6.

Musumeci, Francesco et al. (2022). "Machine-learning-enabled ddos attacks detection in p4 programmable networks". In: *Journal of Network and Systems Management* 30, pp. 1–27.

Pacheco, Fannia et al. (2018). "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey". In: *IEEE Communications Surveys & Tutorials* 21.2, pp. 1988–2014.

Saqib, Muhammad et al. (2022). "An Accurate & Efficient Approach for Traffic Classification Inside Programmable Data Plane". In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, pp. 6331–6336.

Sharafaldin, Iman, Arash Habibi Lashkari, and Ali A Ghorbani (2018). "Toward generating a new intrusion detection dataset and intrusion traffic characterization." In: *ICISSp* 1, pp. 108–116.

Sharma, Naveen Kr et al. (2017). "Evaluating the Power of Flexible Packet Processing for Network Resource Allocation." In: *NSDI*, pp. 67–82.

Sinaga, Kristina P and Miin-Shen Yang (2020). "Unsupervised K-means clustering algorithm". In: *IEEE access* 8, pp. 80716–80727.

Sundararajan, Mukund and Amir Najmi (2020). "The many Shapley values for model explanation". In: *International conference on machine learning*. PMLR, pp. 9269–9278.

Tan, Chang How, Vincent CS Lee, and Mahsa Salehi (2020). "Mir_mad: An efficient and on-line approach for anomaly detection in dynamic data stream". In: *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE, pp. 424–431.

Tgavalekos, Karl, Josephine M Namayanja, and Rasheed Alhassan (2018). "Characterization of network behavior to detect changes: a cybersecurity perspective". In: *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, pp. 1–6.

Wang, Lidong and Randy Jones (2021). “Big data analytics in cyber security: network traffic and attacks”. In: *Journal of Computer Information Systems* 61.5, pp. 410–417.

Xavier, Bruno Missi et al. (2021). “Programmable switches for in-networking classification”. In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, pp. 1–10.

– (2022). “MAP4: A Pragmatic Framework for In-Network Machine Learning Traffic Classification”. In: *IEEE Transactions on Network and Service Management*.

Xie, Guorui et al. (2023). “Empowering In-Network Classification in Programmable Switches by Binary Decision Tree and Knowledge Distillation”. In: *IEEE/ACM Transactions on Networking*.

Xiong, Zhaoqi and Noa Zilberman (2019). “Do switches dream of machine learning? toward in-network classification”. In: *Proceedings of the 18th ACM workshop on hot topics in networks*, pp. 25–33.

Zhang, Menghao et al. (2020). “Poseidon: Mitigating volumetric ddos attacks with programmable switches”. In: *the 27th Network and Distributed System Security Symposium (NDSS 2020)*.

Zhang, Xiaoquan et al. (2021). “pHeavy: Predicting heavy flows in the programmable data plane”. In: *IEEE Transactions on Network and Service Management* 18.4, pp. 4353–4364.

Zheng, Changgang, Xinpeng Hong, et al. (2023). “In-Network Machine Learning Using Programmable Network Devices: A Survey”. In: *IEEE Communications Surveys & Tutorials*.

Zheng, Changgang, Zhaoqi Xiong, et al. (2024). "IIsy: Hybrid In-Network Classification Using Programmable Switches". In: *IEEE/ACM Transactions on Networking*.

Zhou, Guangmeng et al. (2023). "An efficient design of intelligent network data plane". In: *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 6203–6220.

CHAPTER VI

CONCLUSIONS ET ORIENTATIONS FUTURES

6.1 Conclusions

Cette thèse présente des solutions architecturales et algorithmiques au problème de la gestion du trafic hétérogène dans les NGN, avec pour principaux objectifs l'amélioration de la qualité de service et le maintien de la sécurité du réseau. Un examen systématique de la littérature pertinente a révélé la complexité de la gestion du trafic hétérogène dans les NGN, en raison des facteurs suivants :

- Il n'existe actuellement aucun cadre global capable de gérer intelligemment les demandes de trafic hétérogène en intégrant la classification du trafic avec un mécanisme de fourniture de qualité de service tenant compte des objectifs de service ;
- Les implications d'une mauvaise classification du trafic sur la performance du réseau et la rentabilité du fournisseur de services n'ont pas été étudiées ;
- Les classificateurs de trafic actuels basés sur l'apprentissage dans le réseau s'appuient souvent sur des méthodes uniques, qui ne parviennent pas à s'adapter à l'évolution des modèles de trafic.

Dans les réseaux modernes, une gestion efficace du trafic implique de relever

plusieurs défis, notamment la classification du trafic à un stade précoce, l'approvisionnement en qualité de service sans état et le maintien des performances dans un contexte d'évolution du trafic. La nature dynamique du trafic réseau, induite par les nouvelles applications et les menaces émergentes, exige des solutions intelligentes et adaptatives. Nous pensons que la combinaison de la programmabilité du réseau avec la ML est essentielle pour la gestion du réseau à grande échelle, offrant l'adaptabilité, la résilience et l'efficacité requises.

Cette thèse présente trois contributions clés, chacune se concentrant sur un aspect distinct de la gestion du trafic dans les réseaux modernes : la classification du trafic à un stade précoce pour un approvisionnement efficace en qualité de service, des mécanismes basés sur l'apprentissage adaptatif pour maintenir la qualité de service, et des méthodes pour protéger le réseau contre les menaces en constante évolution. Ensemble, ces contributions apportent des solutions pratiques aux défis pressants de l'allocation des ressources, de l'atténuation des menaces et du maintien des performances des réseaux de prochaine génération.

La première contribution, présentée dans l'article intitulé **An Intelligent and Programmable Data Plane for QoS-Aware Packet Processing**, se concentre sur la classification du trafic à un stade précoce et sur l'approvisionnement en qualité de service sans état. Ces travaux soulignent la nécessité d'un cadre intégré capable de prendre des décisions en temps réel lorsque le trafic pénètre dans le réseau. En s'appuyant sur des plans de données programmables, cette recherche introduit un algorithme de classification léger qui intègre des objectifs de qualité de service directement dans les en-têtes des paquets, ce qui permet un traitement des paquets efficace et tenant compte de la qualité de service sans nécessiter un suivi de l'état spécifique au flux à chaque nœud du réseau. Les résultats de cette étude démontrent qu'il est possible d'atteindre une grande efficacité et une faible surcharge dans la gestion du trafic hétérogène, contribuant ainsi à l'objectif

de la thèse, à savoir l'utilisation efficace des ressources du réseau. La deuxième contribution, détaillée dans l'article **Adaptive In-Network Traffic Classifier: Bridging the Gap for Improved QoS by Minimizing Misclassification**, aborde les limites des modèles de classification statiques. Ce travail formule le problème de l'allocation des ressources pour le trafic multi-prioritaire dans un réseau à chemins multiples comme un problème d'optimisation pour cartographier de manière optimale les flux de trafic vers les chemins appropriés du réseau. Il propose un classificateur de trafic adaptatif basé sur le ML qui évolue en fonction des modèles de trafic changeants. Le mécanisme d'apprentissage adaptatif garantit une classification précise du trafic, minimise les erreurs de classification et maintient ainsi la qualité de service. En développant un classificateur intelligent qui s'adapte à la nature dynamique du trafic, cette recherche contribue à maintenir la qualité de service pour le trafic multi-prioritaire dans le réseau.

La troisième contribution, présentée dans l'article **A Data-Driven Approach to Mitigate Evolving Volumetric Attacks in Programmable Networks**, aborde les défis de sécurité auxquels sont confrontés les NGN. Ce travail développe une approche de classification adaptative du trafic pour détecter et atténuer les attaques volumétriques en temps réel. En utilisant des plans de données programmables et la ML, la recherche propose une méthode pour automatiser la détection des dérives et adapter les modèles ML de manière transparente sans perturber les opérations normales du réseau. Des évaluations expérimentales ont confirmé l'efficacité de la méthode dans la défense contre les attaques volumétriques en évolution, remplissant ainsi l'objectif de sécurité de la thèse.

En résumé, ces trois contributions abordent collectivement les principaux défis de la gestion du trafic dans les réseaux de prochaine génération : l'utilisation efficace des ressources, l'adaptabilité pour maintenir la qualité de service et la sécurité. En tirant parti des plans de données programmables et de la ML adaptative, le cadre

proposé garantit une classification efficace du trafic, une allocation dynamique des ressources et une atténuation robuste des menaces. Cette thèse démontre le potentiel de l'intégration de ces techniques avancées pour fournir une solution complète permettant d'optimiser l'allocation des ressources, d'assurer la qualité de service et de maintenir la sécurité dans les réseaux de prochaine génération.

6.2 Orientations futures de la recherche

Certaines approches proposées dans cette thèse apportent des avancées importantes tout en offrant des possibilités d'approfondissement, conduisant à de nouvelles directions de recherche, comme détaillé ci-dessous :

6.2.1 Mise en œuvre matérielle

Outre les performances supérieures de la solution proposée dans le réseau pour gérer les demandes de trafic hétérogène, le cadre peut être efficacement mis en œuvre dans le matériel connu pour son débit élevé (jusqu'à 12,8 Tbps) et sa latence au niveau de la microseconde (Intel, 2024), dépassant de loin les capacités de BMv2. Le BMv2 est un outil de développement, de test et de débogage des plans de données P4, et non un commutateur logiciel de niveau de production. Par conséquent, les performances de BMv2 en matière de débit et de latence sont nettement inférieures à celles d'un matériel de qualité.

En particulier, le chargement des programmes P4 pour les tâches mentionnées ci-dessus modifierait peu le débit, c'est-à-dire le débit de réception (Rx) et le débit de transmission (Tx) du commutateur. Le délai de traitement des paquets devrait rester proche de la performance de base de l'appareil car aucune opération complexe n'est impliquée dans la logique de classification et de fourniture de la qualité de service (Xavier et al., 2021). L'en-tête de qualité de service supplémentaire peut

diminuer le débit et augmenter le délai de transmission sur chaque liaison de sortie. Toutefois, cette surcharge est généralement constante et typiquement négligeable lors de l'utilisation de liaisons de sortie à Gbps de commutation (Turkovic et al., 2021).

6.2.2 Menaces potentielles de l'intégration des objectifs de qualité de service dans l'en-tête des paquets

L'intégration d'objectifs de qualité de service ou d'objectifs de niveau de service dans les en-têtes de paquets introduit des menaces potentielles pour la sécurité. Des acteurs malveillants pourraient tenter de manipuler ou de falsifier les SLO afin d'obtenir un accès prioritaire non autorisé aux ressources du réseau ou de perturber les protocoles de gestion du trafic, ce qui entraînerait une allocation inéquitable de la bande passante ou une congestion intentionnelle du réseau.

Pour atténuer ces risques, il est essentiel de mettre en œuvre des mécanismes de vérification des SLO et des contrôles d'intégrité des paquets. Ces mécanismes garantissent que seules les entités autorisées peuvent modifier ou interpréter les données SLO intégrées dans les paquets, empêchant ainsi toute falsification non autorisée. En outre, l'incorporation de techniques cryptographiques telles que les signatures numériques joue un rôle important dans le renforcement de la sécurité. Ces techniques garantissent l'authenticité et l'intégrité des OLS transportés dans les paquets, offrant ainsi une couche de protection supplémentaire.

Malgré des préoccupations valables en matière de sécurité, une conception soignée et le déploiement de mécanismes de vérification et d'intégrité appropriés peuvent minimiser les risques, en garantissant que les avantages de l'intégration d'objectifs de qualité de service dans les paquets l'emportent sur les vulnérabilités potentielles.

6.2.3 QoS pour le trafic hétérogène dans le cadre d'attaques volumétriques évolutives

Une autre direction prometteuse pour la recherche future consiste à évaluer la défense adaptative en réseau proposée dans un cadre global de gestion du trafic qui prend en compte le trafic hétérogène, y compris les applications critiques ayant des exigences strictes en matière de qualité de service. Cette étude pourrait évaluer l'efficacité avec laquelle le système maintient un service ininterrompu pour les applications hautement prioritaires dans le cadre d'attaques volumétriques évolutives, en s'attachant à garantir une perturbation minimale des flux de trafic normaux. Les éléments clés de l'étude comprennent l'évaluation des mesures de qualité de service - telles que la latence, la perte de paquets et le débit - dans le cadre de scénarios d'attaque, ainsi que la capacité du modèle ML à faire la distinction entre les hausses de trafic bénignes et le trafic malveillant.

BIBLIOGRAPHIE

Alberti, Antônio M et al. (2024). "Disruptive 6G architecture: Software-centric, AI-driven, and digital market-based mobile networks". In: *Computer Networks* 252, p. 110682.

Black, D and Paul Jones (2015). *Differentiated services (DiffServ) and real-time communication*. Tech. rep.

Bosshart, Pat et al. (2014). "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3, pp. 87–95.

Chen, Xiang et al. (2022). "Empowering DDoS Attack Mitigation with Programmable Switches". In: *IEEE Network*.

De Alwis, Chamitha et al. (2021). "Survey on 6G frontiers: Trends, applications, requirements, technologies and future research". In: *IEEE Open Journal of the Communications Society* 2, pp. 836–886.

El Rajab, Mirna, Li Yang, and Abdallah Shami (2024). "Zero-touch networks: Towards next-generation network automation". In: *Computer Networks* 243, p. 110294.

Finn, Norman et al. (2019). "Deterministic networking architecture". In: *RFC 8655*.

Hauser, Frederik et al. (2023). "A survey on data plane programming with p4: Fundamentals, advances, and applied research". In: *Journal of Network and Computer Applications* 212, p. 103561.

Intel (2024). *Intel® Tofino™: Intelligent Fabric Processors*. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html. [Online; accessed 04-June-2024].

International Telecommunication Union (2023). *Fast Forward: Internet Traffic 2023 Report*. Accessed: 2024-10-30. URL: https://www.itu.int/itu-d/reports/statistics/2023/10/10/ff23-internet-traffic/.

Karakus, Murat and Arjan Durresi (2017). "Quality of service (QoS) in software defined networking (SDN): A survey". In: *Journal of Network and Computer Applications* 80, pp. 200–218.

Mininet Project (2024). *Mininet: An Instant Virtual Network on your Laptop (or other PC)*. Accessed: 2024-10-30. URL: https://mininet.org/.

P4 Language Consortium (2024). *P4 Behavioral Model Repository*. Accessed: 2024-10-30. URL: https://github.com/p4lang/behavioral-model.

Salman, Ola et al. (2020). "A review on machine learning–based approaches for Internet traffic classification". In: *Annals of Telecommunications* 75.11, pp. 673–710.

Schwarzmann, Susanna et al. (2024). "Native Support of AI Applications in 6G Mobile Networks Via an Intelligent User Plane". In: *2024 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, pp. 1–6.

ShareTechnote (2024). *Handbook LTE QCI (QoS Class Identifier)*. Accessed: 2024-10-30. URL: https://www.sharetechnote.com/html/Handbook_LTE_QCI.html.

Sharma, Naveen Kr et al. (2017). "Evaluating the Power of Flexible Packet Processing for Network Resource Allocation." In: *NSDI*, pp. 67–82.

Shenker, Scott, Craig Partridge, and Roch Guerin (1997). *Specification of guaranteed quality of service*. Tech. rep.

Turkovic, Belma et al. (2021). "P4qos: Qos-based packet processing with p4". In: *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, pp. 216–220.

Xavier, Bruno Missi et al. (2021). "Programmable switches for in-networking classification". In: *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, pp. 1–10.

Xia, Wenfeng et al. (2014). "A survey on software-defined networking". In: *IEEE Communications Surveys & Tutorials* 17.1, pp. 27–51.

Xiong, Zhaoqi and Noa Zilberman (2019). "Do switches dream of machine learning? toward in-network classification". In: *Proceedings of the 18th ACM workshop on hot topics in networks*, pp. 25–33.

Zhang, Shunliang (2019). "An overview of network slicing for 5G". In: *IEEE Wireless Communications* 26.3, pp. 111–117.

Zheng, Changgang et al. (2023). “In-Network Machine Learning Using Programmable Network Devices: A Survey”. In: *IEEE Communications Surveys & Tutorials*.