UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

Suneet Kumar Singh

Advanced Data Plane Techniques for 5G: Hardware-Accelerated Hybrid User Plane Functions and Intelligent Traffic Identification and Prioritization

Técnicas avançadas de plano de dados para 5G: funções de plano de usuário híbrido aceleradas por hardware e identificação e priorização de tráfego inteligente

Campinas

2024

Suneet Kumar Singh

Advanced Data Plane Techniques for 5G: Hardware-Accelerated Hybrid User Plane Functions and Intelligent Traffic Identification and Prioritization

Técnicas avançadas de plano de dados para 5G: funções de plano de usuário híbrido aceleradas por hardware e identificação e priorização de tráfego inteligente

Thesis presented to the Faculty of Electrical and Computer Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering, in the area of Computer Engineering.

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica, na Área de Engenharia de Computação.

Supervisor: Prof. Dr. Christian Rodolfo Esteve Rothenberg

Este exemplar corresponde à versão final da tese defendida pelo aluno Suneet Kumar Singh, e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

_____

Campinas

2024

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

Singh, Suneet Kumar, 1985-
Si64a Advanced data plane techniques for 5g: hardware-accelerated hybrid user plane functions and intelligent traffic identification and prioritization / Suneet Kumar Singh. – Campinas, SP : [s.n.], 2024.

Orientador(es): Christian Rodolfo Esteve Rothenberg.
Tese (doutorado) – Universidade Estadual de Campinas (UNICAMP), Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas de comunicação 5G. 2. Aprendizado de máquina. 3. Jogos em nuvem. I. Esteve Rothenberg, Christian Rodolfo, 1982-. II. Universidade Estadual de Campinas (UNICAMP). Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações complementares

Título em outro idioma: Técnicas avançadas de plano de dados para 5g: funções de plano de usuário híbrido aceleradas por hardware e identificação e priorização de tráfego inteligente
Palavras-chave em inglês:
5G communication systems
Machine learning
Cloud gaming
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora:
Christian Rodolfo Esteve Rothenberg [Orientador]
Jorge Crichigno
Ramon dos Reis Fontes
Waverton Luis da Costa Cordeiro
Leandro Cavalcanti de Almeida
Data de defesa: 01-11-2024
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-7715-3986
- Currículo Lattes do autor: http://lattes.cnpq.br/2128559375052095

COMISSÃO JULGADORA – TESE DE DOUTORADO

Candidato: Suneet Kumar Singh (**RA**: 211573)

Data da Defesa: 01 de novembro de 2024

Título da Tese: "Advanced Data Plane Techniques for 5G: Hardware-Accelerated Hybrid User Plane Functions and Intelligent Traffic Identification and Prioritization".

Título em outro idioma: "Técnicas avançadas de plano de dados para 5G: Funções de plano de usuário híbrido aceleradas por hardware e identificação e priorização de tráfego inteligente".

Prof. Christian Rodolfo Esteve Rothenberg (FEEC/UNICAMP) - Presidente

Prof. Dr. Jorge Crichigno, University of South Carolina

Prof. Dr. Ramon dos Reis Fontes, Instituto Federal da Bahia

Prof. Dr. Weverton Luis da Costa Cordeiro, Instituto de Informática - Universidade Federal do Rio Grande do Sul

Prof. Dr. Leandro Cavalcanti de Almeida, Universidade Federal de São Carlos

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de PósGraduação da Faculdade de Engenharia Elétrica e de Computação.

I dedicate this Ph.D. thesis to my wife, Shalini, who has supported me unconditionally and enabled me to finish this work. I also dedicate it to my little furry buddy Benji, who always makes me smile.

Additionally, I dedicate this work to my mother, the late Mrs. Geeta Singh, in heaven, for her steadfast love and guidance throughout my life. Even though Mom isn't here with me anymore, I can still feel her love leading the way. I value all of her sacrifices, advice, support, and prayers. This thesis is also dedicated to my father, Mr. Ripu Daman Singh, my hero, for his undying love, support, and advice.

I would also like to dedicate this dissertation to Professor Chaodit Aswakul, who gave me excellent guidance and scholarship throughout my first experience conducting research abroad. As a result, I gained confidence and a growing curiosity about research. I also dedicate this thesis to Professor Christian Rothenberg, who encouraged me, believed in me, gave me a great INTRIG research platform, and gave me invaluable advice during this thesis.

Acknowledgements

First and foremost, I want to thank Prof. Dr. Christian Rothenberg, my advisor. Without his creative suggestions, considerate support, and close oversight, my thesis would never have come to be. Thus, I want to thank him for being an excellent teacher and for giving me the confidence to collaborate and work closely with his amazing group of students.

Additionally, I would like to express my gratitude for all of the cooperative efforts made throughout this research project with academic and industry professionals like Fábio Luciano Verdi (Federal University of São Carlos, Brazil), Gianni Antichi (Queen Mary University of London), Carmelo Casciano, Oguz Sunay, and Yi Tseng (Open Networking Foundation), and Gergely Pongrácz and P. Gomes (Ericsson).

I had the pleasure of meeting several incredibly talented people in the INTRIG and LCA groups. I would especially like to thank Danny, Nathan, Raphael, Gyanesh, Fabricio and Francisco for their years of friendship and support.

I would especially like to thank Ericsson Telecomunicações Ltda. for the financial and technical support that allowed me to conduct this research. Through Funcamp/Unicamp intermediation, Ericsson Telecomunicações Ltda. provided funding for this activity initially under grant agreement UNI.61, and lately this study was financed, in part, by the São Paulo Research Foundation (FAPESP), Brazil. Process Number #2021/00199-8, CPE SMARTNESS.

"Excellence is a continuous process and not an accident."

—A.P.J. Abdul Kalam

"The true sign of intelligence is not knowledge but imagination."

—Albert Einstein

RESUMO

As funções de rede são tipicamente executadas em switches de rede e processadores de propósito geral, dependendo dos requisitos das aplicações e os objetivos de flexibilidade e programabilidade. Equipamentos de rede com funcionalidades fixas processam pacotes mais rapidamente. Porém, devido ao lock-in dos fornecedores, mesmo pequenas modificações funcionais podem levar anos para serem implementadas. Por outro lado, CPUs de propósito geral oferecem maior flexibilidade e programabilidade para realizar funções de rede. Ainda assim, software em CPU processa pacotes mais lentamente devido a falhas de cache e ao manuseio de interrupções das interfaces de rede para implementar as funções de entrada e saída. Por conta de essas limitações básicas, como mencionado acima, o desempenho da rede pode ser significativamente impactado por: 1) o uso de servidores baseados em CPU para atividades de rede, o que causa aumento de latência e jitter; 2) a classificação de tráfego em servidores x86 para priorizar fluxos específicos. Esse tipo de classificação em CPU pode afetar o tempo de detecção de fluxos da aplicação, especialmente para aplicações com requisitos estritos de latência. Isso impacta diretamente a Qualidade de Serviço (QoS). Para superar esses problemas, esta dissertação utiliza switches com Plano de Dados Programável (PDP). Os switches PDP permitem que os operadores projetem e implementem métodos de processamento de pacotes em taxa de linha. Esta tese aproveita as funcionalidades do PDP para melhorar a função de plano de usuário (UPF) do núcleo de rede 5G, a parte mais crítica da rede móvel, e atender aplicações críticas que exigem requisitos de desempenho rigorosos. As contribuições consistem em: 1) aceleração e escalabilidade do UPF 5G por meio de (a) descarregamento do UPF para hardware programável e (b) divisão das funcionalidades do UPF entre dispositivos programáveis, denominada UPF híbrido, com base em características específicas do hardware e necessidades da aplicação; 2) classificação de tráfego em hardware programável, especialmente para (a) jogos na nuvem e (b) fluxos elefante (do inglês, *heavy hitter flows*).

Palavras-chaves: 5G, P4, fluxos de grande volume (heavy-hitters), jogos na nuvem.

Abstract

Network functions are typically executed on network switches and general-purpose processors, depending on the requirements of network applications and aiming for particular flexibility and programmability. Fixed functionality network switches process packets faster, but because of vendor lock-in, even minor functional modifications take years to implement. On the other hand, general-purpose CPUs give more flexibility and programmability to carry out network activities. Still, they process packets more slowly due to handling interrupts from network interfaces to process incoming and outgoing packets and cache misses. Because of these fundamental limitations, as noted above, the network's performance can be significantly impacted by: 1) using x86 servers for network activities causes an increase in latency and jitter. 2) classifying traffic on x86 servers to give priority to particular flows. This kind of x86 classification may affect the application's flow detection time, especially for applications with strict latency requirements. This directly affects the Quality of Service (QoS). To overcome these issues, this dissertation leverages Programmable Data Plane (PDP) switches. PDP switches enable operators to design and implement line-rate packet processing methods. This thesis builds on PDP features to improve the 5G core user plane function (UPF), the most critical part of the mobile network, and enables critical applications requiring strict performance requirements. The contributions consist of: 1) 5G UPF acceleration and scalability by (a) offloading UPF to programmable hardware and (b) dividing UPF functionalities among programmable devices, called hybrid-UPF, based on target-specific features and application needs. 2) traffic classification in the switch hardware, especially for (a) cloud gaming and (b) heavy hitter flows.

Keywords: 5G, P4, heavy-hitters, cloud gaming, machine learning.

List of Figures

List of Tables

Acronyms

ACL Access Control List.

API Application Programming Interface.

ARM Advanced RISC Machine.

ASIC Application-Specific Integrated Circuit.

BMv2 Behavioral Model version 2.

BNG Broadband Network Gateway.

BNG-up Broadband Network Gateway user-plane.

CG Cloud Gaming.

COTS Commercial-Off-The-Shelf.

CP Control Plane.

CPU Central Processing Unit.

DCGWs Data Center Gateways.

DCTCP Data Center TCP.

DDoS Distributed Denial-of-Service.

DL Downlink.

DoS Denial-of-Service.

DP Dataplane.

DPDK Data Plane Development Kit.

DT Decision Tree.

DUT Device Under Test.

eBPF Extended Berkeley Packet Filter.

ECMP Equal-cost Multipath.

eCPA enhanced Content Permutation Algorithm.

eNB evolved Node B.

EPG Evolved Packet Gateway.

EWMA Exponential Weighted Moving Average.

FCT Flow Completion Time.

FEC Forwarding Error Correction.

FN False Negative.

FP False Positive.

FPGA Field Programmable Gate Array.

GTP GPRS Tunneling Protocol.

HH Heavy Hitter.

HSS Home Subscriber Server.

HW Hardware.

I-IoT Industrial Internet of Things.

INT In-band Network Telemetry.

IP Internet Protocol.

IPG Inter Packet Gap.

LAT Latency.

LTE Long-Term Evolution.

MAC Media Access Control.

ML Machine Learning.

MME Mobility Management Entity.

MPC Mobile Packet Core.

NFs Network Functions.

NFV Network Functions Virtualization.

NGMN Next-generation Mobile Packet Core.

NIC Network Interface Card.

NPL Network Programming Language.

ODP Open Data Plane.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OSNT Open Source Network Tester.

P4 Programming Protocol Independent Packet Processors.

PCRF Policy and Charging Rules Function.

PDP Programmable Data Plane.

PFCP Packet Forwarding Control Protocol.

PGW Packet Data Network Gateway.

PHV Packet Header Vector.

PS Packet Size.

QCI QoS Class Identifier.

QoE Quality of Experience.

QoS Quality of Service.

RAN Radio Access Network.

SDN Software-Defined Networking.

SGW Serving Gateway.

SIM Simulator.

SLA Service Level Agreement.

SMA Simple Moving Average.

SMF Session Management Function.

SRAM Static Random Access Memory.

SVM Support Vector Machine.

SW Software.

TCAM Ternary Content Addressable Memory.

TEID Tunnel Endpoint Identifier.

TLB Translation Lookaside Buffers.

TN True Negative.

TNA Tofino Native Architecture.

ToR Top-of-rack.

TP True Positive.

TW Time Window.

UD User Devices.

UDM Unified Data Management.

UE User Equipment.

UL Uplink.

UPF User Plane Function.

URLLC Ultra Reliable Low Latency Communications.

vEPG virtual Evolved Packet Gateway.

VNF Virtual Network Function.

VoIP Voice over Internet Protocol.

VR Virtual Reality.

VxLAN Virtual eXtensible LAN.

XDP eXpress Data Path.

XR Extended Reality.

Contents

1 Introduction

The fifth generation of mobile networks is called 5G (GUPTA; JHA, 2015). This new worldwide wireless standard follows 1G, 2G, 3G, and 4G networks. 5G is intended to lower latency, enhance network capacity, and accelerate transmission speed. Healthcare, education, entertainment, Industrial Internet of Things (I-IoT), autonomous cars, and smart cities are businesses that benefit significantly from these capabilities. 5G should be able to support the services these sectors have outlined adaptably.

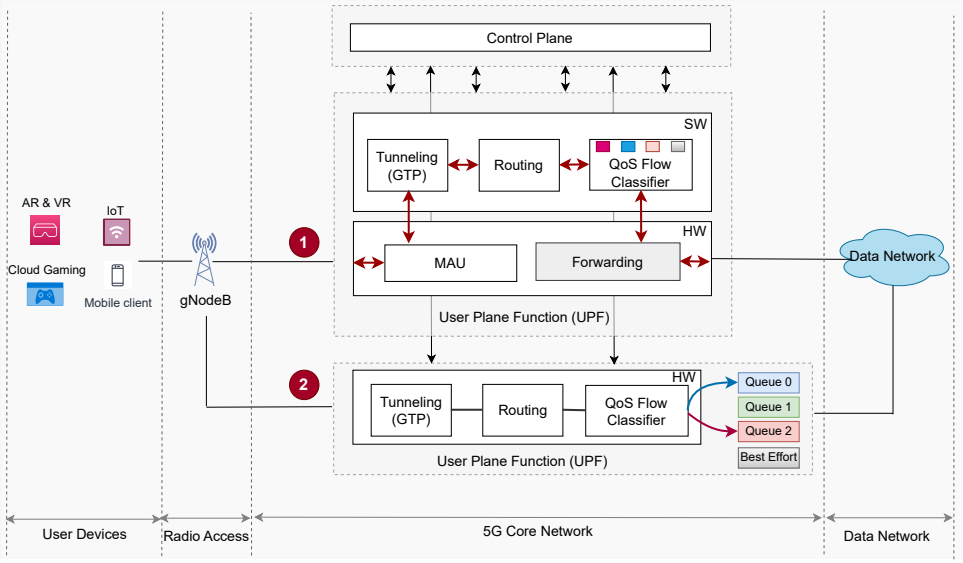


Figure 1 – Software (SW) and hardware (HW) based 5G network infrastructure and packet processing. The options for packet processing are indicated by the numbers in red circles: 1. Hybrid approach, which uses both HW and SW; 2. Offload SW functions to the switch HW.

The recent research and implementation of network virtualization and softwarization have made the 5G network more flexible and scalable (LAKE *et al.*, 2021). However, using a general-purpose Central Processing Unit (CPU) for network tasks can be slow, especially when processing packets at a fast speed (RAUMER *et al.*, 2015). For instance, as seen in Figure 1, several User Devices (UD) are connected to the mobile core user plane through the radio access network. There are two options for packet processing, as depicted in the red circle in Figure 1. In the first option, the Network Functions (NFs) such as User Plane Function (UPF) in Software (SW) have several benefits in terms of

horizontal scalability and deployment flexibility. However, the CPU resources must be devoted to NF processing in addition to other traffic monitoring and prioritizing tasks. More bandwidth is needed, raising operational and capital expenses. Furthermore, it is more challenging to offer low latency and jitter deterministically with a SW-based design (PLANCHER *et al.*, 2021). Also, SW-based solutions may take longer to classify and prioritize traffic related to time-sensitive applications, impacting the applications' overall Quality of Service (QoS).

To overcome the aforementioned problems, fast packet processing in NFs (e.g., UPF or Broadband Network Gateway user-plane (BNG-up)) with scalability and the ability to classify time-sensitive application traffic more quickly is required. There is a significant need for research the underlying Hardware (HW) to offload the CPU tasks to the specialized HW accelerators (as illustrated in red circle 2 in Figure 1), which helps to overcome these drawbacks. These accelerators have less flexibility and programmability because they process packets quickly. Offloading functions to these devices is not straightforward because of the implementation restrictions of the accelerators and Dataplane (DP) programming languages (HOGAN *et al.*, 2022). This thesis discusses these shortcomings and suggests solutions to address them.

1.1 Background and Motivation

This section will cover network virtualization and CPU programmability and how these factors may affect the QoS of traffic pertaining to time-sensitive applications. Next, we will discuss potential solutions to these problems.

1.1.1 SDN and NFV

Two revolutionary networking technologies that handle various facets of network administration and service delivery are Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) (MATIAS *et al.*, 2015). SDN allows centralized network management and programmability by separating the Control Plane (CP) from the DP in net-

work devices. This technology has several advantages, including making it easier to launch and scale network services and enabling network operators to automate and dynamically modify network behavior using SW. SDN is beneficial for several use cases, such as mobile networks, including dynamically optimizing UPF and network traffic patterns. NFV, on the other hand, virtualizes network operations previously handled by specialized HW appliances and converts them into SW-based services compatible with general-purpose CPUs. When combined, SDN and NFV allow for the creation of networks that are more flexible and effective. SDN with Programming Protocol Independent Packet Processors (P4) (BOSSHART *et al.*, 2014), for instance, offers a programmable network architecture that may change on the fly to meet Virtual Network Function (VNF) requirements.

Cache misses and interrupts can cause NFV performance issues, mainly when general-purpose CPUs are used for implementation (RAUMER *et al.*, 2015). For instance, VNFs process much data, such as network packets. Data structures, including packet buffers, routing tables, and session states, must be quickly accessed to process packets efficiently. Cache misses may result because these caches might need to include necessary data. The CPU must, therefore, retrieve it from slower main memory. Network interface cards notify the CPU to receive attention in case of interrupt handling. The CPU must halt its ongoing work, store its state, and transition to handle the interrupt request when an interrupt occurs. As a result, CPU performance is variable and confronts overhead when processing packets (ASIATICI; IENNE, 2019). Also, due to these overheads, processing a packet requires more CPU cycles. On the other hand, an Application-Specific Integrated Circuit (ASIC) developed exclusively for packet processing is faster than general-purpose CPU processing.

As discussed above, increased overhead causes lower network performance, affecting several network applications and use cases, including those that demand real-time processing (e.g., Voice over Internet Protocol (VoIP), video streaming). We will next go over the significance of having a mobile core NF with minimal latency and how the overheads above affect NF performance.

1.1.2 Mobile Network

High-performance mobile networks are necessary to satisfy the diverse QoS requirements of applications, which range from vital industry requirements to real-time communications (ERUNKULU *et al.*, 2021). Mobile operators may provide higher QoS that meets user expectations and supports the changing needs of digital services and applications by maximizing network capacity, decreasing latency, and improving reliability. The Radio Access Network (RAN) and the core network are the two primary parts of a mobile network. The core network serves as the backbone of the mobile network, managing and directing data and voice traffic among base stations and external networks such as the Internet and other mobile networks. The RAN, responsible for managing radio signal transmission and reception, is composed of base stations and their controllers.

Enhancing the performance of the mobile core user plane and implementing efficient traffic management are crucial for mobile networks for several reasons. The first reason is that 5G promises faster data speeds than earlier generations. Second, ultra-low latency (e.g., 1 ms) is needed for applications like industrial automation, remote surgery, Cloud Gaming (CG), Virtual Reality (VR) and Virtual Reality (VR), and autonomous cars to guarantee real-time response. Thus, effective user plane operations are required to process massive amounts of data traffic and reduce transmission latency. Furthermore, different applications with different QoS needs (e.g., throughput, latency, dependability) are supported by 5G networks. To guarantee consistent service quality for users and network applications, effective traffic management ensures that UPF prioritizes and deprioritizes traffic in compliance with Service Level Agreement (SLA).

The mobile core network's UPF is typically driven by NFV nodes, offering significant benefits such as scalability, flexibility, and service agility. As mentioned in section 1.1.1, however, executing functions on an NFV node results in unpredictable performance, which impacts the latency and QoS requirements for 5G use-cases. Next, we'll talk about potential solutions for these problems.

1.1.3 Programmable Dataplane

DP programmability has opened the door for new applications (KALJIC *et al.*, 2019) that require fast packet processing. Different solutions have emerged to support the programmability of HW devices, including Extended Berkeley Packet Filter (eBPF)/eXpress Data Path (XDP) (VIEIRA *et al.*, 2020), Data Plane Development Kit (DPDK)'s flow Application Programming Interface (API) (INTEL, 2010), domain-specific languages like P4 and Network Programming Language (NPL) (NPLSPEC, 2021). Though the goals of these approaches are similar, they provide different abstraction levels. eBPF/XDP and DPDK are low-level C/C++ libraries supported mainly by smart and standard Network Interface Card (NIC), requiring deep domain-specific knowledge and programming skills. In addition, DP programming languages like P4 and NPL, created for describing packet processing pipelines, have high abstraction levels and do not require deep programming skills. While NPL is designed to program switches, P4 intends to be more generic, supporting various targets through its architecture models in a common framework (e.g., same language, CP, etc.). This makes P4 a good choice for designing and implementing hybrid solutions that integrate heterogeneous devices into a common platform.

The P4 language (BOSSHART *et al.*, 2014) describes the DP's pipeline behavior of a forwarding element such as HW or SW switch and network interface card. The primary purpose of designing this language is 1) Target Independence: P4 programs can be compiled on different targets such as CPUs, Field Programmable Gate Array (FPGA), system(s)-on-chip, network processors, and ASIC, 2) Protocol Independence: P4 allows to describe the header formats, and field names of the required protocols. 3) Reconfigurability: the P4 targets can change the forwarding pipeline anytime after they are deployed. Including these features, P4 also generates the CP API for a given switch target.

As previously indicated, Programmable Data Plane (PDP) has several benefits that enable the programmable switches to handle light network functions, but there are also many implementation challenges. These include having a finite number of Static Random Access Memory (SRAM) and Ternary Content Addressable Memory (TCAM)

tables per stage, a restricted amount of arithmetic operations, a fixed number of stages, and a limited number of parsers per pipeline. Network functions must be optimized with these constraints in mind, considering P4 and target-specific limits.

A hybrid method that optimally uses CPUs and programmable HW can offer scalable UPF with low latency to fully utilize the advantages of PDP while adhering to deployment limitations. This method can also be helpful for quickly classifying traffic so that it can be prioritized to meet QoS requirements for 5G time-sensitive use cases.

1.2 Research Challenges and Goals

Given the deployment constraints, the abovementioned motivation poses several challenges for NF design and implementation. Another significant area for improvement is categorizing QoS flows with lower detection times. In mobile UPF, there are two main concerns in this work:

Challenge. *Accelerate network functions (e.g., user plane of wireless networks) with high scalability.*

To manage complex operations and millions of flows, the network functions such as UPF for 5G terminating wireless users or BNG for terminating fixed access customers need to be adaptable and expandable. With this adaptability, the user plane can provide high performance (low latency and high throughput) to meet the 5G QoS criteria. More than relying on SW solutions, programmable HW must also be considered when offloading lightweight tasks. More flexibility can be achieved by combining HW and SW solutions without sacrificing user plane performance. Offloading functions with flexibility and high performance is challenging when HW deployment constraints are considered.

Challenge. *High-performance application flows should be classified at line rate, as part of the network function (e.g., UPF).*

Prioritizing and identifying the delay-critical flows is a crucial aspect of the UPF. These operations are straightforward to implement on servers despite their complexity. However, because this approach takes more time to identify the flows, it has an adverse effect on the necessary QoS. It is a significant challenge to offload these complex operations to programmable HW. This necessitates simplifying classification processes while staying within HW constraints.

Taking into consideration the aforementioned challenges, we outline the two research goals. The research questions associated with each research goal provide more specificity. These research questions complement one another to provide a point of convergence and formulate the main idea of this thesis.

Research Goal 1: *Acceleration of UPF to meet the high-performance requirements of 5G mobile networks while maintaining scalability.*

By addressing the following research questions, this goal can be accomplished:

Research Question #1.1: *How can UPF be designed in programmable switch HW to satisfy 5G low latency requirements?*

5G User Plane Function Acceleration. Despite the flexibility offered by SDN and NFV, the performance of mobile core networks cannot yet be fully guaranteed. Developments in programmable switch ASICs allow DP VNFs operating on x86 servers to be offloaded to programmable HW with tight performance guarantees. This chapter describes the architecture and performance analysis of a 5G mobile core user plane network component.

Using P4, we implement the core functionalities of 5G UPF in programmable HW. We show that compared to NFV-based implementations, the suggested method outperformed them. However, there are numerous limitations in computations and scalability because of the restricted memory and resources of programmable HW. To address these drawbacks, we discuss the Hybrid-UPF solution in Chapter 3 and present the subsequent research

query:

Research Question #1.2: *How can the UPF be scaled with high-performance?*

Hybrid Design for 5G User Plane Function. This chapter focuses on target-specific properties-based hybrid pipeline approaches for UPF. P4 targets are utilized for either the complete or disaggregated UPF, assigning packet processing data pathways to DPDK/x86 software or P4 hardware depending on flow characteristics. (Heavy Hitter (HH), for example). By utilizing HH-based hybrid-UPF, most of the traffic will be handled by switch HW, with only light traffic being handled by NFV nodes. This lowers overall CPU consumption and enhances network performance.

Research Goal 2: *Classify traffic at line rates to meet the desired QoS.*

By addressing the following research question, the second goal can be achieved:

Research Question #2: *How can high-performance application flows in programmable HW be classified with high accuracy and lower detection time to enable use for QoS scheduling?*

We mainly focus on the HH and CG flow classification, which are described as follows:

Heavy Hitter Detection in the Data Plane. In this chapter, we present a novel metric for HH detection. We compute the Inter Packet Gap (IPG) rather than relying on the packet count for each flow. This method detects flows faster and produces more accurate results. We demonstrate that, compared to current solutions, it is possible to reduce the control channel overhead since the controller does not need to reset the counter frequently.

Hyrbid-UPF uses this HH detection approach, as mentioned in the previous research question, to increase scalability. Furthermore, we show how this proposed method can be utilized to improve QoS by scheduling flows in a high-priority queue within a programmable switch. Alternative approaches could involve using Machine Learning (ML) to classify specific application flows directly in the dataplane. We go into further detail

on this in Chapter 5.

Ml-based Cloud Gaming Traffic Detection. This chapter presents an ML model and the computation of essential features to train the model for classifying cloud gaming traffic in programmable dataplanes. Enhancing QoS and providing end users with a top-notch gaming experience is possible by promptly identifying CG flows in mobile networks. We demonstrate how the detection process may be integrated with switch hardware while adhering to all current switch HW limitations.

1.3 Contributions

All contributions are mapped in Figure 2 along with the related published papers listed below. We investigate PDP's shortcomings and develop methods to enhance the UPF's performance. By using the strategies and contributions outlined in the upcoming subsections, each defining a step in the development of this work, we achieve the goal discussed in Section 1.2.

All the contributions and publications are briefly explained and summarized as follows:

[A] **Suneet Kumar Singh**, Christian Esteve Rothenberg, Gyanesh Patra, Gergely Pongrácz. "Offloading Virtual Evolved Packet Gateway User Plane Functions to a Programmable ASIC". In *2019 CoNEXT Workshop on Emerging in-Network Computing Paradigms (ENCP '19),*, Orlando, FL, USA. December 2019.

[B] **Suneet Kumar Singh**, Christian Esteve Rothenberg, Jonatan Langlet, Andreas Kassler, Sandor Laki, Gergely Pongracz. "Hybrid P4 Programmable Pipelines for 5G gNodeB and User Plane Functions". In *IEEE Transactions on mobile computing.* December 2023.

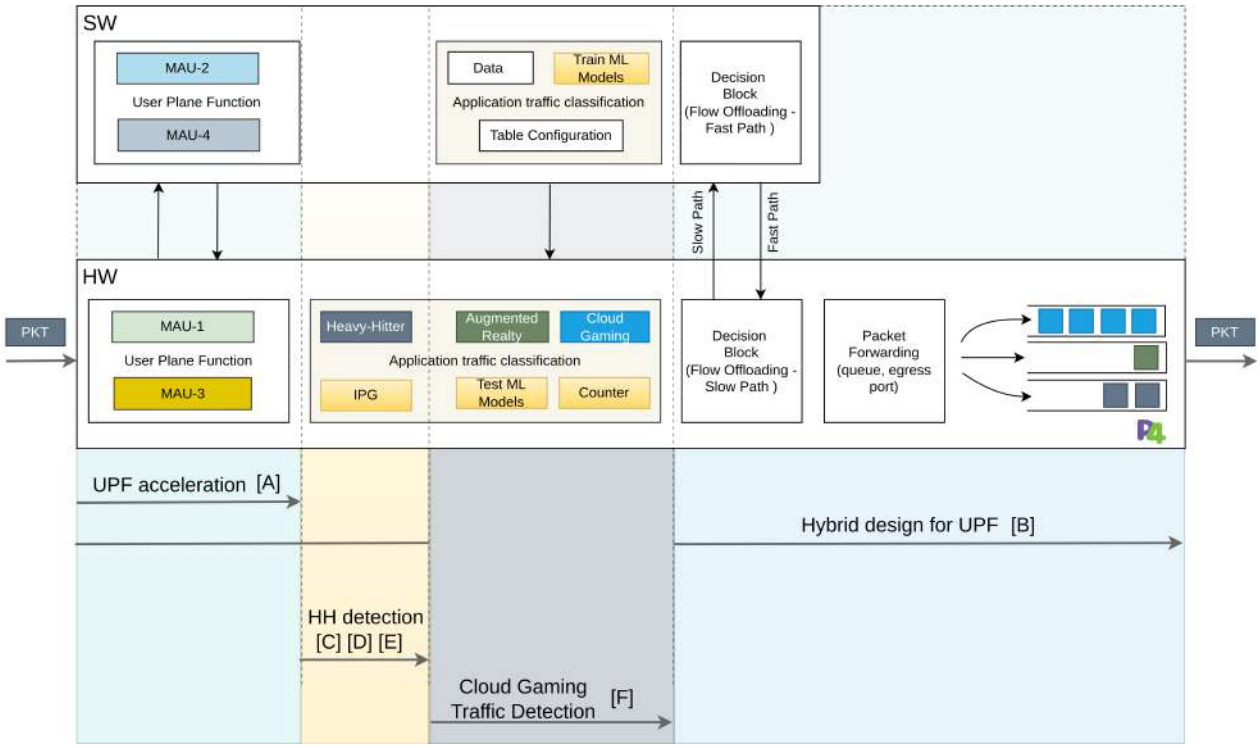


Figure 2 – Contributions and associated published works, indicated by the letters enclosed in square brackets.

[C] **Suneet Kumar Singh**, Christian Esteve Rothenberg, Marcelo Caggiani Luizelli, Gianni Antichi, Gergely Pongrácz. "Revisiting Heavy-Hi!ers: Don't count packets, compute flow inter-packet metrics in the data plane". In *SIGCOMM Posters*. August, 2020.

[D] **Suneet Kumar Singh**, Christian Esteve Rothenberg, Marcelo Caggiani Luizelli, Gianni Antichi, Pedro Henrique Gomes, Gergely Pongrácz. "HH-IPG: Leveraging Inter-Packet Gap Metrics in P4 Hardware for Heavy Hitter Detection". In *IEEE Transactions on Network and Service Management*. September, 2023.

[E] **Suneet Kumar Singh**, Christian Esteve Rothenberg, Pedro Henrique Gomes, Gergely Pongrácz. "Heavy hitter flow classification based on inter-packet gap analysis". PCT Patent Application Serial No. PCT/IB2021/053738. Filing Date: 02 Apr, 2019. Publication Number: WO2021229361A1. Publication Date: 18 November, 2021. Available link: https://patents.google.com/patent/WO2021229361A1/

[F] **Suneet Kumar Singh**, Christian Esteve Rothenberg, Alireza Shirmarz, Fábio Luciano Verdi, Israat Haque, Gyanesh Patra, Gergely Pongrácz. "MATADOR: Ml-

based Cloud Gaming Traffic Detection entirely in Programmable Hardware". In *IEEE Conference on Network Function Virtualization and Software Defined Networks*. November, 2024.

Available open-source repositories:

- [A] [B] https://github.com/intrig-unicamp/macsad-usecases/tree/master/p4-16/vEPG-TNA
- [B] [C] [D] [E] https://github.com/intrig-unicamp/P4-HH
- [F] https://github.com/intrig-unicamp/cg-classifier

1.3.1 5G User Plane Function Acceleration

- **Main objective:** We present the design and performance evaluation of the UPF, a critical element of 5G Mobile Packet Core (MPC). We describe the P4-based Uplink (UL) and Downlink (DL) pipelines and evaluate a SW and HW implementation based on x86, a Barefoot Tofino HW switch, the Open Network Operating System (ONOS) controller, and P4Runtime support to manage match-action tables.
- **Main Contributions:**
  1. UPF functions design in P4 featuring [1]: (i) GPRS Tunneling Protocol (GTP) (de)encapsulation; (ii) Virtual eXtensible LAN (VxLAN) (de)encapsulation of GTP flows; (iii) Internet Protocol (IP) routing towards the Internet and eNodeB nodes; (iv) Management of Tunnel Endpoint Identifier (TEID) for flow multiplexing in GTP sessions; (v) Stateless firewall; (vi) Eth/IP forwarding to/from datacenter gateways.
  2. Implementation of UPF on Barefoot Tofino HW switch using P4 Runtime APIs and the ONOS controller to manage table entries.

[1] https://github.com/intrig-unicamp/macsad-usecases/tree/master/p4-16

3. Experimental evaluation of (i) performance of ASIC offloaded UPF compared to x86 server, (ii) scalability for increasing active mobile users, and (iii) Barefoot Tofino HW resource utilization.
4. Memory efficient P4 match+action mapping methods delivering 8x increase in table entry capacity (active users).

- **Related Publication:** [A]
- **Explored In:** Chapter 2

1.3.2 Hybrid Design for 5G User Plane Function

- **Main Objective:** We focus on hybrid pipeline designs for UPF and next-generation NodeB leveraging target-specific features and an insightful discussion of P4 and target challenges and limitations. The entire or disaggregated UPF runs on P4 targets and allocates packet processing data paths in P4 HW or DPDK/x86 SW based on flow characteristics (e.g., Heavy Hitter (HH)) and QoS requirements (e.g., low-latency slices).
- **Main Contributions:**
  1. We propose and implement hybrid design model for 5G UPF defined in P4 over different targets (ASIC and x86).
  2. We identify and discuss limitations and implementation challenges of the P4 language and target-specific nuances.
  3. We perform an extensive evaluation with real network traffic traces to analyze the scalability, throughput, and latency for varying User Equipment (UE) sessions.
- **Related Publication:** [B]
- **Explored In:** Chapter 3

1.3.3 Heavy Hitter Detection in the Data Plane

- **Main Objective:** In this work, we show that using specified time windows can lead to high inaccuracies. We make a case for rethinking how switches analyze the incoming packets and propose to leverage per-flow IPG analytics instead of using flow counters for HH detection. We propose an algorithm and present a P4 pipeline design using this new metric in mind. We implement our solution on P4 HW and experimentally evaluate it against real traffic traces.

- **Main Contributions:**

  1. We present in detail the methods for HH characterization using IPG as the key metric (first introduced in [16]) and demonstrate its practicality.
  2. We discuss the challenges in calculating per-flow IPG in current off-the-shelf programmable switches and implement the proposed method on P4 Tofino HW.
  3. We propose an algorithm inspired by a state-of-the-art HH detection algorithm to work with IPG and experimentally evaluate its detection capabilities using real traffic traces in both simulation and P4 HW.
  4. We showcase IPG-based HH detection for a QoS application in a mobile network user plane pipeline.

- **Related Publication:** [C] [D] [E]

- **Explored In:** Chapter 4

1.3.4 Machine-Learning-based Cloud Gaming Traffic Detection

- **Main Objective:** Recent efforts have been made to improve networks and perform CG traffic classification on NFV nodes to prioritize these flows, with the goal of delivering a high-quality gaming experience. To ensure that CG offers gamers a responsive and seamless experience, more work must be done to reduce detection

times. The limitations and difficulties that P4 and switch targets must overcome are thoroughly examined in this study, which also uses ML on P4 switch HW to classify CG traffic. The performance is analyzed using Tofino switch HW and a simulator based on Python; the results are compared with alternative approaches.

- **Main Contributions:**

  1. We describe in detail the CG classification approach on a Tofino switch hardware and demonstrate its feasibility using two CG datasets: one for ML model training and testing, and another used exclusively for MATADOR performance confirmation on a different dataset.
  2. We discuss P4 language's constraints and implementation difficulties as well as target-specific subtleties.
  3. Our proposal includes features that are necessary for CG detection that are compatible with P4-Tofino Native Architecture (TNA) while maintaining higher detection accuracy.
  4. We optimize the ML model to fit all processing, including feature computation and packet forwarding operations, within the limited number of stages of the Tofino switch.

- **Related Publication:** [F]

- **Explored In:** Chapter 5

1.3.5 Further Contributions, Results & Collaborative Activities

Throughout this effort, the following additional related contributions were made as co-authors in various academic publications:

- Alireza Shirmarz, Fábio Luciano Verdi, **Suneet Kumar Singh**, Christian Esteve Rothenberg. "From Pixels to Packets: Traffic Classification of Augmented Reality

and Cloud Gaming". In *6th IEEE International Conference on Network Softwarization (NetSoft'24) - Technical Sessions*, MO, USA. Jun, 2024.

- Francisco Germano, Fabricio Eduardo Rodriguez, Ariel Góes de, **Suneet Kumar Singh**, Marcelo Caggiani, Christian Esteve Rothenberg, Gianni Antichi. "Video Streaming QoE Meets Programmable Data Planes: In-Network QoE Management". In *IEEE Network - The Magazine of Global Internetworking*, 2024.

Collaborations with industry players such as Open Networking Foundation (ONF) were also secured as part of the internship process while this thesis proposal was being developed. Many responsibilities were assigned throughout the internship, including leading the Aether project[2] and building the first Tofino-based QoS solution for UPF. In order to provide QoS support for fabric-tna, feature support had to be added to the P4 program, UP4 ONOS application, and Packet Forwarding Control Protocol (PFCP) agent. During this internship, two internal QoS demos were given, using live video streaming and the PFCP agent to set the priority of flows within the switch.

Public git repositories show contributions made during an internship are listed as follows:

- https://github.com/stratum/fabric-tna
- https://github.com/omec-project/up4
- https://github.com/opennetworkinglab/onos
- https://github.com/stratum/fabric-line-rate-test

1.4 Outline

This is how the rest of the thesis is structured. For fast packet processing, Chapter 2, "5G User Plane Function Acceleration," suggests offloading 5G UPF to programmable hardware. In particular, it analyzes and compares the performance of HW- and SW-based

[2] https://opennetworking.org/aether/

UPFs. The next Chapter 3, titled "Hybrid Design for 5G User Plane Function," describes how to use target-specific characteristics in the hybrid UPF design to scale user devices without compromising performance. HH Detection in the Data Plane (Chapter 4): This approach helps to improve QoS by identifying HH flows in UPF running on programmable switches. Furthermore, we investigate the ML method on PDP to understand traffic behavior and classify it appropriately. We go into detail about this in Chapter 5. In Chapter 6, "Conclusion & Future Work," we share our conclusions and thoughts on possible future initiatives.

2 5G User Plane Function Acceleration

2.1 Background and Motivation

Next-generation Mobile Packet Core (NGMN) describes twenty five use cases for 5G (HATTACHI, 2015), many of which require extremely low latency and high data rates. To satisfy such performance targets and the rapid traffic growth, next generation MPC calls for flexible, scalable, and cost-effective approaches (COSTA-REQUENA *et al.*, June 2015). To this end, mobile operators are leveraging key technologies such as SDN (KREUTZ *et al.*, 2015) and NFV (LI; CHEN, 2015).

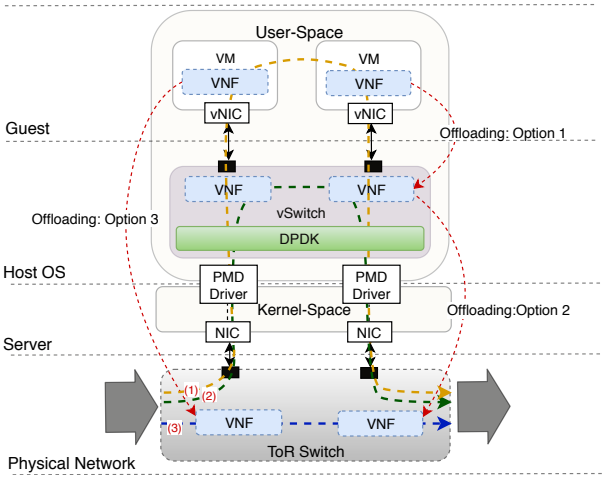


Figure 3 – Performance bottlenecks in NFV infrastructure and VNF offloading options in multi-tenant environment to accelerate packet-processing functions

A network softwarization approach to the MPC provides agility and flexibility (HE *et al.*, 2019) by decoupling the control and dataplanes, and cost-effectively moving MPC network functions to computational pools at strategic data centers (KEMPF B. JOHANSSON; NILSSON, Oct, 2012). As shown in Figure 4, user plane entities of MPC are being deployed in telco cloud infrastructures as VNFs on x86 servers, proving adequate performance for some 5G services. However, for time-sensitive applications like autonomous driving, robot control and VR/AR, there is a need to avoid performance bottlenecks (RAJAN *et al.*, 2015) of NFV infrastructures and the non-deterministic behavior of x86 soft-

ware stacks (MOHAMMADKHAN *et al.*, 2016b; LINGUAGLOSSA *et al.*, 2019).

The advent of programmable switch ASICs and high-level network-specific languages like P4 (BOSSHART *et al.*, 2014) opens the door to offloading user plane VNFs from x86 servers to the network infrastructure. Figure 3 illustrates VNF offloading options to accelerate packet processing functions in programmable SW or HW switches. Performance bottlenecks due to memory read/writes and SW/HW resource sharing (LINGUAGLOSSA *et al.*, 2019) can be avoided when directly running network functions in a programmable Top-of-rack (ToR) switch.

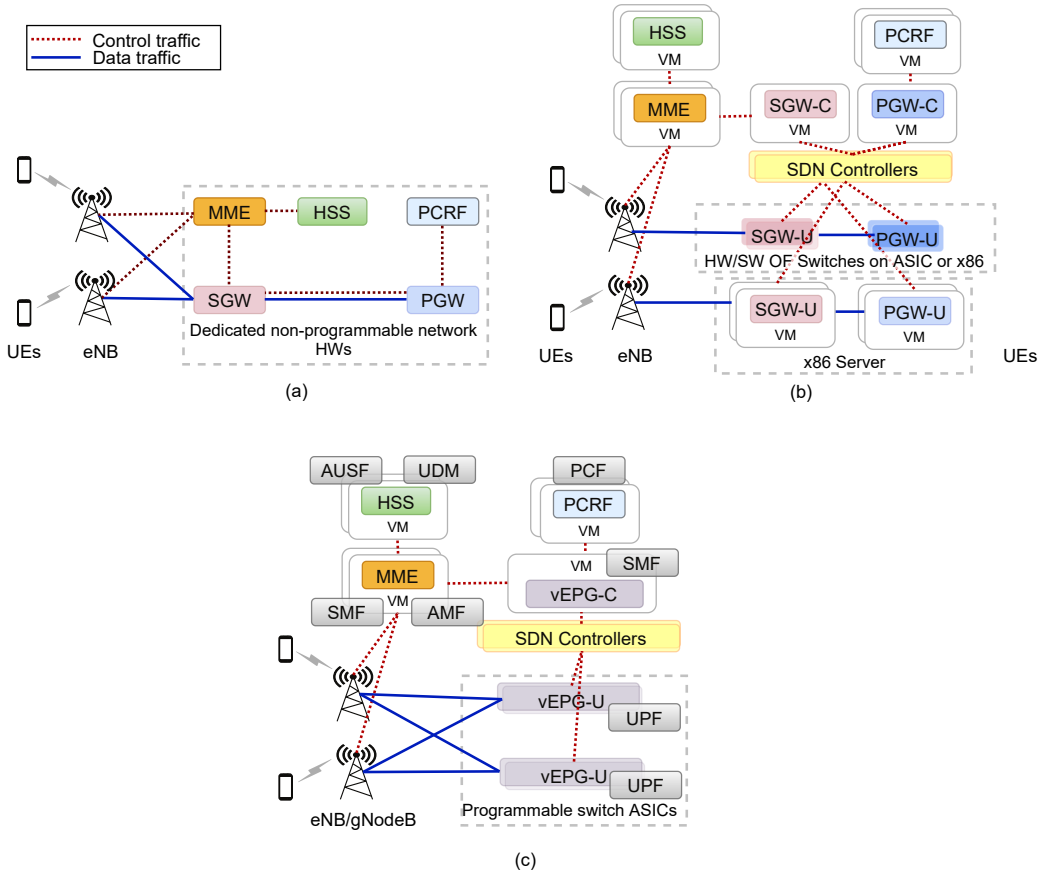


Figure 4 – MPC architecture implementation alternatives: (a) Traditional 4G MPC appliances, (b) 4G MPC using SDN and NFV with full and partial virtualization, and (c) 4G/5G MPC user plane functions offloaded to programmable ASICs

In this chapter, we investigate offloading MPC user plane functions to P4-capable programmable switch ASICs. More specifically, we present a P4 implementation of a 5G UPF that compiles in a Barefoot Tofino switch and uses the ONOS[1] controller to manage

[1] <https://github.com/opennetworkinglab/onos>

the match-action table entries through P4 Runtime[2]. To analyze the performance gains when offloading the UPF to a P4 switch ASIC, we also compile the UPF P4 pipeline to a high-end x86 using the MACSAD (PATRA *et al.*, 2017) compiler, which leverages Open Data Plane (ODP)[3] APIs to run P4 applications on different target platforms (e.g., x86, Advanced RISC Machine (ARM)) at high-speed (DPDK acceleration).

2.1.1 5G Mobile Packet Core

MPC refers to the core functional components of modern mobile networks. Figure 4(a) illustrates a traditional Long-Term Evolution (LTE) MPC network, where the Serving Gateway (SGW) routes uplink and downlink IP traffic to radio network evolved evolved Node B (eNB) stations. SGWs handle multiple eNodeBs and provide handover between eNodeBs or between LTE and other 3GPP technologies. The Packet Data Network Gateway (PGW) connects the MPC with external IP networks and performs packet filtering, charging, and QoS enforcement as instructed by the Policy and Charging Rules Function (PCRF).

Unlike the SGW, the PGW acts as anchor point for mobility between 3GPP and non-3GPP networks. Both SGW and PGW use GTP for the communication between eNodeB and PGW via SGW, which maintains the GTP sessions between eNodeBs and the PGW. The Mobility Management Entity (MME) is responsible for security procedure such as end-user authentication with the support of Home Subscriber Server (HSS), terminal-to-network session handling, and user tracking across the network.

The Evolved Packet Gateway (EPG) merges the functions of both SGW and PGW and can be implemented as a physical (bare-metal appliance) or a virtual node (UPF VNF). Figure 4(c) shows the 4G-5G MPC functions mapping. In this proposal, we focus on the 5G MPC.

[2] <https://github.com/p4lang/p4runtime>
[3] <https://opendataplane.org/>

2.1.2 Programmable ASICs and P4

Programmable ASICs such as Intel Flexpipe, Cisco Doppler, Cavium (Xpliant) and Barefoot Tofino are emerging networking technologies that allow developers to add new protocols and custom packet de-parsing and match-action pipelines.

With growing support, P4 (BOSSHART *et al.*, 2014) simplifies the programmability of datapath pipelines through high-level language constructs. Programming ASICs with P4 allows implementing feature-rich datapaths supporting high data rates at deterministic ultra-low latency with less power consumption compared to x86 approaches. Therefore, the rationale of offloading MPC user plane functions to programmable ASICs.

2.1.3 Related Work

Approaches to re-design MPC (NGUYEN *et al.*, 2017) include (*i*) NFV-based (YOUSAF *et al.*, 2013) , (*ii*) SDN-based (LI; REXFORD, 2012), and (*iii*) SDN/NFV-based-MPC architectures (MOHAMMADKHAN *et al.*, 2016a). MPC approaches based on SDN/NFV are prevailing due to flexibility (HE *et al.*, 2019) and backward compatibility among other factors (NGUYEN *et al.*, 2017). Decoupling the control and user planes of MPC allow them to scale separately in a cost-effective manner.

In (COSTA-REQUENA *et al.*, June 2015; HASEGAWA; MURATA, 2015; KAIPPALLIMALIL; CHAN, 2014), authors propose MPC architectures implementing SGW and PGW user-plane functions in OpenFlow switches with GTP tunneling support and MPC control functions executing in the cloud. (PATRA *et al.*, 2017) and (KANNAN; CHAN, 2019) point to P4 programmable switches as capable of defining complex forwarding pipelines with high-performance.

Efforts in (KUNDEL *et al.*, October 2019; CASCONE; CHAU, March 2018) are closest to our work on MPC dataplanes embracing P4 to overcome well-known limitations from OpenFlow on protocol flexibility, extensibility, and full hardware support. (KUNDEL *et al.*, October 2019) presents a P4 implementation of a Broadband Network Gateway (BNG) compiled into P4-NetFPGA, Netronome P4-SmartNIC, and Barefoot

Tofino switch. The performance evaluation against x86 targets provide similar insights compared to our work sustaining the potential gains of hardware offloading approaches. (CASCONE; CHAU, March 2018) outlines a VNF offloading approach of S/PGW functionality into a data center fabric solution for MPC where GTP headers are encapsulated in HW switches. The S/PGW pipeline design in P4[4] follows the Method 1 implementation described in our work.

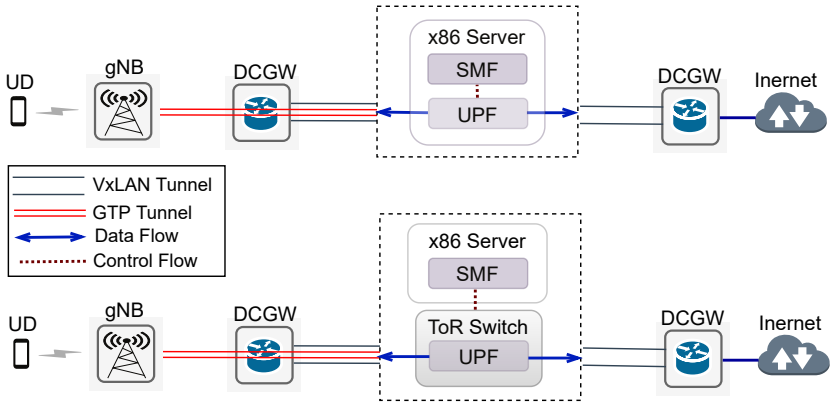


Figure 5 – UPF offloading use case : UPF running on (a) x86 server and (b) Barefoot Tofino ToR switch.

2.2 UPF P4 Pipeline & Use Case

Figures 4 (b) and 4 (c) show SDN and NFV-based MPC architectural approaches with different candidate targets to run MPC gateway functions. Our goal is implementing in P4 a UPF combining both SGW and PGW user plane functions. The target UPF use cases are presented in Figure 5, where (a) UPF and Session Management Function (SMF) run on the x86 server, and (b) UPF is offloaded to a programmable ASIC.

We divide the UPF datapath in UL and DL sub-pipelines where UPF-UL indicates the traffic coming from UD and UPF-DL refers to traffic from the Internet towards the users. All implemented user plane functions are given in Table 1. The total size of the P4 program is around 750 lines of code.

Effectively occupying 100% of the available memory footprint to maximize the amount of active UDs ultimately depends on the P4 program structure in addition to

[4] <https://github.com/opennetworkinglab/onos/blob/master/pipelines/fabric/src/main/resources/include/spgw.p4>

potential program- and target-specific compiler optimizations – a topic we scratch the surface of later in the evaluation section. Next, we provide further details on the UPF pipeline.

Infrastructure Virtualization. VxLAN provides the isolation capabilities among VNFs in multi-tenant environments. VxLAN (de-)encapsulation of L2 frame within UDP header and handling outer L2 forwarding and L3 routing towards Data Center Gateways (DCGWs) are common for both UPF-UL and UPF-DL. Outer and inner IP and Media Access Control (MAC) addresses represent underlay and overlay network, respectively.

L2/Ethernet. Incoming packets are parsed and then passed through the L2 tables to execute two separate lookups for checking source and destination outer MAC addresses.

Firewall. After successful key matches in L2 tables, outer IPv4 and GTP headers validity check is performed to select the packet flow towards UL or DL pipeline. If GTP headers are valid, packets follow UL, otherwise they pass through the DL pipeline. In both the cases, the global firewall rule is applied to mitigate the unauthorized access of servers or EPC network connected to the internet.

GTP Decap/Encap. In GTP (de-)encapsulation , the UPF removes or adds GTP headers. GTP encapsulation table lookup is responsible for checking UDs IP addresses, while GTP decapsulation table matches UPF IP address to apply the actions accordingly. Also, this table manages inner L2/L3 MAC, the TEID, and routes inner IP packets towards the gNodeB or DCGW.

IPv4 Routing. Outer IP routing towards DCGWs.

2.3 Realizing UPF with P4 Switch ASIC

Like any HW-SW system design, offloading MPC gateway functions to programmable hardware is subject to certain trade-offs. The fixed memory resources of ASICs provide performance guarantees but pose hard limits on the number of active UDs, which could be 'arbitrarily' large (e.g., 10^6 - 10^9) in vSwitch or VM/containers on high-end x86 servers.

Table 1 – 5G User Plane Functions: Uplink and Downlink

UPF-UL	**UPF-DL**
Match Outer L2 and L3	
VxLAN Decapsulation	
Apply Metering	
Match 5 Tuple, set Queue Id	
Validate IPv4/GTP	
Match UPF IPv4	Match User Device IPv4
Apply global firewall rules	
GTP Decapsulation	GTP Encapsulation
IP Routing towards Internet	IP Routing towards eNodeB
VxLAN Encapsulation	
L3 Routing towards DCGW	
Outer L2 Forwarding	

Furthermore, in programmable ASICs, match-action tables are assigned to stages containing limited amount of resources following specific (fixed) allocation rules.

After describing testbed, we evaluate the performance of UPF in HW and SW targets. The objective is not to directly compare both targets but to profile their performance behaviour. Finally, we introduce two P4 code optimization methods to re-design match-action tables to reach larger active UD capacity.

2.3.1 Testbed

Figure 6 shows the testbed featuring Open Source Network Tester (OSNT) (GIANNI *et al.*, 2014) as the traffic generator using NetFPGA-SUME with 10G SFP+ interfaces connected to the Device Under Test (DUT), in our case (1) Edgecore Wedge 100BF-32X with Barefoot Tofino ASIC[5], or (2) x86 server with Intel Xeon D-1518 processor (4 CPU cores, Cache 6M, 2,20 GHz) for the UPF application generated by the MACSAD compiler for ODP-DPDK x86 targets (cf. (PATRA *et al.*, 2017)). Additionally, by setting the Forwarding Error Correction (FEC) option to NONE from the command line interface to make ports up, the FEC setting is turned off in the Tofino interfaces. Although it is outside the scope of this work, we plan to analyze the effects of FEC enabled or disabled in the future.

[5] <https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335>

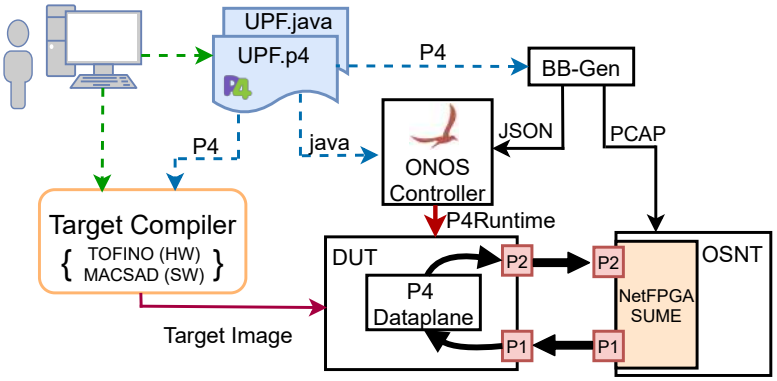


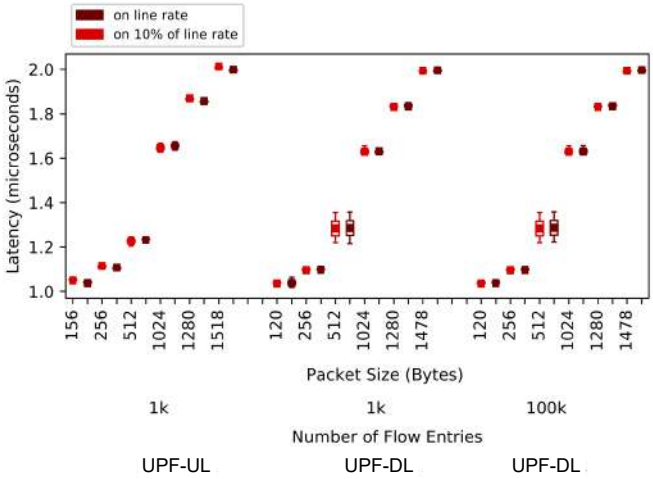
Figure 6 – Testbed for UPF evaluation on HW and SW.

The same UPF pipeline functions (Table 1) written in $P4_{16}$[6] are used as an input P4 code for the Barefoot and MACSAD compilers to generate the target images for the Tofino HW and x86 SW platforms, respectively. P4Runtime through ONOS controller (version 2.1.0) REST APIs is used to manage the table entries in the Barefoot Tofino DUT. From the UPF P4 code, we create the required java files such as the pipeline interpreter for mapping between ONOS known headers/actions to P4 specific entities among other files to create an ONOS application. The BB-Gen (RODRIGUEZ *et al.*, 2018) tool is used to generate the JSON file for the required table entries to be populated by ONOS and the PCAP packet traces to feed OSNT to generate the test traffic.

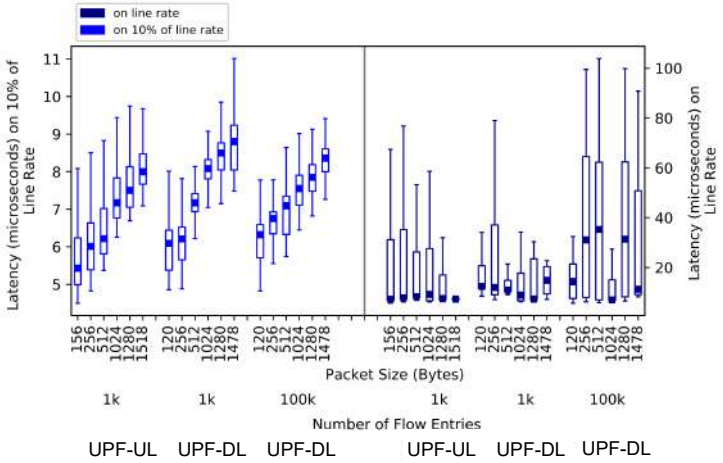
2.3.2 Performance

We analyze the UPF performance in terms of latency and throughput on the hardware switch (Barefoot Tofino HW) and the x86 server (MACSAD SW) for various traffic patterns and packet sizes. We measure latency under two different load scenarios. Firstly, OSNT sends packets at 10% of line rate, and, secondly, packets are transmitted at line rate. Latency is measured one way end-to-end by OSNT (see Figure 6) and includes propagation, transmission, queuing, and processing delays (cf. (ANTICHI *et al.*, 2014)). OSNT hardware measurements add 16 bytes headers to the original packet for TX/RX timestamps. Throughput is calculated in million packets per second (mpps). Minimum packet sizes are 140 and 104 bytes for UPF UL and DL, respectively. Since 36-byte headers

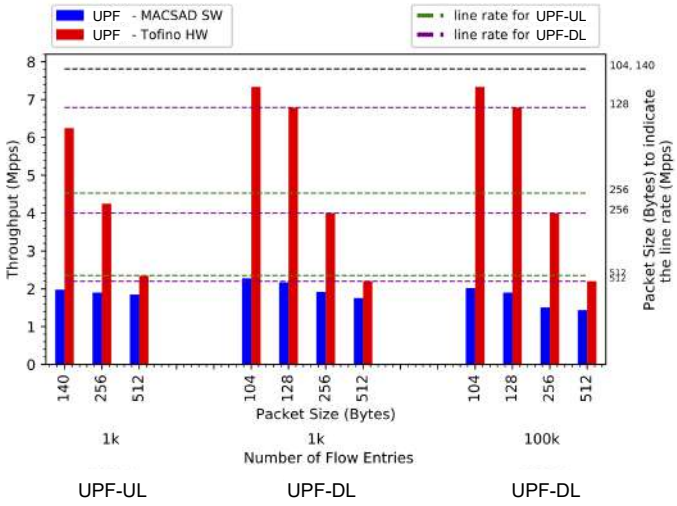
[6] Minor differences in $P4_{16}$ code are due to target architecture specificities



(a) Latency for Barefoot Tofino HW



(b) Latency for x86 using MACSAD SW



(c) Throughput

Figure 7 – UPF performance on HW and SW targets.

are added/removed during GTP (de-)encapsulation, the received packet size is modified accordingly.

For table key matching, we consider 1k unique IP addresses to apply firewall rules for each Firewall UL and DL table. Since the number of table entries in the GTP Encap table depends on the number of UDs, we use 1k, 10k, and 100k unique table entries (UD IP). A single entry is considered for each of the remaining tables in the UPF pipelines.

Performance for different workloads. Due to the Tofino HW line rate packet processing, the observed performance of UPF-UL and UPF-DL is not influenced by the traffic rate. The latency of the UPF on Tofino HW (Figure 7(a)) remains below 2μs with the packet size being responsible for the transmission and serialization delays (KANNAN; CHAN, 2019). Additionally, we do not see a significant change in latency when we alter the packet size and number of flows due to ASIC's line rate packet processing. Compared to x86 (see Figure 7(b)), the observed latency is about 2-4x lower than in case of 10% line rate and 20-40x at line rate – an evidence of the non-deterministic behavior (cf. large jitter) of x86 packet processing due to cache misses, branch mis-predicts, long latency instructions and exceptions, etc. (WEAVER D. TERPSTRA, 2013).

Increasing active user devices. We now measure latency and throughput as number of active UD increases. As shown in Figure 7, except for smallest packet sizes (104 and 140 bytes) because of adding headers in encap in case of DL which causing bottelneck at output port to drop some packets and removing headers in case of UL to drop throughput at output port, UPF achieves line rate on the Tofino HW for all packet sizes when increasing UD from 1k to 100k. Figure 7 shows that UPF achieves line rate on the Tofino HW for all packet sizes when increasing UD from 1k to 100k, with the exception of the smallest packet sizes (104 and 140 bytes), where adding headers in GTP encap in the case of DL causes a bottleneck at the output port to drop some packets, and removing headers in the case of UL causes throughput at the output port to drop. The throughput is 2-8x higher on the Tofino hardware compared to x86. Increasing UD does not affect the UPF performance on either platforms due to the deterministic nature of the hardware ASIC, and the use of hugepages by DPDK to efficiently handle the Translation Lookaside Buffers (TLB) in case of x86 servers (LINGUAGLOSSA *et al.*, 2019).

Table 2 – L2 and UPF on Tofino HW (highlighted rows) and x86 (PS: Packet Size in Bytes; 10% of Line Rate; 1k Entries)

Platforms (Tofino/x86)	PS	Average Latency (μs)			Throughput (Mpps)		
		L2	UPF-DL	UPF-UL	L2	UPF-DL	UPF-UL
x86	156	5.01	6.17	5.42	5.24	1.98	1.87
Tofino HW	156	0.97	1.04	1.04	7.10	5.89	6.02
x86	n256	5.52	6.23	6.10	4.20	1.92	1.80
Tofino HW	256	1.04	1.10	1.09	4.52	4.00	4.25
x86	512	5.90	7.10	6.23	2.26	1.76	1.85
Tofino HW	512	1.24	1.25	1.25	2.34	2.20	2.34

Pipeline complexity. As a baseline performance experiment, we run a simple L2 use case on both platforms and analyze the impact of the UPF pipeline complexity (Table 2). As one would expect from the HW implementation, we do not see any significant performance difference between L2 and UPF. In contrast, latency and throughput in the x86 are impacted by the increased pipeline complexity. These outcomes confirm that, in addition to overall total packet processing capacity requirements (e.g., 3.2 Tbps of 32x100G interfaces), for increasing user plane functions complexity and whenever strict performance guarantees are required, programmable ASICs become the candidate design choice.

2.3.3 Scalability / HW Resource Utilization

We now turn the attention to the scalability properties by analysing the average ASIC hardware resource utilization for an increasing amount of active UD. We focus the HW resource consumption analysis on the UPF-DL where the UD IP exact-match lookup is performed and SRAM becomes the critical resource. We analyze the resource utilization of different implementation alternatives, from our initial naive one to more optimized table designs.

Naive Implementation. We use our very first P4-based UPF pipeline design for 100k UDs as a baseline to evaluate the SRAM consumption. As shown in Figure 8, up to 220k entries could be compiled. We observed that large amounts of memory were occupied by GTP Encap action data (including constant values – hence the naiveness of the implementation). Under these circumstances, to scale up the total number of entries, we identified

some P4 code optimization methods based on minimizing the action data per table entry and memory efficient match+action table splits. The basic idea is to execute the UD IP lookup on a table with less action data, reducing the memory consumption and, therefore, allowing for an increased number of table entries.

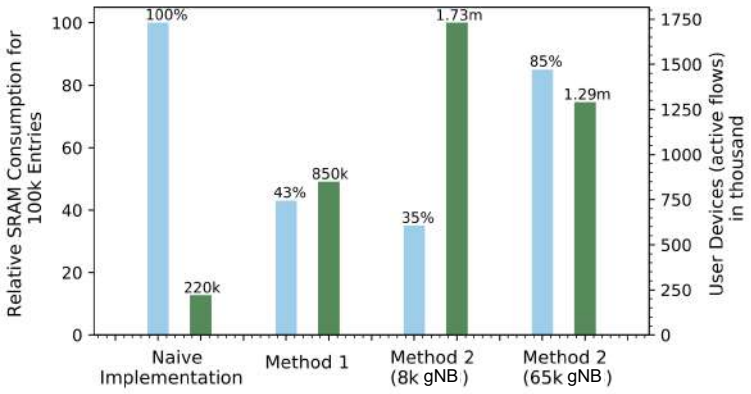


Figure 8 – Resource consumption in Tofino switch using different optimization methods. The blue bar indicates relative SRAM consumption in percentage, while the green bar indicates the number of user devices.

Method 1. The UD IP lookup is performed with an action to set the TEID and destination IP (i.e. Next-Generation Node B (gNB) address) for downlink sessions, whereas the remaining actions are executed independently. Under this arrangement, the GTP Encap table becomes narrower and can scale up to 850k UDs. For 100k UDs, this method consumes 43% of SRAM when compared to the naive implementation.

Method 2. The GTP Encap table is divided into two tables, one for UD's IP lookup with an action to set the corresponding eNB ID, and a second table to execute the actions using the gNB ID as a key to index the gNB 32-bit IP to forward the packets. This method allows up to 1.73 million(m) table entries for 8k gNBs (13-bit eNB ID), and 1.29 m entries for 65k gNBs (16-bit gNB ID) – an arguably over-provisioned design of eNBs in scope of a single UPF node. For 100k UDs, compared to the naive approach, 35% and 85% of SRAM is occupied for 8k and 65k gNBs, respectively. The increased scale obtained through such a table split method is explained by the action data bits (16 or 13) saved per entry in the larger GTP Encap table at the cost of an extra table with fewer entries (65k or 8k) of 32-bit action data (gNB IP) each.

Lessons learned from these initial P4 pipeline design optimization methods suggest that different levels and types of code optimization techniques should be investigated to achieve higher scalability by effectively using the target HW resources.

2.4 Concluding Remarks

This chapter on hardware offloading of MPC user plane functions running at line rate suggests a promising approach, featuring low and predictable latency and the potential to scale to over 1.5 million active users in a commercial P4 switch. P4 programs, similar to code written in other programming languages, can be effectively optimized. The design of the P4 pipeline tables can significantly influence the scalability, as demonstrated by our alternative designs and optimization techniques. Future studies that build on the research in this chapter should concentrate on the scalable mobile network and missing features. The following are some considerations relevant to this study:

- Future activities will devote efforts to implementing some missing features of UPF, such as charging functionality and rate limiting, in addition to more elaborate traffic management features per user or service type in the spirit of network slicing.
- This chapter primarily focuses on implementing UPF on programmable hardware and improving its performance using preliminary P4 code optimization. Nevertheless, various P4 code optimization techniques will be explored and further developed to enhance the ASIC's resource utilization for stateful applications.
- Another line of study leading to a highly scalable UPF solution is a hybrid architecture that dynamically distributes P4 match action tables, and user states over numerous HW switch ASICs and SW servers. In the next chapter (Chapter 3), we will delve deeper into the hybrid UPF and talk about how to scale the network in terms of user count without adversely impacting performance.

3 Hybrid Design for 5G User Plane Function

3.1 Background and Motivation

3.1.1 Next-generation Requirements and Existing Solutions

Next-generation mobile networks, including future versions of the currently deployed 5G networks, aim to support a diverse set of traffic mixes stemming from demanding applications such as Extended Reality (XR), I-IoT, and self-driving vehicles (HATTACHI, 2015). To support the ever-increasing data rates while at the same achieving low and predictable latency for critical services (e.g., Ultra Reliable Low Latency Communications (URLLC)), the user plane of the mobile core network is the most critical component because it deals with the packet forwarding between the UE and the packet gateway, which connects the operator network to the Internet. Consequently, the user plane faces complex requirements to ensure that diverse services can be delivered under tight latency bounds at scale.

Delivering new functionalities in a timely and customized manner demands a flexible user plane, which has led to the current softwarized packet core network design, where most packet processing is done on commodity servers in software (LAKE *et al.*, 2021). However, such flexibility comes with several drawbacks. Firstly, softwarized packet processing has difficulties maintaining low and predictable latencies due to several bottlenecks that are intrinsic to the hardware design of modern servers, such as limited and varying PCIe bus transfer speeds, cache misses, and other phenomena (NEUGEBAUER *et al.*, 2018), which makes it difficult to maintain stable packet processing latencies at high load. Secondly, softwarized packet processing on commodity servers typically relies on kernel bypass to process packets at user space using frameworks such as the DPDK (INTEL, 2010), which fully utilizes CPU cores for constantly polling the NIC for packets as shown in Figure 3. While reducing the latency, such polling wastes CPU cycles and leads to high energy consumption, significantly increasing operational costs.

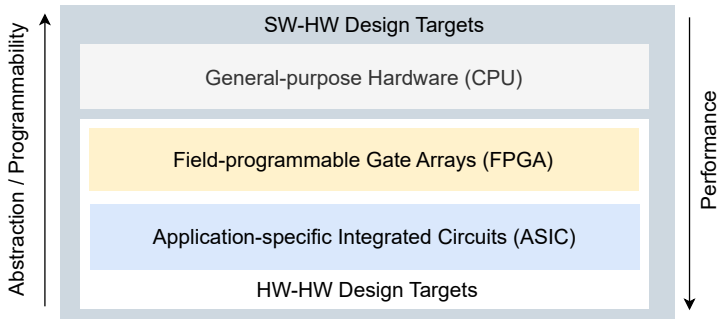


Figure 9 – Performance and programmability trade-offs.

3.1.2 Leveraging P4 Programmability and Recent Efforts

Recent efforts in the P4 (BOSSHART *et al.*, 2014) community provide support for a diverse range of various targets, i.e., ASICs, FPGAs, and x86 while providing flexibility to offload network functions to leverage the performance capabilities of different targets. Figure 9 shows the trade-off between programmability and performance. For example, x86-based packet processing provides the highest degree of programmability and flexibility at the expense of low throughput, high and unpredictable latency, and increased operational costs in CPU cores and energy consumption. On the other end, ASICs provide predictable latency at a line rate with a limited degree of programmability. FPGA, on the other hand, offers more programmability compared to ASIC with good performance characteristics.

To leverage emerging P4 programmable devices discussed above, most state-of-the-art (SINGH *et al.*, 2019) (SHAH *et al.*, 2020) compiles the user plane to P4 specific targets such as programmable ASIC, SmartNIC, or x86, but this approach limits the performance and functionality of the P4 program (DANG *et al.*, 2017) (KFOURY *et al.*, 2021). For example, the P4 programmability and available resources such as stateful memory or match action table entries of a switching ASIC such as Tofino are limited. On the other hand, compiling a P4 program to a SmartNIC may leverage its micro-C-based programmability to implement more complex external functions. While the SmartNIC may have abundant resources, the performance may also be impacted, e.g., large table lookups. Finally, cryptographic operations, packet buffering, and retransmissions, which

are required to implement the lower layer packet processing within a 5G dataplane, may require the P4 program to be compiled to the x86 due to the needed flexibility for external function processing. To leverage the target specific features, some recent efforts (ABHIK *et al.*, 2021) (MACDAVID *et al.*, 2021) split the UPF and offload unsupported functions, such as buffer services or hierarchical QoS from Tofino switch to x86 or SmartNIC. However, their solutions are either unable to answer how traffic allocation between targets impacts the overall performance or only focus on the specific target's performance.

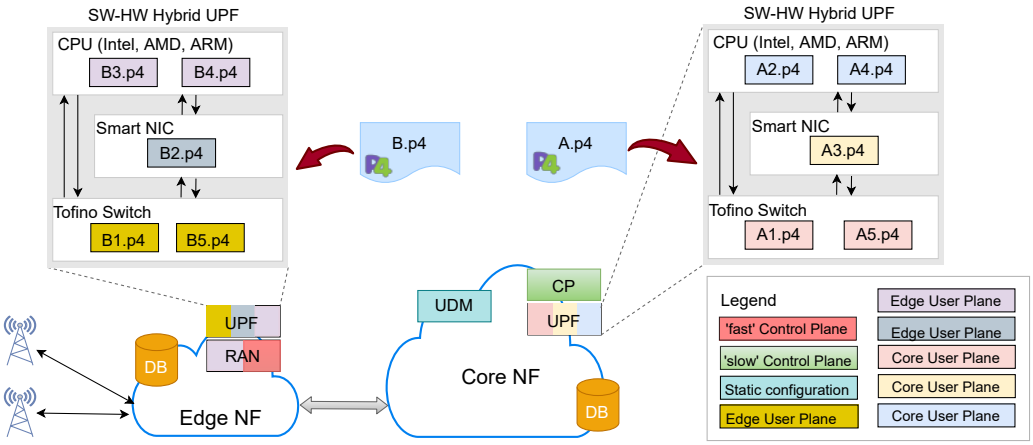


Figure 10 – 5G mobile network to show the disaggregation of a single P4 and run on different targets.

3.1.3 Our Proposed Solution

In this chapter, we propose *hybrid design approaches*, that P4 packet processing pipelines be split up into their basic processing units (SULTANA *et al.*, 2021), individually compiled, and run on different, possibly heterogeneous, targets to enhance overall performance. The proposed dataplane disaggregation approaches to split the functionality by carefully considering the mobile packet core packet processing requirements to combine each target's strength. The packet processing pipeline (Figure 10 is disaggregated into small sub-programs and flexibly deployed from the edge to the core of the network across different targets. For example, assume that the program `A.p4` can be split into five sub-programs. `A2.p4` and `A4.p4` could be run on the x86 to utilize the high degree of programmability, functionality, and available resources while observing the target latency and throughput. The sub-program `A3.p4` could be run on SmartNIC to exploit the more predictable latency

while releasing CPU cycles. Finally, 5G user plane functionality implemented in `A1.p4` and `A5.p4` that require limited programmability and resources could be run on Tofino switch ASICs. The same approach can be applied for edge network user plane functions. For example, the `B.p4` is split based on similar logic to `A4.p4`. Also shown in Figure 10 but not relevant to the scope of this work are the Unified Data Management (UDM) static configuration functions interfaced by 'slow' 5G CP functions to access user data network profiles and handle access authorization and registration. The 'fast' CP functions refer to low-level radio-related radio resource management per-user basis or control spectrum on a system level.

Our hybrid design models support diverse targets, ASIC, SmatNIC, and x86, focusing on *5G UPF*, but applicable to other datapath pipelines. We perform an extensive experimental evaluation to assess our proposed design model.

3.2 Related Work

The section discusses the related work on P4 hybrid designs focusing on mobile networks.

3.2.1 Programmable Hardware Acceleration

Offloading network functions running on general-purpose Commercial-Off-The-Shelf (COTS) servers (e.g., x86) to the hardware switch can provide two significant benefits. First, network functions can perform with ultra-low latency and high throughput. Second, due to big data analytic applications or other data center computations, energy consumption can be reduced (NESHATPOUR *et al.*, 2015). As mentioned in the previous section, there are various network P4 programmable hardware accelerator platforms; this chapter mainly focuses on x86, SmartNICs, and ASICs.

The P4→NetFPGA (IBANEZ *et al.*, 2019) provides an environment and workflow to use FPGAs as a switch to run P4 programs using the Xilinx P4-SDNet toolchain. In (RICART-SANCHEZ *et al.*, 2018), 5G backhaul is implemented on the NetFPGA P4 pipeline in two stages: Parsing and Match/Action. The proposed design supports

the double encapsulation required to communicate between 5G edge and core network within the multi-tenant 5G networks. The evaluation results have shown a lower packet loss from 49% to 4% in the worst-case to provide QoS in network congestion. Similarly, in (RICART-SANCHEZ *et al.*, 2019)(WANG *et al.*, 2019), a 5G-based QoS-aware network slicing solution is performed on P4-NetFPGA to fulfill the SLA requirement in terms of end-to-end latency and bandwidth. Not only offloading the user plane functions but (SHAH *et al.*, 2020) redesigns mobile packet core and offloads a significant fraction of signaling procedures from the CP to P4-programmable hardware (Netronome Agilio SmartNIC) or software switch (BMv2).

The performance of NetFPGA/SmartNIC lies in between a commodity x86 server and switch ASIC. However, some network applications require very high-performance, e.g., very high throughput and ultra-low latency, which are difficult to achieve using NetFPGA or any SmartNIC. For the solution, many works have been proposed where network functions are offloaded to switch ASIC. For example, in (SINGH *et al.*, 2019), the virtual Evolved Packet Gateway (vEPG) in the 4G access network is designed and evaluated on Tofino switch ASIC. The evaluation results show a low end-to-end latency of less than 2 μsec. Also, in (KUNDEL *et al.*, 2022), 5G UPF is performed under four different approaches as a device under tests, such as a kernel and userspace implementation on servers with dedicated NIC or virtual R-IOV ports and P4 based UPF implementation on Tofino switch. The results confirm that the P4 switch performs far better than other approaches. Apart from the UPF implementation, complex algorithms such as enhanced Content Permutation Algorithm (eCPA) (LIN *et al.*, 2019) can be executed on Tofino switch to encode/decode up to 6.4 Tb/sec to protect the 5G packets.

3.2.2 P4 Hybrid Datapaths

In the above discussion, we focus mainly on the existing works which offload network functions either on switch ASIC or P4NetFPGA. However, network applications can be disaggregated and run on different platforms based on the application requirements to

utilize the benefits from different targets (KATTA *et al.*, 2016). Recently, a disaggregated UPF (SILVA *et al.*, 2018)(WHITEPAPER, 2020) has been proposed commercially. Specifically, the P4 program is optimized and split into multiple components. Then, each component runs on different targets based on the complexity of function and application requirements. For example, running the same P4 based 5G UPF on an FPGA can scale the number of sessions from hundreds of thousands to millions compared to switch ASIC.

Similarly, in the ongoing Aether project (ONF, 2020) (MACDAVID *et al.*, 2021), which provides mobile connectivity and edge cloud services for distributed enterprise networks at ONF, two P4-based UPF designs are presented: (1) model-UPF and (2) performant-UPF. The Performant-UPF has been implemented on Tofino switch ASIC. Also, supporting other functionalities such as packet buffering or hierarchical QoS to provide isolation for each UE, which are limited with the current versions of Tofino switches, are executed on P4 based SmartNIC or x86. The model-UPF is developed as an open-source model data-plane, written for the Behavioral Model version 2 (BMv2) software switch to simplify the interface with the CP.

As mentioned above, the HW switches can support QoS up to some extent with limited queues (e.g., only eight queues per port in Tofino 1). However, handling the rate limit for QoS might be difficult because of the unsupported packet buffering features in HW switches to prevent packet dropping. In (ABHIK *et al.*, 2021), the UPF is performed on Agilio CX 2x10GbE SmartNICs. The packets which are required to be handled by buffer service are sent to the software. However, this work fails to show the overall impact of processing packets through different paths compared to SW or HW-based implementation. Finally, instead of UPF offloading, in (KUNDEL *et al.*, 2021), an open-source implementation of a BNG is executed on NetFPGA and also performed on Tofino switch ASIC. The packet queuing and scheduling are performed on NetFPGA.

Our work on hybrid designs for 5G UPF implementation differs from the existing works as follows. Compared to related work on P4-based UPF, our Hybrid-UPF effectively utilizes the target-specific features based on vertical/horizontal split of packet processing

pipeline and runs the entire or disaggregated UPF on different P4 targets to make UPF more flexible and programmable at high-performance and scale.

3.3 P4 Implementation Challenges

A limited number of ingress parsers per pipeline push their Packet Header Vector (PHV) (a collection of containers that carry the headers and matadata from the parser through the match-action units to deparser) into the ingress match-action pipeline. Suppose packet headers for parsing are longer than the ingress parser limit. In that case, the packet must be recirculated, and then the parser has to parse the remaining packet headers during the second phase. This directly impacts the performance of the switch. The precious PHV resources, a limited collection of bits, are allocated to match-action ingress and egress pipes based on the P4 program.

Each stage can have a fixed number of SRAM and TCAM tables, and each table can use a fixed number of bits for a single match. Although a table can span multiple stages, the number of stages also poses a limitation. So due to the limited SRAM and TCAM per stage and the limited number of stages, the size of big tables - such as the UE lookup table - is limited. We hit the maximum table size to perform UE matches during the 5G UPF execution on Tofino HW. For improving scalability, we propose a hybrid approach.

Furthermore, arithmetic operations such as multiplications and divisions or comparison two variables operations are generally not supported or limited by traditional switch ASIC. The emerging P4 programmable HW (e.g., P4 Tofino) supports simple bit operations. For complex computations, sometimes, we have to rely on the average calculations. This chapter uses the Heavy-Hitter detection algorithm based on Inter-Packet Gap (IPG) (SINGH *et al.*, 2020), where the Exponential Weighted Moving Average (EWMA) is performed. Since EMWA is difficult to compute accurately in Tofino switch HW, we depend on the average EWMA calculation. To execute such HH algorithms (SIVARAMAN *et al.*, 2017)(BASAT *et al.*, 2020)(YANG *et al.*, 2018) in programmable HW, we

use registers to keep flow states, e.g., 5-tuple flow Id, counters. In such cases, we mostly hit the limitation to access the register memory. For example, when a packet enters the switch, it can access only a few addresses in the register memory because of per-stage timing demand. Also, the incoming packet accesses a register instance only once in its lifetime. Such limitations can force to re-circulate the packet for re-accessing memory based on algorithm requirement (BASAT *et al.*, 2020).

In addition, to guarantee low and bounded per-packet latency, P4 programmable HW contains a fixed number of stages. To fit the given P4 program within the available stages, we require to avoid unnecessary table dependencies. We face this issue in executing the 5G UPF functionalities in parallel with the HH algorithm. Since the decision of packet routing is based on the output of the HH algorithm, it is hard to fit the hybrid design pipeline within the fixed number of stages (i.e., 12 Stages in Tofino). We optimize the code to efficiently use the table dependencies to fit the proposed design in the available stages. However, the upcoming P4 programmable switch ASIC (WHITEPAPER, 2020) might have extra features, including the number of stages to overcome these issues.

3.4 General Architecture of Hybrid Pipeline

Figure 11 shows the general architecture of hybrid UPF. Switch ASIC is connected with two different P4 targets: 1) x86 server behaving as a P4 SW switch, and 2) SmartNIC. The incoming packets at switch ASIC can be routed in three different paths based on the application's requirement. Three different routes have been shown in Figure 11.

In the fast path, incoming packets pass through NFs running on the P4 programmable switch ASIC and are sent to the output port. For slow path, packets can be routed through SW switch running on x86. For the slow path, there may be different options also on terms of hardware and programmability support on the x86. For example, if the host is equipped with a P4 programmable smart NIC, (parts of) the processing pipeline or parts of the overall traffic can be completely or partially processed by the SmartNIC (right side of the Figure 11). The remaining parts of the processing pipeline

or user traffic can be processed in the CPU. If a standard NIC is available (left side of the Figure 11), the NIC forwards user traffic to the x86 (e.g., through a software switch). Then, the remaining traffic or parts of the pipeline can be processed on the CPU using frameworks such as T4P4S (VöRöS *et al.*, 2018), which integrates more general packet processing with software switch functionality or using, e.g., a P4 compiler that compiles from P4 to DPDK code directly.

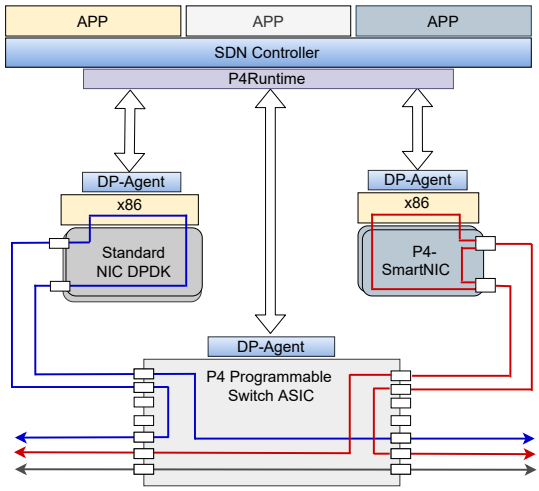


Figure 11 – General architecture view of hybrid dataplane pipeline design options.

There are two main approaches to partition the packet processing pipeline. In the first method (horizontal split), the pipelines have identical logic, but parts of the user packets are processed on each back-end, e.g., SmartNIC or host, observing the latency and capacity constraints of the different users (e.g., low latency traffic will be processed inside the programmable SmartNIC while parts of the high capacity user traffic will be processed at the x86). This has the benefit that low latency traffic that is completely processed in SmartNIC does not need to transit PCIe bus and is completely kept inside SmartNIC.

In the second method (vertical split), the logic of the pipeline is split into basic functional blocks; thus, processing the pipeline requires multiple processing backends to process each packet. Each backend will process parts of the pipeline (e.g., the SmartNIC performs a GTP de-tunneling operation while the host does table lookups and header rewriting). Using this approach, we need to consider the architectural constraints for

hardware acceleration (e.g., a crypto operation should be run on a device with effective crypto support or a crypto chip). Also, this approach follows the Service Function Chain concept but is more challenging to realize due to the consideration of different hardware capabilities that need to be considered for allocating the individual atomic processing tasks.

3.5 Hybrid UPF Design with P4 Switch ASIC and x86

3.5.1 Limited Resources and Scalability

Tofino ASIC is a high-performance P4 programmable switch that can run up to 6.5 Tbit/sec speed. VNF can be offloaded to the Tofino switch ASIC for achieving high performance. VNF offloading depends on the complexity of the functions because of the limited resources and flexibility of programmable hardware to guarantee low and bounded per-packet latency. Therefore, scalability for UE can be a significant issue for the 5G UPF running on programmable switch HW without compromising the performance. As given in Table 1, we perform a UE IPv4 address match with GTP encapsulation as an action for downlink UPF. In Tofino, there is limited SRAM memory per stage to keep the hash entries to perform the match. Instead, telecom operators require the limited and precious resources in the programmable switch ASIC for crucial control functions such as Access Control List (ACL) rules, customized forwarding, and other network applications. Thus, we are allowed UE matches in thousands only, while the UEs can be in a million, especially when the 500 billion mobile devices are expected to be connected to the Internet by 2030 (CISCO, 2016).

3.5.2 Forwarding Pipeline based on Heavy-Hitter

To overcome this problem, we propose a hybrid design based on HH detection. In real network traffic, the majority (i.e., 90-95%) are inactive UEs (i.e., non-HHs, low throughput), while the minority (i.e., 5-10%) are active UEs (i.e., HHs, high throughput) at a time

(ROSTAMI *et al.*, 2018). Also, most of the traffic in the network is contributed by HHs. Therefore, the HHs can be maintained on the Tofino switch to consume fewer hardware resources. The rest flows (i.e., non-HHs) can go through the x86 server without degrading the Key Performance Indicator for the 5G network. The proposed hybrid design leverages the IPG based HH detection algorithm to reduce the control channel overhead and for higher detection accuracy (SINGH *et al.*, 2020)(P4-HH, 2022a). In this case, the switch relies on a push-based approach to report the flow as HH to the controller for fast reaction.

3.5.3 Design Overview

As shown in Figure 12, we offload the entire UPF pipeline to the Tofino switch while the GTP Encap table is maintained in both x86 and Tofino. For the uplink, the packet goes directly through the Tofino switch. In the case of downlink, the packet is routed based on the HH detector and UE address match (i.e., GTP hit/miss). Suppose the packet is detected as HH with GTP hit. In that case, the first packet is sent to the ONOS controller to install the corresponding table entry to the switch, and the rest of the packets of that flow follow the HW pipeline to exit. At GTP hit for non-HH flow, entry is removed from the table, while packet goes through the x86 server for GTP hit to perform the match on UE address.

3.6 Experimental Evaluation

In this section, we experimentally evaluate our hybrid pipeline design presented in the previous section using real-time traces and artificial workloads. We aim to answer the following questions:

- How do hybrid approaches improve the scalability of P4-based 5G dataplanes?
- Which and how much are the P4 Tofino hardware resources consumed by our hybrid 5G UPF pipelines?

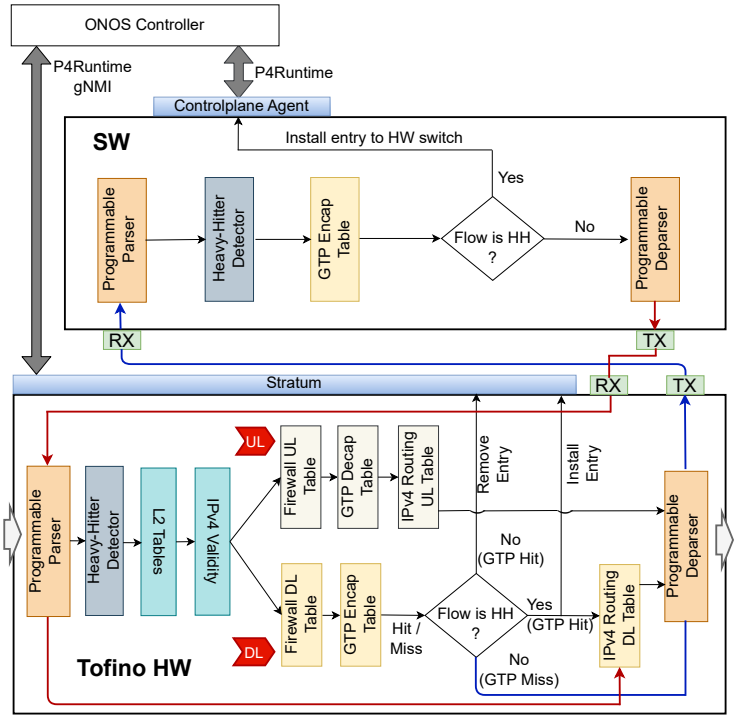


Figure 12 – P4-based hybrid Tofino-HW/x86 architectures for UPF.

- What is the performance (throughput and latency) of our hybrid UPF for different target combinations, varying UE sessions, and HH/offloading configurations?
- Which QoS improvements in terms of Flow Completion Time are possible in our hybrid UPF implementations?

3.6.1 Hybrid-UPF (Tofino + x86)

3.6.1.1 Methodology

We analyze the performance using a *Barefoot Tofino switch ASIC* (Edgecore Wedge 100BF-32X) and x86 server with Intel Xeon D-1518 processor (4 CPU cores, 2.20 GHz) for the UPF application generated by the MACSAD compiler for ODP-DPDK x86 targets (PATRA *et al.*, 2017). Tofino is connected with an x86 server with 10G SFP+ interfaces, as shown in Figure 12, for offloading traffic based on the HH detector running on Tofino. We use the NetFPGA-SUME based OSNT (GIANNI *et al.*, 2014) as a traffic generator running on an x86 server (Intel Xeon D-1518) with 10G SFP+ interfaces connected to the Tofino switch DUT. For pushing the required table entries related to the UPF and

HH detector, we use the ONOS controller connected with Tofino switch supported to P4Runtime. In the case of UPF, running on x86, we use the MACSAD controller to push the required entries.

We use CAIDA-2016 (CAIDA, 2016) ISP backbone link traces for evaluation. The traffic traces are available for four different days, and each is around 60 Secs long and contains around 20 Million packets and 800K flows. To analyze the performance for different flows or UEs, we split or concatenate the traces and prepare 100K, 800K, 1M, and 2M flow traces. Since the DDR3 memories of OSNT NetFPGA-SUME are 4 Gbytes, we can load a maximum of around 2M flows to replay at a 10Gbps rate.

3.6.1.2 Resource Utilization

For the programmable switch ASIC, the P4 pipeline of hybrid-UPF (H-UPF.p4) is around 1000 lines of code. We add IPG based heavy-hitter (HH-IPG.p4) functional block to the UPF P4 pipeline for hybrid design. H-UPF.p4, running on the Tofino switch, classifies the incoming packets based on HH and routes the non-HH flow packets to the x86 server to match UE IPv4 address and GTP encapsulation for achieving higher scalability.

Table 3 – Resource usage in Tofino switch ASIC

	UPF-Tofino.p4		HH-IPG.p4	H-UPF.p4	
Resource	**10K** (%)	**430K** (%)	**NA** (%)	**10K** (%)	**430K** (%)
Exact Match Input Xbar	**0.8**	**1.6**	**5.0**	**5.9**	**6.5**
Hash Bit	**1.8**	**4.5**	**2.8**	**8.6**	**12.8**
Hash Distribution Unit	**0.0**	**0.0**	**29.2**	**29.2**	**29.2**
Meter ALU	**0.0**	**0.0**	**10.4**	**10.4**	**10.4**
SRAM	**1.9**	**32.0**	**1.5**	**3.9**	**34.4**
TCAM	**0.0**	**0.0**	**4.2**	**6.3**	**2.1**
VLIW Instruction	**1.8**	**3.4**	**4.7**	**6.8**	**7.8**
Exact Match Result Bus	**1.6**	**3.1**	**8.3**	**10.4**	**12.0**

To analyze the resource consumption on the Tofino switch, we focus on different resources shown in Table 3. The exact match input crossbar is used for choosing input keys for exact match lookup, while the hash bit is for table lookup. Hash distribution unit is utilized to map hash output to PHV containers without using any table lookup.

Meter ALU and SRAM are used for stateful memory and exact match lookups and action tables, respectively. As shown in Table 3, we notice that all resources are used less than 5% for UPF-Tofino, but SRAM is around 32% for exact match lookups of 430K flows. For HH-IPG, 29.2% of the hash distribution unit is used because of hashing used to index the register and table and compute the flow ID. Therefore, adding the HH functional block to the UPF-Tofino for hybrid design shows a significant difference only for the Hash distribution Unit compared with UPF-Tofino.

3.6.1.3 Scalability

Offloading 5G UPF to the Tofino HW provides high-performance; however, it is hard to scale the number of UEs because of the HW resource limitations. In our proposed hybrid design, we can significantly scale the number of UEs with high-performance. For evaluation, we increase the table size, where we perform the matches on UE IPv4 address in Tofino and try to hit the maximum possible SRAM utilization. However, to evaluate the maximum number of UEs in Tofino, we must carefully optimize the H-UPF P4 code to manage the table dependencies to utilize maximum SRAM. When we add the HH-IPG functional block to the UPF-Tofino, the conditional statement can be a cause to keep HH-IPG and GTP Encap table in separate stages, which reduces the SRAM utilization. We optimize the code and try to keep both HH-IPG and GTP Encap table without any conditions, which allow the UE IPv4 matches and action in parallel to HH-IPG.

Table 4 – Scalability analysis with UPF-Tofino and H-UPF

	UPF-Tofino	**H-UPF(Tofino+x86)**
SRAM	60%	34.4%(Tofino)
UEs	850K	430K(Tofino)+15M(x86)

Table 4 presents the maximum SRAM usage with the number of UEs for UPF-Tofino and H-UPF. The UPF-Tofino usages 60% of SRAM with a maximum of 850K UEs, while H-UPF for Tofino utilizes 34.4% of SRAM with a maximum of 430K UEs treated as HH. SRAM 100% utilization is limited by table dependencies and other switch resource

Table 5 – The number of Heavy-Hitter per second for different threshold ranges, where "f" represents the total number of flows and "n" is the total number of packets. We get around the same number of HH flows per second for f $\geq$ 500K.

	HHs/sec (f=100K, n=1M)				HHs/sec (f=500K, n=9M)			
ISP Trace	**≥2 Mbps**	**≥5 Mbps**	**≥10 Mbps**	**≥20 Mbps**	**≥2 Mpbs**	**≥5 Mbps**	**≥10 Mbps**	**≥20 Mbps**
CAIDA 2016/01/21	632	186	56	23	513	107	26	13
CAIDA 2016/02/18	612	162	46	16	509	102	19	10
CAIDA 2016/03/17	646	193	62	27	528	119	35	16
CAIDA 2016/04/06	648	196	60	26	518	117	32	17

consumption. For practical cases, 430K HH flows would be sufficient for 10-100Gbps links. As shown in Table 5, the number of HH flows is less than 700 in all the given threshold ranges for CAIDA-2016 traces captured on the 10G link. For non-HH flows to keep on x86, we insert up to 15M flows to H-UPF running on x86 using the MACSAD compiler. However, we can keep more than 15M (e.g., 64M, 5 tuple keys) that fits in the CPU cache depending on the server memory specifications.

3.6.1.4 Performance

The performance of UPF is evaluated in terms of throughput and end-to-end latency. We analyze the throughput in the worst-case scenario where packets are smaller in size (payload is removed from all packets because of an anonymized dataset). To measure throughput, we make 3 cases: (1) UPF is running entirely on the x86 server (UPF-x86), (2) UPF offloaded to Tofino switch (UPF-Tofino), and (3) HH flows on Tofino, and non-HH flows go through the Tofino and then x86 server to perform UE address match and encapsulation (H-UPF). These all three cases are evaluated for a different number of UEs and 4 Days of CAIDA-16 traces. In addition, H-UPF is also analyzed for different HH threshold ranges.

As shown in Figure 13(a), as expected, UPF achieves a maximum of 2-3 Mpps, while UPF-Tofino is 3xtimes faster than x86 for 100K and 800K UEs. While H-UPF performance lies between them and reaches up to 6-7 Mpps. Since UPF-Tofino hits the limitation to run a maximum of 850K UEs discussed in the previous section, we evaluate the performance of

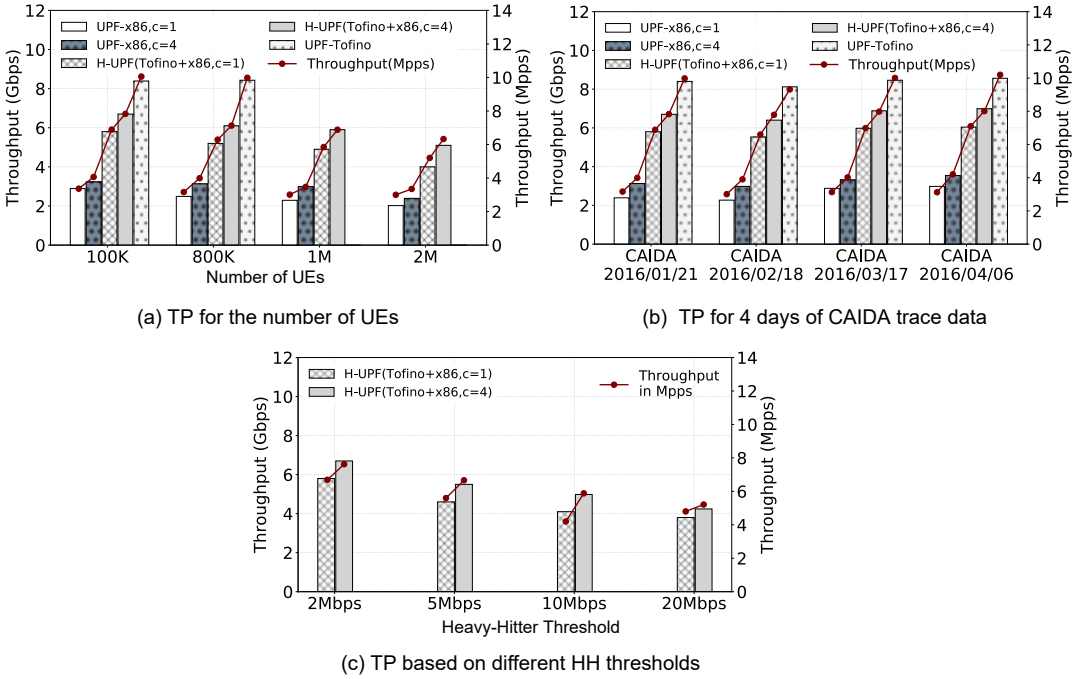


Figure 13 – Throughput (TP) comparison among UPF on x86 (UPF-x86), Tofino (UPF-Tofino) and hybrid UPF on Toinfo+x86 (H-UPF). For H-UPF in Figure (a) and (b), Heavy-Hitter threshold is set to 2 Mbps. Here, "c" denotes the number of CPU cores utilized by the MACSAD compiler to run UPF on an x86 server.

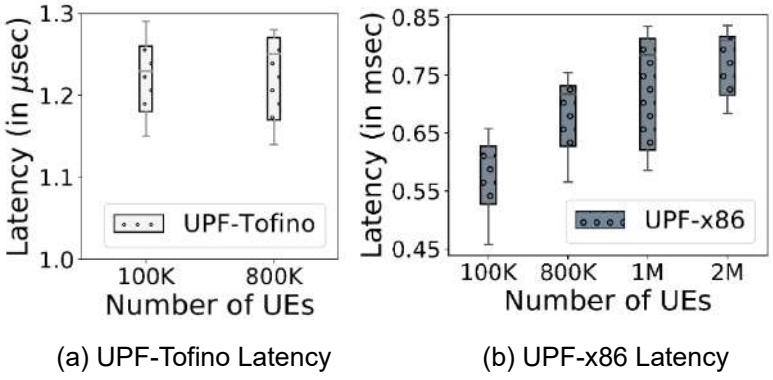


Figure 14 – End-to-end latency for UPF-Tofino and UPF-x86.

UPF-x86 and H-UPF for 1M and 2M UEs. H-UPF performs 2xtimes better than UPF-x86. We also note that the number of entries to perform exact matches impacts the performance.

Figure 13(b) shows the performance using CAIDA-2016 traces of 4 different days. However, we do not see any significant changes in the performance. In Figure 13(c), H-UPF is analyzed for different HH threshold ranges. For better understanding, we make Table 5, showing the number of HH flows per second for different ranges of HH thresholds (i.e., 2, 5, 10, and 20 Mbps). For 2 Mbps, Tofino keeps around 600 HH flows, while other

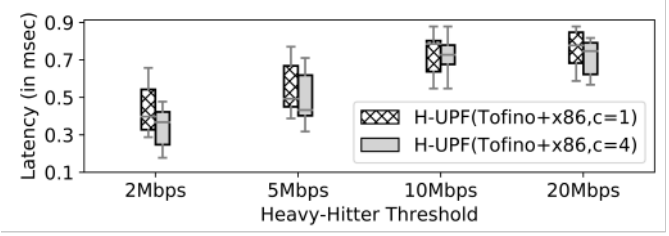


Figure 15 – End-to-end latency of H-UPF for non-HH flows at different threshold ranges.

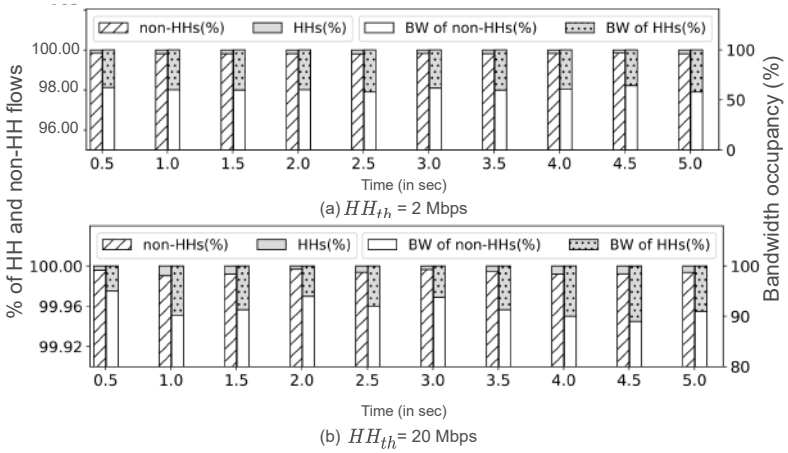


Figure 16 – Percentage of HH and non-HH flows and their impact on link bandwidth using CAIDA-16 traces for HH threshold 2Mbps and 20Mbps. The same behaviour of flows has been seen for long duration interval.

flows go through the x86 server. Since 600 flows can have two or more than 2 Mbps flow speed, H-UPF improves throughput by around 3 Mpps. Also, we notice that H-UPF has less improvement in throughput for higher HH thresholds because of less number of HH flows (i.e., 10-25 flows of 20 Mbps threshold).

For end-to-end latency, we use OSNT to send packets at line rate and calculate latency at the receiver end when the link is saturated. As shown in Figure 14(a) and (b), as expected, UPF latency is around 1.1 to 1.23 microseconds on Tofino HW. However, for x86, latency is between 0.45 to 0.85 msec. We also notice that latency is affected by increasing the number of flows because of the exact match lookup. For H-UPF, we analyze the impact on latency for non-HH flows passing through Tofino and then x86 to match the UE IPv4 address. As shown in Figure 15, offloading around 600 HH flows to the Tofino in case of 2Mbps threshold, we observe lower latency of non-HH flows because of reduction in the x86 bottleneck, but for other threshold values, we do not see any changes.

3.6.1.5 Runtime Flow Classification and HW Offloading

We classify the incoming traffic based on the HH algorithm running on the Tofino hardware and then re-route the packet depending on the algorithm decision. To understand the percentage (%) of HH flows offloading on runtime and their overall bandwidth occupancy, we use CAIDA-2016 ISP backbone link traces (CAIDA, 2016) for analysis on a microsecond scale. We replay the CAIDA traces at line rate (i.e., 10 Gbps) and analyze the number of HHs and their bandwidth occupancy.

As shown in Figure 16, the analysis is performed on the two different HH thresholds values, i.e., 2 Mbps and 20 Mbps. Table 5 provides more insights on the number of HH values based on the total number of flows per sec for other threshold values. In Figure 16, for both the threshold values, only a few HHs utilize 10-40% of the link bandwidth. This confirms that handling heavy flows within the programmable switch ASIC, where the resources are limited, would be a better choice to make the network more scalable with high-performance.

3.7 Concluding Remarks

This chapter explores a hybrid design model for 5G UPF, combining the features of different P4 targets to make the user plane more flexible and high-performing. Each proposed hybrid approach has its trade-offs and seeks to overcome P4 language limitations and challenges towards realizing softwarized 5G dataplanes. We split the entire pipeline and compiled it on different P4 targets, namely x86 and Tofino ASIC. The extensive experimental evaluation shows that hybrid user planes are an attractive approach to meet feature requirements and improve overall scalability with high throughput and low latency. The following are potential directions for future hybrid UPF research:

- Tofino ASICs are suitable for offloading heavy traffic flows if the number of rules is relatively small. This way, programmable HW pipelines are suitable for running a simplified user plane (e.g., for user equipment not requiring complex treatment) or

offloading specific users or flows selectively based mainly on their intensity. SmartNICs are in between smart switches and CPUs when it comes to performance. So, although they are less limited in terms of memory and thus the number of rules that could be handled, they can handle much fewer heavy users. Their main tasks would be to act as a protection and load-sharing layer in front of the CPUs.

Besides that functionality, they could also run some user plane functions similar to Tofino. Still, the number of users or flows could be smaller due to their limited bandwidth. Another option could be to run more complex user plane rules in SmartNICs, entirely offloading those users from the CPUs. The current study did not cover this option, as the SmartNIC used for the tests could not handle that high complexity. As new SmartNICs are expected to be available on the market in the coming years, the offload architecture might also evolve.

- To carry out the proper placement of functions, a critical aspect is to have adequate strategies to select which flows or users are handled by hardware pipelines and the software counterparts. We leverage a HH technique based on inter-packet gap metrics that runs entirely in the P4 dataplane at affordable resource costs with high accuracy and timeliness. In the future, refinements of the HH detection module and other candidates' HW/SW splitting techniques and slicing mechanisms should be explored.

- Based on our experiences, including knowledge and technology transfer to business units of telecommunication equipment providers, and analyzing the observed trends of the networking community, we believe that hybrid dataplanes are likely to become common in P4-like programmable dataplanes for different 5G flavors and other networking domains. One final concluding remark would be to expect scale-down (smaller fan-out) versions of Tofino ASICs to fit better the needs of distributed edge in addition to edge-optimized servers equipped with SmartNIC as programmable networking and compute targets for 5G and beyond.

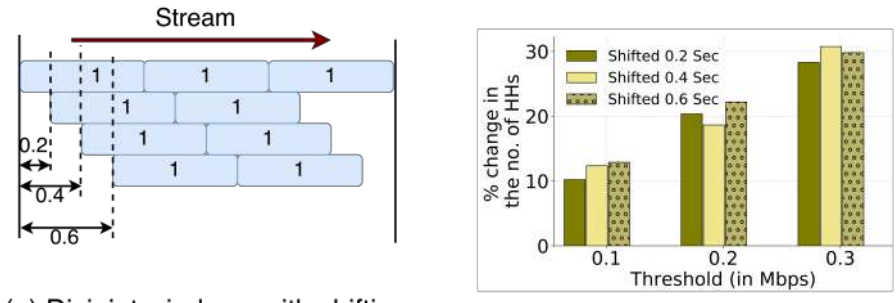
4 Heavy Hitter Detection in the Data Plane

4.1 Background and Motivation

4.1.1 Heavy-Hitter flow estimation in the Data Plane

Many network management applications benefit from identifying the flows contributing the most to the traffic, i.e., elephant flows or HH. This is the case, for example, for accounting (ESTAN; VARGHESE, 2002), network capacity planning (FELDMANN *et al.*, 2000), load balancing (AL-FARES *et al.*, 2010), caching (ROTTENSTREICH; TAPOLCAI, 2017), or anomaly detection (LAKHINA *et al.*, 2004). The recent uptake of programmable switches (Intel, 2017; CAVIUM, 2014) has given the means to move the detection process directly in the dataplane, enabling use cases requiring decisions at time scales comparable to traffic variations (BENSON *et al.*, 2011), such as dynamic routing (CURTIS *et al.*, 2011) or flow scheduling (AL-FARES *et al.*, 2010). The research community has lately worked hard towards the design and development of new algorithms that can efficiently detect heavy flows within the constraints of programmable devices (LIU *et al.*, 2016; SIVARAMAN *et al.*, 2017; YANG *et al.*, 2018; YANG *et al.*, 2019; BASAT *et al.*, 2020; KUCERA *et al.*, 2020a).

Most state-of-the-art proposals divide the time in disjoint windows and identify the HHs at the end of each of them by looking at the flows that consume more than a fraction of the link capacity. This is done with an iterative process at fixed time scales: flow counters or specific data structures, e.g., sketches, are first exported to a central controller and then reset in the dataplane to deal with the new traffic (LIU *et al.*, 2016; SIVARAMAN *et al.*, 2017; YANG *et al.*, 2018; YANG *et al.*, 2019). The intrinsic problem with this approach is that the final result is tightly coupled with traffic patterns and window characteristics. Figure 17 exemplifies this concept. Here, the *yellow flow* would not be considered *heavy* because neither Time Window (TW) A nor B has enough occurrences. To overcome this,



(a) Disjoint windows with shifting of 0.2, 0.4 and 0.6 Sec

(b) Compare the changes in the number of HHs

Figure 18 – Analysis of the number of HH flows due to the micro variations in the window using CAIDA traces.

recent proposals have explored more complex traffic analysis mechanisms in the dataplane by employing an approximation of a sliding window (CHEN *et al.*, 2019) or prefix trees that quickly adapt to traffic patterns (KUCERA *et al.*, 2020a).

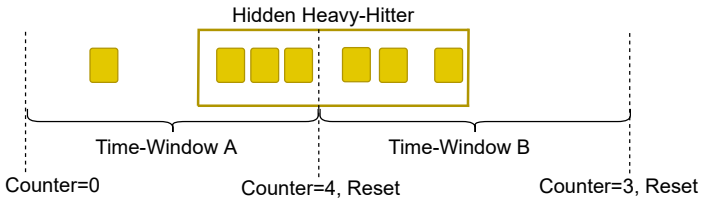


Figure 17 – Counting packets over disjoint windows.

To meet the strict requirements imposed by current programmable switches, most of the state-of-the-art proposals divide the time in disjoint windows and export data plane statistics to be used to detect HH, e.g., raw counters, sketches, at the end of each of them (LIU *et al.*, 2016; SIVARAMAN *et al.*, 2017; YANG *et al.*, 2018; YANG *et al.*, 2019).

Such architectural approaches suffer from the problem discussed in Figure 17. To illustrate this, we performed a number of tests using real traffic traces from CAIDA. Specifically, we used a 60-second long trace (CAIDA, 2016) and split it into 1-second intervals. We evaluated the number of HH flows using different thresholds and used this as a reference point. We then did the same by shifting the beginning of the analysis to 0.2, 0.4, and 0.6 seconds (Figure 18a). In Figure 18b, we compare how much the HHs detected have changed: the results show that micro variations in the window can impact the final result by 10% to 30%.

To better understand how much this variability can impact a network application,

we tested Hedera, a state-of-the-art intra-datacenter load balancing solution (AL-FARES *et al.*, 2010). The idea behind Hedera is to opportunistically schedule HH on different paths and let Equal-cost Multipath (ECMP) routing dealing just with the short flows. Starting from the open-source code[1], we implemented a fat-tree topology (k = 4) and generated traffic following many different communication patterns, the same used in the Hedera paper. Figure 19 shows the impact on bisection bandwidth when different percentages of HH flows are not detected. Most importantly, we can see a performance degradation up to approximately 30% when only 15% of HH are not correctly detected.

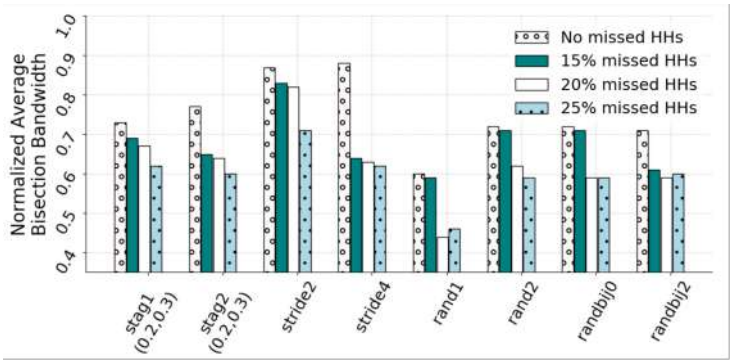


Figure 19 – Impact analysis of missing HHs for load-balancing in Data Center Networks using Hedera Algorithm (AL-FARES *et al.*, 2010), for different communication patterns.

4.1.2 Inter Packet Gap as a main metric

Instead of counting packets belonging to each flow, we propose to use IPG as the main metric to detect HH. The idea comes from the observation that the lower the IPG, the higher the flow throughput (LIU *et al.*, 2005): the smaller the flow's IPG, the more packets in-flight, and the heavier the flow. Notably, this simple philosophy cannot be naively applied to counters, i.e., the bigger the flow counter, the heavier the flow. This is because a flow is commonly defined as *heavy* only for a *given* amount of time. While we would lose the time reference by adopting only counters, with IPG, this is naturally considered. Figure 20 illustrates the main difference between the counter and IPG based approaches.

[1] <https://bitbucket.org/msharif/hedera/>

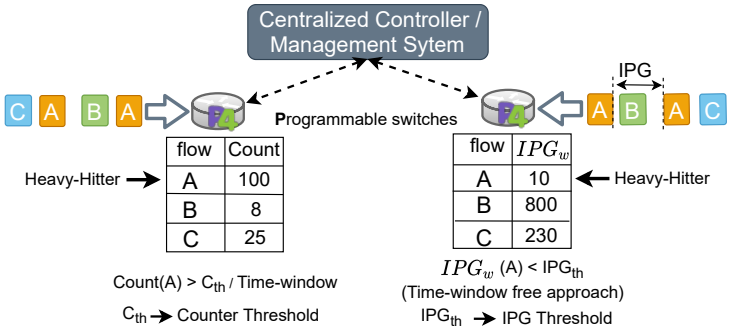


Figure 20 – Detecting Heavy Hitters: comparison between using counters and IPG as the main metric. IPG_w indicates the weighted IPG.

All the problems mentioned earlier could be solved by adopting sliding-windows-based algorithms in the dataplane. However, those algorithms are too complex to be implemented in off-the-shelf programmable switches (BASAT *et al.*, 2016; BASAT *et al.*, 2018), or successfully applied when considering micro-second level timescales events, i.e., microbursts (CHEN *et al.*, 2019).

This chapter demonstrates that there is no need to completely rethink existing dataplane algorithms to solve the mentioned issue. Instead, the most critical aspect to consider is the primitive adopted for HH detection. While previous works have focused on *counting packets* and optimized *data structures* to store the gathered data in the switch, we argue that IPG is a better metric for the HH detection problem. We show that it is possible to rethink state-of-the-art algorithms with this new metric and demonstrate that this can be done while meeting two fundamental properties discussed below.

Property 1: Low control channel overhead. State-of-the-art solutions leveraging dataplane programmability for HH detection are based on a simple concept: the switch collects traffic statistics, stores them inappropriate data structures, and then exposes them regularly to a (logically) centralized controller (SIVARAMAN *et al.*, 2017; LIU *et al.*, 2016; YANG *et al.*, 2018) in charge of the detection process. This behavior can impose unnecessary load on the CP, especially when multiple switches report their information to the same controller. Recent works have tried to mitigate this problem by proposing a push-based approach (KUCERA *et al.*, 2020a). The switch exposes only a single entry upon detecting a HH instead of letting the controller poll the entire data structure stored in

the dataplane. However, while successfully implemented on FPGA, the proposed solution appears to fail short in meeting the tight requirements in the number and the pattern of accesses to on-chip registers to be considered a viable solution for today's programmable switches (KUCERA *et al.*, 2020a).

Property 2: Low switch memory overhead. The previous properties cannot come at the cost of high memory requirements in the dataplane. This is because operators need the limited and precious resources available in current programmable switches for essential control functions such as ACL rules, customized forwarding (SIVARAMAN *et al.*, 2015), and other network functions and applications (MIAO *et al.*, 2017; JIN *et al.*, 2017).

4.2 Related Work

In this section, we discuss in detail the related works. IPG analysis has been employed in some networking applications (HAO *et al.*, 2011)(SUN *et al.*, 2015)(TANG *et al.*, 2019), but never to the HH detection problem to the best of our knowledge. In the past, the detection of heavy flows was performed outside the dataplane in software collectors. Network devices employ packet sampling and exported statistics using well-known protocols such as NetFlow (Cisco NetFlow, 1996) or sFlow (Cisco SFlow, 1996) to lower overheads and data collection bandwidth at the cost of estimation accuracy.

Recently, the uptake of programmable dataplanes has enabled the possibility to involve switches in the traffic analysis process. Specifically, several efforts have been proposed to count every packet belonging to every flow and storing this information in probabilistic data structures, i.e., sketches (XIAO *et al.*, 2020; LIU *et al.*, 2016; YANG *et al.*, 2018; FU *et al.*, 2020), to be implemented in the switch ASIC and exported at regular intervals to a central controller for further processing analysis. Some other proposals have suggested counting only on the heavy flows (METWALLY *et al.*, 2005; SIVARAMAN *et al.*, 2017; YANG *et al.*, 2019) to increase the detection accuracy as much as possible. The idea is to dynamically evict from a probabilistic data structure, stored in the switch ASIC, the contribution of short flows, thus minimizing as much as possible the effect of small

flows in the estimation of the large ones. In all cases, these counting algorithms rely on periodic reset of the switch's memory at the end of each TW. This makes those solutions prone to the problem shown in Figure 17.

Implementing a sliding-window based algorithm would circumvent the aforementioned issue. In this context, Memento (BASAT *et al.*, 2018), WCSS (BASAT *et al.*, 2016), SWAMP (BASAT *et al.*, 2017), and (TURKOVIC *et al.*, 2019) maintain the last n packets of a network stream to detect HHs. However, implementing those data structures considering the restriction of today's programmable dataplanes is not possible. Finally, other approaches decoupled from using windows by proposing a prefix tree can expand or collapse over time (KUCERA *et al.*, 2020a; KAMMENHUBER; KENCL, 2005). However, in this case, due to limitations in the number of register accesses, these algorithms are not feasible for today's programmable switches.

4.3 IPG analysis for HH detection

As discussed in the previous section, window-based solutions using counters rely on periodic resetting of the previous window's memory. This approach negatively impacts accuracy. In contrast, IPG allows keeping a flow's memory to make this approach more accurate than counter-based solutions. This section examines the IPG values of network flows using real-time traces to verify their significance for detecting HH flows. We consider an EWMA function of the observed IPG values and optimize the degree of weighting decrease to calculate the moving average for improved accuracy. In addition, we highlight the challenges of using IPG and discuss the proposed solution.

4.3.1 Weighting Inter Packet Gap

IPG directly relates to the flow rate (i.e., packet size / IPG). However, as per the standard HH definition (i.e., flows size larger than or equal to the number of packets or x kB in a fixed time interval), we cannot rely only on the current IPG value. For the solution, we consider an EWMA function of the observed IPG values. In EWMA, the weighting can

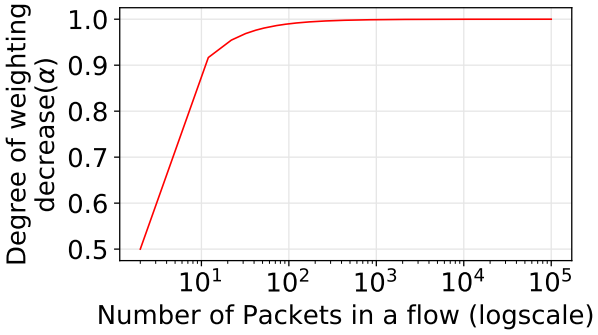


Figure 21 – Relation between α and the number of packets (n) of a flow when the SMA of IPGs equals to EWMA.

be set higher to decrease older observations slower. Also, the higher weighting smoothes out the sudden spikes in the IPG.

We consider a given set of network flows $f \in \mathcal{F} = \{f_1, f_2, ..., f_M\}$, where M represents the total number of flows. The IPG metric is updated every time a network packet of $f \in \mathcal{F}$ gets into the switch pipeline. When a packet enters the switch, the last seen ingress timestamp ($TS_f^l \in \mathbb{N}^+$) is subtracted from the current timestamp ($TS_c \in \mathbb{N}^+$) to calculate the current IPG_f^c estimator of network flow f (Equation 1).

$$IPG_f^c = TS_c - TS_f^l, \qquad f \in \mathcal{F} \tag{4.1}$$

Next, an EWMA metric IPG_f^w is computed:

$$IPG_f^w = \alpha \cdot IPG_f^{w-1} + (1-\alpha) \cdot IPG_f^c, \quad f \in \mathcal{F} \tag{4.2}$$

where $\alpha \in [0, 1]$ is the degree of weighting decrease and IPG_f^{w-1} is the last noted weighted IPG. IPG_f^{w-1} is initialized by IPG_{init} (i.e., $\frac{PacketSize \cdot 8}{HH_{th}}$), where HH_{th} indicates the HH threshold of a flow in Mbps in the given TW and packet size is in Bytes.

The α value directly impacts the accuracy of HH detection. For α optimization, a Simple Moving Average (SMA) is used as a reference point. In SMA, weightings are equally distributed among all the IPGs of a flow, directly related to a flow's size. On the other hand, older IPGs decrease exponentially for EWMA. Thus, we find an expression for the α, where the SMA of IPGs of a flow is equal to EWMA.

4.3.1.1 Optimization of degree of weighting decrease

Let's suppose the total number of packets in a flow $f \in \mathcal{F} = \{f_1, f_2, ..., f_M\}$ is $n+1$ and the IPG between two consecutive packets for each incoming packet are $IPG_f^{c,1}$, $IPG_f^{c,2}$, $IPG_f^{c,3}$, ... , $IPG_f^{c,n}$. For simplicity, we extend the definition of current IPG (Eq. 1), adding an extra superscript index to refer to the order of incoming packets (e.g., $IPG_f^{c,n}$ refers to the IPG between the n^{th} and $(n+1)^{\text{th}}$ packet). SMA is given as follows,

$$IPG_f^{\text{SMA}} = \frac{1}{n} \cdot \sum_{j=1}^{n} IPG_f^{c,j}, \qquad f \in \mathcal{F} \tag{4.3}$$

EWMA is given below,

$$IPG_f^{\text{EWMA}} = \alpha \cdot IPG_f^{\text{EWMA}-1} + (1-\alpha) \cdot IPG_f^{c,n}, f \in \mathcal{F} \tag{4.4}$$

where $IPG_f^{\text{EWMA}-1}$ represents the last noted EWMA and $IPG_f^{c,n}$ is the inter packet gap when $(n+1)^{\text{th}}$ packet enters. Suppose $IPG_f^{\text{EWMA}-1}$ is equal to the $IPG_f^{\text{SMA}-1}$ for $(n-1)^{th}$ packet. We can also write $IPG_f^{\text{EWMA}-1}$ as,

$$IPG_f^{\text{EWMA}-1} = \frac{1}{(n-1)} \sum_{j=1}^{n-1} IPG_f^{c,j}. \tag{4.5}$$

Now, we insert the value of $IPG_f^{\text{EWMA}-1}$ from Eq. 4.5 into Eq. 4.4,

$$IPG_f^{\text{EWMA}} = \frac{\alpha \cdot 1}{(n-1)} \sum_{j=1}^{n-1} IPG_f^{c,j} + (1-\alpha) \cdot IPG_f^{c,n}$$

For $IPG_f^{\text{SMA}} = IPG_f^{\text{EWMA}}$,

$$\frac{1}{n} \cdot \sum_{j=1}^{n} IPG_f^{c,j} = \frac{\alpha \cdot 1}{(n-1)} \sum_{j=1}^{n-1} IPG_f^{c,j} + (1-\alpha) \cdot IPG_f^{c,n}.$$

After simplifying the above equation, we get,

$$\alpha = \frac{(n-1)}{n} \tag{4.6}$$

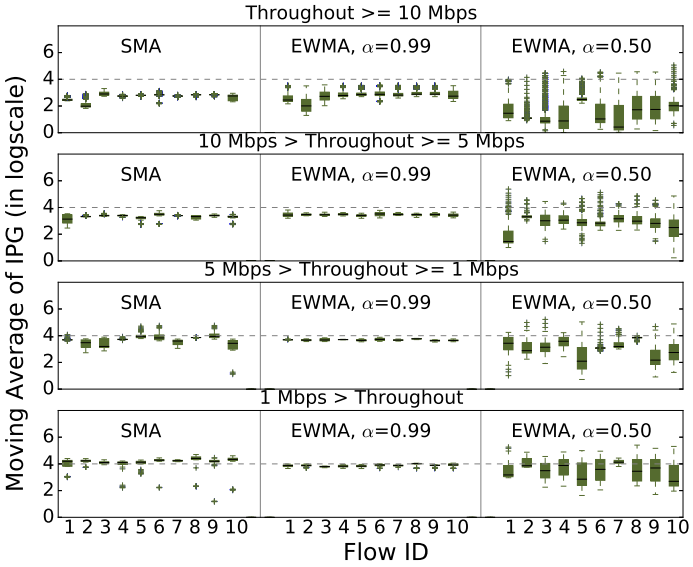


Figure 22 – Distribution of EWMA ($\alpha = 0.99$ or 0.50) and SMA of IPGs of 10 flows in different throughput scale.

As per Equation 6, α can be equal to or greater than 0.5. However, the α changes rapidly between 0.5 to around 0.98 for the first few packets, as shown in Figure 21 (plotted Equation 6). So picking these values would not be the right choice for HH detection. To validate this point, we plot a graph, as shown in Figure 22. We illustrate the distribution of EWMA and SMA of IPG of each incoming packet of 10 flows in different throughput scales using CAIDA trace for 1 Sec TW. We notice that EWMA at $\alpha = 0.99$ is more stable and similar to SMA than EWMA at $\alpha = 0.50$. Similar results are observed for *other real traces* such as data center (IMC10 (BENSON *et al.*, 2010)) and MAWI backbone (MAWI20 (MAWI, 2020)) traces. The above analysis suggests that α above 0.98 can be used for HH detection to get higher accuracy.

Also, we notice that for a small-duration TW (e.g., 1 Sec), where the number of packets per flow is lower than a long-duration time window, the $\alpha = 0.99$ is suitable for HH detection. However, for long-duration time-window (e.g., 10 Secs), we need to choose a higher α (e.g., around 0.999), which imposes additional challenges to calculate EWMA in the Tofino switch ASIC due to the floating-point operations required. Also, higher α yields a slower response to recent IPG values and can negatively impact accuracy on runtime. We propose a solution to handle long-duration TW discussed in detail in Section 4.3.2.

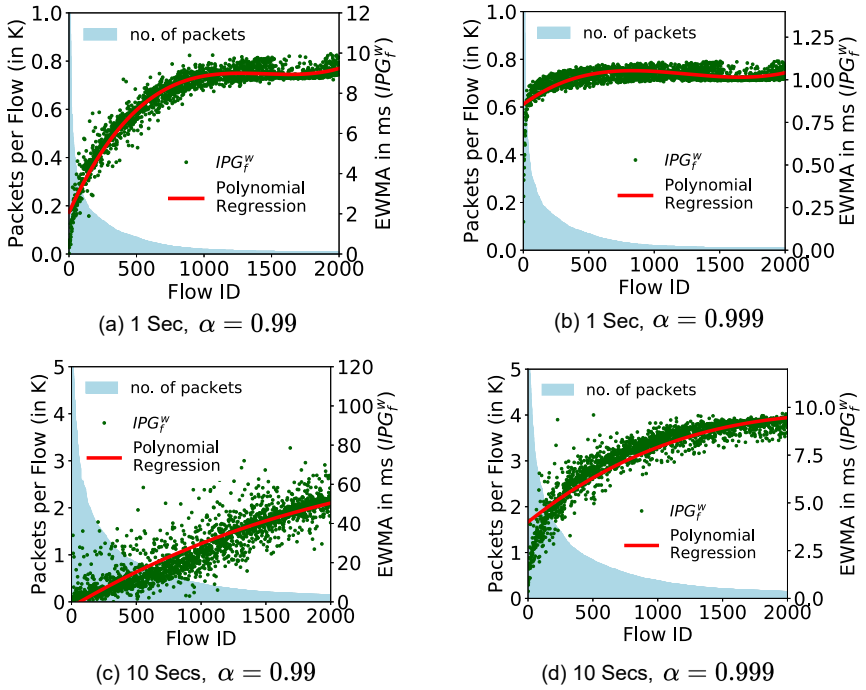


Figure 23 – IPG_f^w vs Throughput. IPG_f^w indicates EWMA of IPGs at the end of TW. We consider only long-duration top-2k flows for 1 Sec TW.

4.3.1.2 Weighted IPG and flow's throughput

We analyze the relationship between IPG_f^w and flow throughput (total number of packets per TW). We make two cases: (1) consider only *long-duration* flows (Figures 23), (2) examine both *small* and *long* duration flows (Figures 24). We use CAIDA traffic traces (CAIDA, 2016) and plot the number of packets per flow against the IPG_f^w calculated at the end of 1 sec TW, i.e., about 65K flows and 0.7M packets, and 10 Secs TW, i.e., around 270K flows and 5M packets. The analysis is performed for the top-2K flows. We do not observe much difference in the results for other traces (BENSON *et al.*, 2010)(MAWI, 2020).

Case I. Figures 23a and 23d show a strong correlation between IPG_f^w and flows throughput. If we set $\alpha = 0.999$ for 1 Sec TW, we observe a weaker response to recent IPG values (Figure 23b). In Figure 23c, where $\alpha = 0.99$, some flow's IPG_f^w does not correlate well with their throughput values because of the long TW duration.

Case II. In Figures 24a and 24b, considering small-duration flows with long flows, the throughput does not correlate well with IPG_f^w. This is because the lower IPG does not

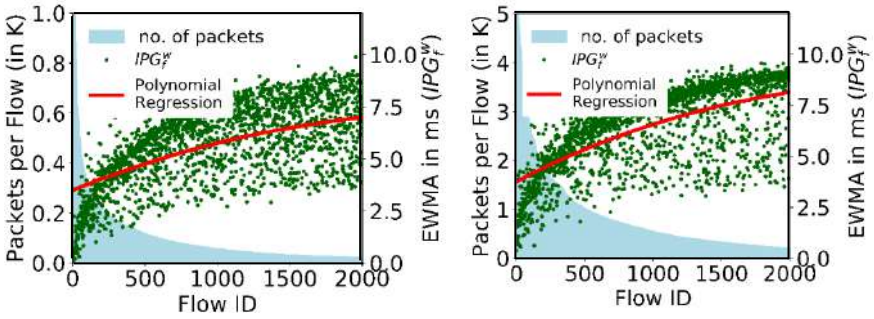


Figure 24 – IPG_f^w vs Throughput, considering both small and long-duration top-2k flows for 1 Sec TW.

guarantee a higher throughput. Specifically, the flow with only two packets back-to-back can be considered HH (i.e., false positive).

As per the above discussion, we conclude that while the initial analysis points to IPG_f^w as a candidate metric for HH detection considering long-duration flows, the main challenge is handling small-size flows of lower IPG. In the next section, we seek to answer the following questions: (1) Can we handle small-size flows, e.g., *flow only two packets back to back*, using IPG metric?, (2) How to deal with a *long duration* time window?

4.3.2 Memory of Flow's Characteristic

When relying on IPG instead of packet counters to detect HH, we encounter two challenges: 1) handling *small-size flows of lower IPG*, and 2) detecting HH flows for *long-duration* TWs as discussed in subsection 4.1. To overcome these challenges, we split a flow duration into small timeslots, and at the end of each timeslot (T_{wt}), we store the throughput state of a flow based on IPG_f^w, which is used to detect HH flows. Maintaining IPG_f^w within the P4 switch hardware decides to keep only lower IPG_f^w flows in the data structure, while the stored throughput will decide the flow is HH or not. In such a way, we do not require resetting the switch memory, and the stored throughput only reset once the flow turns into HH. This simple approach cannot be applied with the counter and must reset the previous window switch's memory.

As a consequence, the small-size flows are difficult to survive in the data structure for multiple timeslots by keeping the low IPG_f^w. Also, as discussed in Section 4.3.1.1,

α=0.99 performs well for small timeslots; we can use the same α value for long-duration TW to get high accuracy.

4.3.2.1 Storing flow's throughput state

To store a flow's throughput state, we introduce a non-dimensional metric tau (i.e., compactly maintained in the programmable hardware). IPG_f^w metric is used to keep the heavy flows in the data structure, whereas the τ_f metric, updated per-flow depending on IPG_f^w at the end of each timeslot, is used to decide whether the flow is HH or not. We use a match-action table, where the match is performed on IPG_f^w and pass τ_f as an action parameter to store in metadata and add this value in previously-stored τ_f in the switch register. The P4 TNA snippet is given as follows:

```
1
2       action setTau(bit<16> tau) {
3         meta.hash_meta.tau =  tau;}
4       table storeFlowTPState {
5       key = {meta.hash_meta.IPGw :  exact;}
6       actions = {setTau; nop;}
7       size = IPG_init;
8       default_action = nop;}
```

After completing the match-action, we update the τ_f register (i.e., rTau in the P4 code):

```
1
2       RegisterAction<bit<16>, rSize, bit<16>>(rTau) rTau_action = {
3         void apply(inout bit<16> value, out bit<16> readvalue){
4         if (value > TAU_TH) {
5         value = 0;
6         } else {
7         value = value + meta.hash_meta.tau;}}};
```

The complete P4 code is available at Github (P4-HH, 2022b). When τ_f reaches the pre-defined threshold (τ_{th}), the switch informs flow as HH to the controller for further

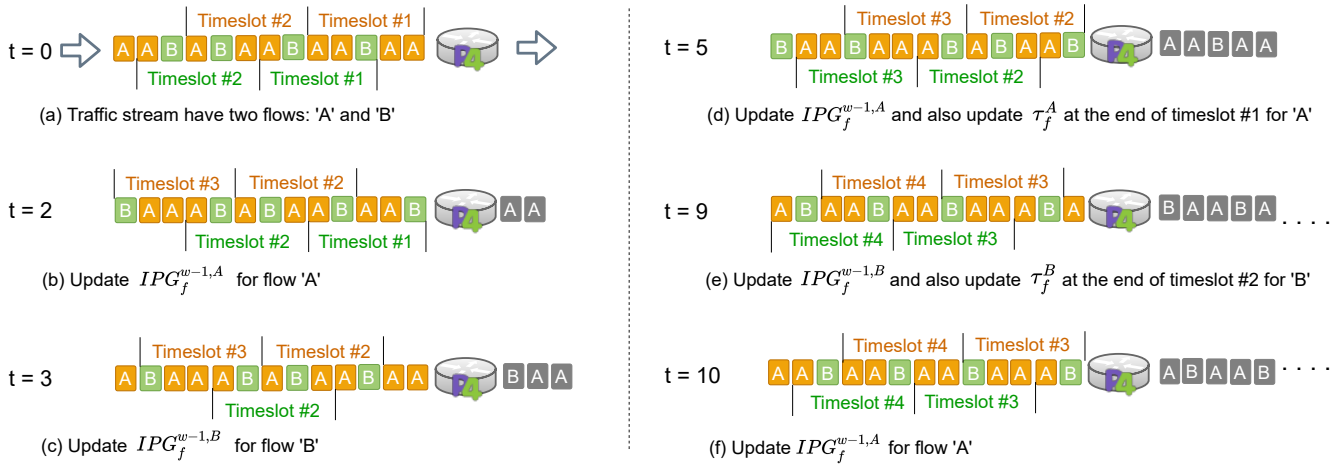


Figure 25 – Example of updating IPG_f^{w-1} and maintaining the throughput state of HH flows using τ_f metric.

action and reset τ_f to zero. For calculating τ_f, first, we compute the minimum number of packets (n) required for HH in a TW as follows:

$$n = \frac{TW \cdot HH_{th} \cdot 10^6}{PacketSize \cdot 8}, \tag{4.7}$$

We consider 1000 Bytes as a default packet size in this chapter. Then, we evaluate the number of timeslots (num_{wt}) required to achieve n packets based on IPG_f^w,

$$num_{wt} = \frac{n \cdot IPG_f^w}{T_{wt}}, \tag{4.8}$$

where $num_{wt} \geq 2$. We divide τ_{th} by num_{wt} to get τ_f. The final expression for τ_f is as follows,

$$\tau_f = \frac{\tau_{th} \cdot T_{wt} \cdot PacketSize \cdot 8}{TW \cdot HH_{th} \cdot IPG_f^w \cdot 10^6}. \tag{4.9}$$

The controller pre-calculates the τ_f based on IPG_f^w range 1 to IPG_{init} using Equation 4.9 and pushes the table entries to the switch. We should choose τ_{th} to generate the maximum distinct τ_f integer values for IPG_f^w. For example, if we set TW=1 Sec, HH_{th}=10 Mbps, and T_{wt}=20 msec, we can generate a maximum of 48 distinct τ_f values for τ_{th}=1850. If we choose τ_{th} lower than 1850 (e.g., 500), 33 distinct τ_f values can be generated, impacting the accuracy and reporting time.

Table 6 – Correlation between IPG_f^w or τ and flow size

	Flow Size (in Packets)				
[l]**Parameters**	**1-5 Mbps**	**6-10 Mbps**	**11-20 Mbps**	**20-30 Mbps**	**>30 Mbps**
$IPG_f^{w,\alpha=0.50}$	-0.13	0.063	0.22	-0.22	-0.15
$IPG_f^{w,\alpha=0.80}$	-0.21	0.02	0.23	-0.23	-0.12
$IPG_f^{w,\alpha=0.99}$	-0.82	-0.39	-0.03	-0.06	-0.11
$IPG_f^{w,\alpha=0.999}$	-0.88	-0.75	-0.53	-0.53	-0.54
$\tau_f^{\alpha=0.50}$	0.79	0.74	0.52	0.85	0.85
$\tau_f^{\alpha=0.80}$	0.86	0.83	0.63	0.86	0.86
$\tau_f^{\alpha=0.99}$	0.91	0.95	0.96	0.97	0.96
$\tau_f^{\alpha=0.999}$	0.86	0.94	0.95	0.98	0.95

We follow the standard definition of HH (i.e., number of packets in a fixed TW greater than or equal to HH_{th}) used in state-of-the-art (LIU *et al.*, 2016) (YANG *et al.*, 2018). If τ_f of any flow reaches τ_{th}, this indicates that the flow's throughput is HH_{th} or above in fixed TW.

4.3.2.2 Steps to detect HH flows

Figure 25 shows an example to update the τ_f and IPG_f^{w-1} to detect HH flows. There are two flows. At t=2, the switch calculates $IPG_f^{c,A}$ and updates $IPG_f^{w-1,A}$ for flow A. The same step applies for flow B at t=3. The switch observes the end of timeslot #1 for A at t= 5 and updates both $IPG_f^{w-1,A}$ and τ_f^A. We follow the same steps at t=9 for flow B. Before updating τ_f^A or τ_f^B, the switch compares them with τ_{th} and reports the flow as HH to the controller for τ_f^A or $\tau_f^B \geq \tau_{th}$. τ_f for lower IPG_f^{w-1} flows increases rapidly, and the switch can report the HH flows to the controller as soon as possible. E.g., as shown in Figure 25, flow A has lower IPG_f^{w-1}, i.e., higher τ_f^A than flow B. Flow A would be reported as HH earlier than flow B. The higher IPG_f^{w-1} can quickly be evicted from the data structure to maintain only heavy flows. The detail about the algorithm is discussed in Section 4.4.

4.3.2.3 Correlation between τ_f and flow size

To validate the hypothesis discussed above, we analyze the correlation between τ_f or IPG_f^w and flow size using ISP (CAIDA16), data center (IMC10), and WIDE backbone

(MAWI20) traces. Since the outputs from all three traces do not show a significant difference, we present the results using CAIDA in this research proposal. Also, analysis is performed for different throughput ranges to understand the impact of flow throughput on both IPG_f^w and τ_f metrics. We update the flow size and τ_f every 20 milliseconds, i.e., T_{wt} = 20 msec. As shown in Table 6, the τ_f highly correlates with flow size for α=0.99. At the same time, IPG_f^w does not correlate well with flow size because of small size flows with lower IPGs.

Also, we can observe that the correlation between τ_f and flow size decreases by decreasing flow throughput. We conclude with the following outcomes:

1. IPG_f^w and τ_f metrics allow keeping flow's throughput without any memory reset than maintaining counters, making this approach more accurate.

2. τ_f, a dimensionless metric, is used to keep flow's throughput dependent on IPG_f^w.

3. At α=0.99, τ_f highly correlates with flow size for HH detection to get high accuracy. Also, the same α value can be used for any duration of TW.

4. T_{wt}, a small timeslot, is used to update τ_f within the switch ASIC.

5. The small-size flows with low IPG values (e.g., flows of only two packets back-to-back) can be easily recognized as true negative using IPG_f^w and τ_f metrics.

6. The proposed method is independent of a flow duration.

Note: Our proposed approach uses a small timeslot (T_{wt}), different from a TW based methods. In a window-based approach, the previous window memory must reset at the end of each TW for all flows in the data structure. In contrast, T_{wt} is used to update the τ_f based on IPG_f^w for each flow maintained in the data structure; no memory reset is required. τ_f is reset to zero once the flow is detected as HH.

4.4 Design and Implementation

This section highlights the implementation challenges and design of the proposed method, discussed above, on the *Tofino switch* and adopted solutions. Then, we discuss limitations and corner cases.

4.4.1 P4 ASIC Implementation Challenges

P4 offers an exciting opportunity to run algorithms directly in programmable switch ASICs. However, implementing algorithms on switch ASIC brings some challenges (Figure 26). We discuss the engineering challenges illustrated in red boxes in Figure 26 and adopted solutions to implement a P4 pipeline of the proposed IPG based method validated in a programmable switch ASIC, as detailed in Section 4.5.

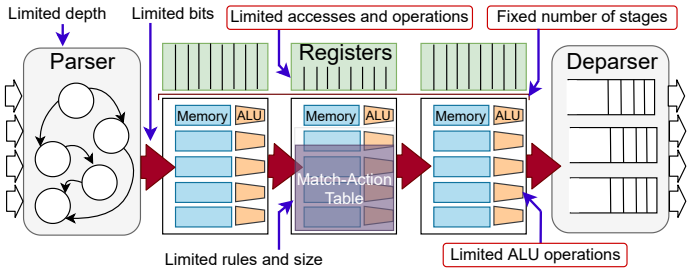


Figure 26 – Resource limitations in Programmable HW.

4.4.1.1 Access Limitations to Registers

Typically, in HH detection algorithms, first, the switch checks the flow ID of incoming packets that already exist in the register. Then, the related parameters are updated, such as the packet counter. In the existing HH algorithms (YANG *et al.*, 2019)(YANG *et al.*, 2018), we decrease the counter and evict the flow from the data structure for unmatched Flow ID once the counter reaches zero. Since we have already visited the flow ID register to confirm the match, we cannot re-access the flow ID register to replace the entry. The register can be accessed once per packet lifetime in Tofino switch ASIC. We face this challenge. As a solution, we rely on packet re-submission. Suppose the flow ID of incoming packets exists in the data structure (Algorithm 1: Line 7); the packets exit

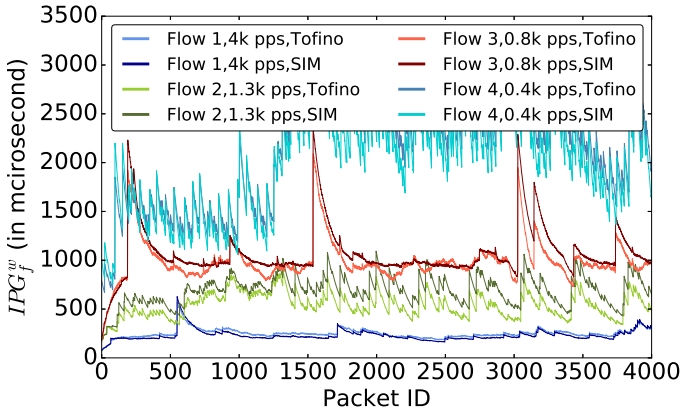


Figure 27 – Running IPG_f^w for Tofino and Simulator (SIM).

without any re-submission. The re-submission happens only for unmatched packets when $IPG_f^{w-1,i} > IPG_{th}$ (Algorithm 1: Line 19 to 22).

The same problem discussed in (BASAT *et al.*, 2020) comes up with packet re-circulation as a solution for unmatched packets. The two main observations are noticed in (BASAT *et al.*, 2020): (1) the expected number of recirculated packets are bounded by the square root of the number of packets which confirms the sub-linearity, (2) 1% of packets re-circulation impacted 1% on throughput. The same applies in our case to perform re-submission only for unmatched packets. The re-submission is performed efficiently within the switch ASIC pipeline and does not impact the performance much, as analyzed in Section 4.5. We use metadata to differentiate the re-submitted packet from the regular packets (P4-HH, 2022b). Also, the re-submit packet replaces the old with a new entry without any additional computation steps, as shown in Figure 29.

4.4.1.2 Arithmetic and Comparison Operations

Traditional switch HW do not support multiplications and divisions for register actions. In addition, comparison operations are limited to a fixed number of bits and can be performed between constant and variable values, not allowing the comparison of two variables. Recently available P4 programmable HW (e.g., P4 Tofino) supports bit operations that we can use to perform simple multiplications and divisions. As per Equation 5.2, EWMA calculation can be challenging to implement in an HW pipeline. Our solution is

based on an approximate EWMA calculation that makes all the required arithmetic and comparison bit operations with limited HW resources.

Figure 27 presents the run-time IPG_f^w values in the Tofino HW and Simulator (SIM) implementations for four different flow throughput rates. In SIM, we use the standard formula to calculate the EWMA, while in Tofino, we depend on some approximations due to the P4 HW constraints. Figure 27 shows that Tofino HW and SIM do not exhibit significant differences, confirming that IPG_f^w can be used to classify HH flows directly in the Tofino HW. For future steps, we will consider Tofino-specific extern, i.e., *low pass filter*, that allows us to calculate EWMA, giving a more accurate calculation of EWMA.

4.4.1.3 Fixed Number of Stages

To guarantee low and bounded per-packet latency, P4 programmable HW (Tofino) is based on a fixed number of pipeline stages. To fit our program within the available stages, we must avoid unnecessary table dependencies to perform our proposed algorithm. Currently, we need 12 stages to complete all the required steps. We run our algorithm on top of a baseline switch.p4 (Open-Source P4 Switch, 2017) compiled on Tofino (P4 Studio SDE 9.3.1) confirms that our algorithm can run with other switching functions without significant additional resources and stages constraints (Table 8).

4.4.2 Design on Programmable Switch ASIC

This section shows how to use existing data structures and algorithms with the IPG metric instead of packet count to detect HH flows accurately. In the proposed IPG method, the HH decision is based on the τ_f metric following the standard HH definition discussed in Section 4.3.2. Let $\mathcal{P} = \{P_1, P_2, P_3, ..., P_N\}$ be a network stream with N packets. Each packet $P_j (1 \leqslant j \leqslant N)$ belongs to a given network flow $f \in \mathcal{F} = \{f_1, f_2, ..., f_M\}$, where M is the total number of network flows in the network stream $\mathcal{P}$. Each flow $f \in \mathcal{F}$ has τ_f, where $\tau_f \in \tau = \{\tau_{f_1}, \tau_{f_2}, ..., \tau_{f_M}\}$ and τ is the set of throughput state of each flow in $\mathcal{F}$. Given a network stream $\mathcal{P}$, and the pre-defined threshold τ_{th}, a HH is a flow f where

$\tau_f \geq \tau_{th}$.

Our proposed method can fit most of the existing counter-based algorithms; we leverage the HeavyKeeper (HK) (YANG *et al.*, 2019) algorithm for our prototype implementation on a Tofino HW switch, which is amenable to programmable HW. We also analyzed our proposed approach with Space Saving (METWALLY *et al.*, 2005) and Elastic Sketch (YANG *et al.*, 2018) using a simulator. We achieved high accuracy, but due to space limitations, we focus on HeavyKeeper in this chapter. We use HK-IPG to indicate HK algorithm using IPG and HK-Count to show HK algorithm using counters.

4.4.2.1 Data Structure

HK-Count algorithm tries to keep the only heavy flows in the hash table, and smaller flow sizes are removed using an exponential-weakening decay scheme. When a packet enters, the main idea is if the flow ID is not the same as in the hashed table entry, HK decays the flow size by 1. When the flow size deteriorates to zero, the entry is replaced by a new flow with counter value 1.

We replace the counter with a weighted IPG (i.e., IPG_f^{w-1}) metric and apply the same scheme in reverse order. Instead of decaying the flow size, we increase the weighted IPG when the hash collisions occur. When the weighted IPG is higher than the predefined threshold, it is removed, and the new entry is inserted. In this algorithm, we choose IPG_{th} as a threshold for evicting entries from the data structure. IPG_{th} is set the same as IPG_{init}, discussed in Section 4.3.1.1.

4.4.2.2 Insertion

As shown in Figure 28, the data structure consists of a table (i.e., T = 1,2,...,m) associated with hash function h(.) that contains m slots. Each slot includes four fields: flow id, $IPG_f^{w-1,i}$ (last noted weighted IPG), $TS_f^{l,i}$ (last noted timestamp), and τ_f^i, where $i \in T$. For this example, we consider $IPG_{th} = 10000$ μsec. HK-IPG checks the condition $TS_c < TS_f^{l,i}$ for each incoming packet to confirm the end of T_{wt} (timestamp wraps around in

every T_{wt} (P4-HH, 2022b)). If it is true, τ_f^i is updated based on $IPG_f^{w-1,i}$ and $IPG_f^{c,i}$. If $\tau_f \geq \tau_{th}$ is true, switch reports the flow as HH to the controller. Specifically, there are three cases:

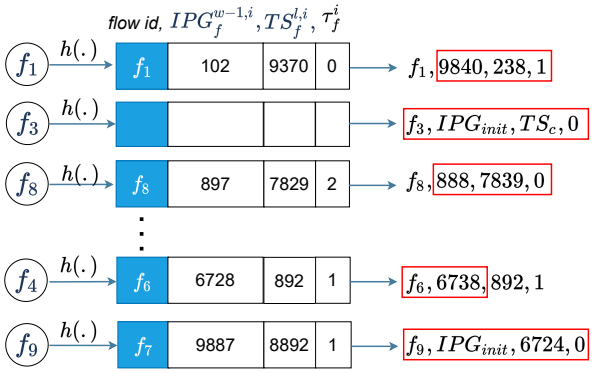


Figure 28 – The data structure using IPG metric.

Case I: The slot is empty. When the packet of f_3 enters the switch since the slot is empty, we insert $f_3, IPG_{init}, TS_c, 0$, where we initialize the $IPG_f^{w-1,i}$ with IPG_{init} (i.e., equal to IPG_{th}), TS_c is the current time-stamp, and τ_f^i is initialized with 0.

Case II: $f = f_i$. This condition occurs for f_1 and f_8. In this case, HK-IPG calculates the IPG and updates each field of the slot accordingly. In the case of f_1, τ_f^i is updated with one based on the $IPG_f^{w-1,i}$; however, for f_8, since the condition $TS_c < TS_f^{l,i}$ is not true, τ_f^i remains the same.

Case III: $f \neq f_i$. In this case, first, HK-IPG checks the condition $IPG_f^{w-1,i} \leqslant IPG_{th}$. If true, $IPG_f^{w-1,i}$ is increased linearly by adding k, which is pre-defined by the controller based on α and table size. In this example, we consider k=10. When the condition $IPG_f^{w-1,i} > IPG_{th}$ becomes true, the existing entry is replaced by the new entry. For f_4, ten is added in the field $IPG_f^{w-1,i}$. In the case of f_9, after updating the field $IPG_f^{w-1,i}$, the condition $IPG_f^{w-1,i} > IPG_{th}$ is true. HK-IPG replaces the entry with the new entry.

Figure 29 depicts the high-level view of the proposed P4 pipeline to detect HH flows. Algorithm 1 represents the detailed description of the HK-IPG algorithm and explains the three main steps for keeping the per-flow state.

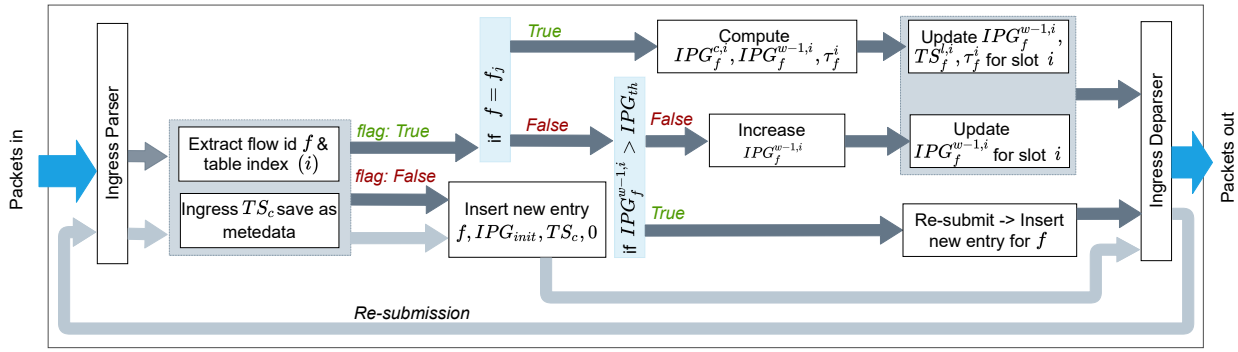


Figure 29 – The proposed P4 pipeline for HH detection using novel IPG metric implemented on *Tofino switch ASIC*.

4.4.3 Limitations and Corner Cases

Limitations. Currently, our solution requires at least two timeslots ($num_{wt} \geq 2$) discussed in Section 4.3.2 to detect HH flows, which makes undetectable short-lived HH flows that appear in a single timeslot. We can reduce the probability of missing these flows by reducing the length of the timeslot. However, in that case, we may lose the overall accuracy to detect HH flows.

Moreover, our approach may not detect HH flows with a specific pattern, where the inter-burst gap is more significant than the timeslot (T_{wt}) due to failure to check the condition: $TS_c < TS_f^l$. Thus, although we optimize the size of T_{wt} to lower the possibility of missing this type of HH flows, it is still possible to miss a few HHs. However, we can completely diminish this limitation with some memory expenses by keeping a flow's starting time-stamp and updating that at the end of each T_{wt}.

Corner Cases. Using the IPG metric for HH detection, there could be a case where a flow only contains two packets back to back (i.e., low IPG). *Do we treat such flows as HH using our proposed approach ?*. The answer is no. As we discussed in Section 4.3.2, to resolve this issue, we use multiple timeslots to decide the flow is HH or not. Also, we initialize IPG_f^{w-1} with the same value as IPG_{th}. If the flow has a lower IPG with only a few packets, there is a small probability of reducing IPG_f^{w-1} significantly. It is also unlikely that these types of flows survive in multiple timeslots with low IPG.

Since we choose a small timeslot (T_{wt}) to update the τ_f as discussed in Section 4.3.2,

Algorithm 1: Proposed Algorithm: HK-IPG

Input: A packet $P_j (1 \leqslant j \leqslant N)$ belonging to flow ID f.
Get table index i from $h(f)$, where $i \in T = (1, 2, .., m)$.
if *flag* = *false* **then**
    /* Case 1 */
    $flag \leftarrow true$;
    $f_i, IPG_f^{w-1,i}, TS_f^{l,i}, \tau_f^i = f, IPG_{init}, TS_c, 0$;
else
    if $f = f_i$ **then**
        /* Case 2 */
        if $TS_c > TS_f^{l,i}$ **then**
            $IPG_f^c = TS_c - TS_f^{l,i}$;
            $IPG_f^{w-1,i} = \alpha \cdot IPG_f^{w-1,i} + (1-\alpha) \cdot IPG_f^c$;
            $TS_f^{l,i} = TS_c$;
        else
            $IPG_f^c = TS_c + T_{wt} - TS_f^{l,i}$;
            $IPG_f^{w-1,i} = \alpha \cdot IPG_f^{w-1,i} + (1-\alpha) \cdot IPG_f^c$;
            $TS_f^{l,i} = TS_c$;
            match on $IPG_f^{w-1,i}$, set *metadata.tau* as an action;
            $\tau_f^i = \tau_f^i$ + *metadata.tau*;
        end
    else
        /* Case 3 */
        if $IPG_f^{w-1,i} \leqslant IPG_{th}$ **then**
            $IPG_f^{w-1,i} = IPG_f^{w-1,i} + k$;
        else
            $(f, IPG_f^{w-1,i}, TS_f^{l,i}, \tau_f^i = f, IPG_{init}, TS_c, 0)$;
        end
    end
end

there could be another corner case: *is the proposed approach limited to detect the HH flows for small length TW only?* Note that we use T_{wt} to update the τ_f metric for keeping the throughput state of a flow. There is no direct relationship between the timeslot and the length of the time window. The detailed discussion can be found in Section 4.3.2.

4.5 Evaluation

We analyze the performance of HK-IPG using the Tofino HW switch implementation with real traffic traces and compare it with the state-of-the-art. We detect HHs using different flow ID definitions such as 5-Tuple (i.e., source IP address, destination IP address, source port number, destination port number, and protocol number), source IP, and destination IP. Furthermore, four different measuring time intervals are investigated: 1, 5, 10, and 60 Secs.

4.5.1 Experiment Setup

Testbed. We evaluate the performance of our proposed IPG based HH detection method on a *Barefoot Tofino switch ASIC* (Edgecore Wedge 100BF-32X). We use the FPGA-based OSNT (ANTICHI *et al.*, 2014) and the DPDK-based TRex (TRex, 2024) as the traffic generators on an x86 server (Intel Xeon D-1518) with 10G SFP+ interfaces connected to the Tofino switch DUT.

Dataset. We use the CAIDA-2016 (CAIDA, 2016) ISP backbone link, IMC-10 (BENSON *et al.*, 2010) data center, and MAWI-20 (MAWI, 2020) WIDE backbone traces for evaluation.

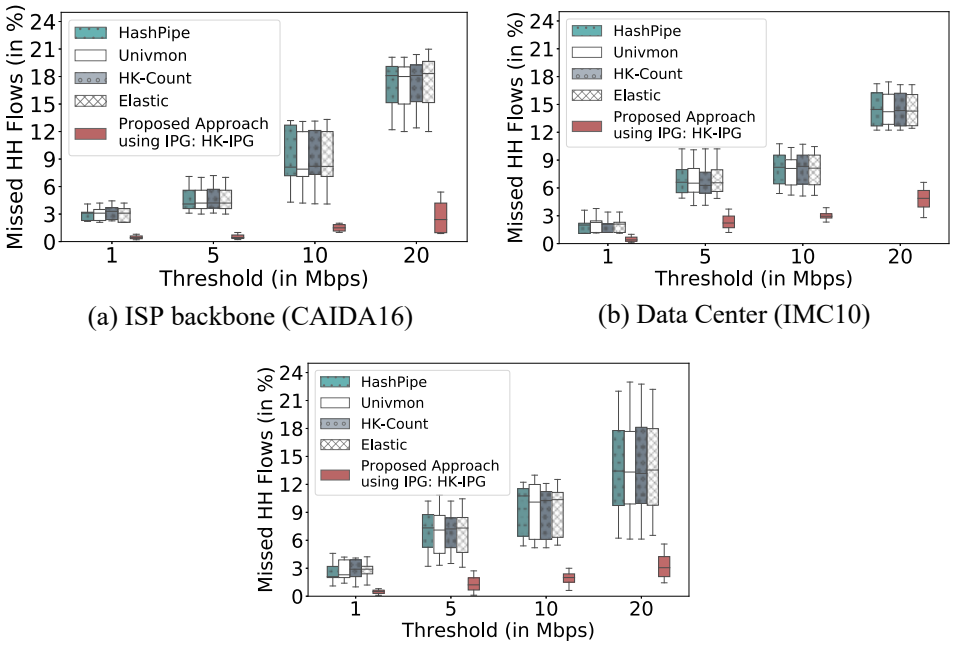
CAIDA16. The traffic trace is around 60 Secs and contains 20 Million packets and 800K flows. We split this trace into 1, 5, and 10 Secs small chunks. The 5 Secs chunk contains around 2 Million packets and 150K flows, while the 10 Secs chunk carries about 4.5 Million packets and 280K flows.

MAWI20. The WIDE backbone traces are 15 minutes long with 600K packets (4000 flows) per second.

IMC10. The data center traces are 20 minutes long with approximately 15K packets (200 flows) per second. We replay the packets at a higher rate (i.e., about 3500 flows per second), following the exact state-of-the-art (SIVARAMAN *et al.*, 2017).

4.5.1.1 Implementation.

For the programmable switch ASIC, the P4 pipelines of HK-IPG (i.e., around 500 lines of code (P4-HH, 2022b)) algorithm is implemented using the TNA. A single Table is maintained with m number of memory slots. Each slot contains four values: 32-bit flow ID, 16-bit IPG_f^{w-1}, 16-bit TS_f^l, and 8-bit τ_f. Total memory use is calculated by $m \times 72(bits)$. For setting the weighted IPG threshold, we assume a default packet size of 1000 Bytes. Also, the value of α is set to 0.99 for the EWMA calculation. Flows above 1 Mbps are considered as HHs. We get around the same results for other higher HH threshold values.



(a) ISP backbone (CAIDA16)

(b) Data Center (IMC10)

(c) WIDE Backbone (MAWI20)

Figure 30 – Comparison of the proposed algorithm in terms of missed HH flows due to memory resets after TW.

4.5.1.2 Evaluation Metrics.

(1) F1 Score, Recall and Precision: We define as True Positive (TP) those HH flows which are correctly detected, FP (False Positive) those flows which are non-HH but detected as HH, and FN (False Negative) the flows which are HH but detected as non-HH. We evaluate the Precision ($Pr = \frac{TP}{TP+FP}$) and Recall ($R = \frac{TP}{TP+FN}$) metrics, from which we derive the F1 score as $\frac{2\cdot(R\cdot Pr)}{(R+Pr)}$.

(2) Bandwidth: We measure the control channel overhead in bandwidth (kB) required per sec for different reporting time intervals.

(3) Throughput and Latency: We measure the computational complexity of our proposed algorithm in terms of throughput and latency on the Tofino switch with and without the proposed algorithm. The throughput defined in packets per second is measured with TRex, while latency is measured one-way end-to-end using OSNT.

4.5.2 Accuracy

4.5.2.1 Missed Heavy-Hitters using disjoint windows

As discussed in Section 4.1, in most of the existing disjoint time window-based algorithms, we can miss the true HH flows because of resetting the previous interval memory, as shown in Figure 17. In this section, we compare our proposed algorithm with the state-of-arts. We perform the simulator analysis since it is challenging to implement all the other state-of-arts on Tofino switch ASIC and some of the P4 source code's unavailability. We use the 60 Secs long *CAIDA 2016* trace(CAIDA, 2016) to detect the true HH flows for 1 Sec TW by applying the *sliding window* approach. The complete simulation and P4 code are available at GitHub (P4-HH, 2022b). To find the true missed HH flows due to resetting the counters in every 1 Sec TW, we split CAIDA traces into 1 Sec TWs and evaluate the true HH flows. Second, we apply state-of-art algorithms to detect the number of missed HH flows out of true HH flows due to reset counters. In the same way, we use our algorithm to detect the number of missed HH flows. The same steps are performed for *IMC10* and *MAWI20* traces.

Figure 30 shows that our proposed algorithm can detect most of the missed HH flows. Only 0.2-5.0 % of true HH flows are not detected by HK-IPG, while up to 20% of true HH flows are not exposed in the case of state-of-art algorithms.

Impact of missing HH flows. As discussed in Section 4.1, if we miss 15-25% of HH flows, around 30% of bisection bandwidth can be impacted for load balancing in data center networks. Also, we analyze the impact of HH detection on other real-time applications such as QoS for the mobile network, discussed in detail in Section 4.6.

4.5.2.2 Accuracy on Tofino Switch ASIC

We assess the impact of memory and the duration of measuring time-interval on precision and recall. Figure 31 presents the results of the HK-IPG algorithm on the *Tofino switch ASIC*. We observe that our proposed solution can detect HHs with around 0.92 precision and 0.95 recall with only 144 KB memory. We analyze the performance for different

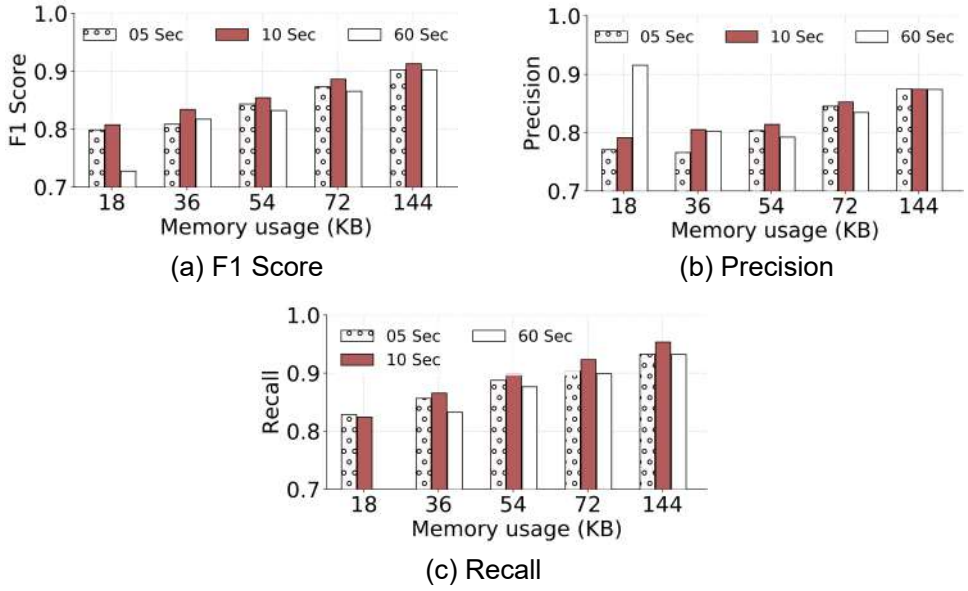


Figure 31 – Heavy-Hitter detection capability of HK-IPG Algorithm on *Tofino Switch ASIC* using CAIDA16.

measuring time intervals (i.e., 5 Sec, 10 Sec, and 60 Sec) and find that precision and recall values are consistent for larger memory sizes. For smaller memory sizes (i.e., 18 KB), the probability of evicting the HH flows from the data structure increases.

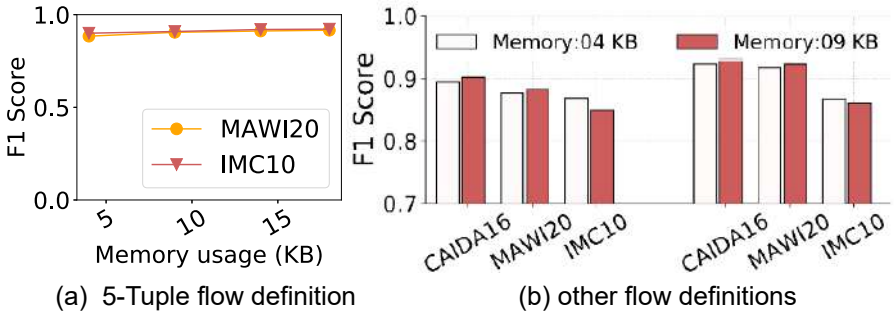


Figure 32 – Accuracy with IMC10 and MAWI20 traces.

Consequently, the total reported flows as HH are lower than the actual number of HHs because flows are unable to maintain τ_f above τ_{th} during their lifetime. For longer measuring time intervals, we can see this impact much more effectively. That is why recall is negatively hit low for smaller memory sizes, especially for 60 Secs, because of the higher number of false negatives. For larger memory sizes, the F1 Score value is above 0.90, confirming that precision and recall are balanced enough to provide high accuracy. Also, we evaluate our proposed algorithm with different flow definitions and other real traces, IMC10(BENSON *et al.*, 2010) and MAWI20 (MAWI, 2020), as shown in Figure 32.

We achieve high accuracy for both data center and WIDE backbone network traces.

4.5.3 Evaluation with other Performance Metrics

4.5.3.1 Bandwidth Utilization

Figure 33 compares the amount of data exchanged between a switch and a controller. We analyze the performance using CAIDA traces by setting 1 Mbps as the threshold for different reporting times. The switch reports the HH flows to the controller when they turn into the HH (i.e., push-based approach). It is confirmed from Figure 33 that our proposed algorithm can save a significant amount of CP traffic, around two orders of magnitude compared to state-of-the-art algorithms (pull-based approach).

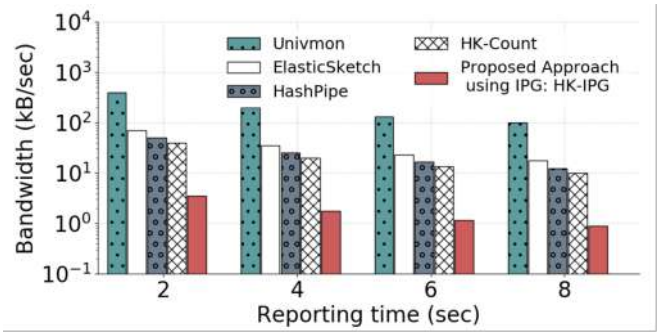


Figure 33 – Bandwidth Utilization

4.5.3.2 Computational Complexity

To examine the computational complexity of the proposed approach, we perform throughput (TP) and Latency (LAT) tests using CAIDA traces and compare them with state-of-art algorithms. Since only a few existing algorithms are implemented on P4 switch ASIC, and there is a lack of availability of TNA P4 codes, we choose HK-Count for comparison. First, we measure TP and LAT without using any algorithm (i.e., just output packet forwarding without applying any actions). In the second case, we measure the performance with the HK-IPG and HK-Count algorithms. As we can observe in Figure 34, the throughput is not affected in both cases, as expected from a line-speed hardware pipeline. The difference in the latency with and without the HK-IPG algorithm is 176 nanoseconds.

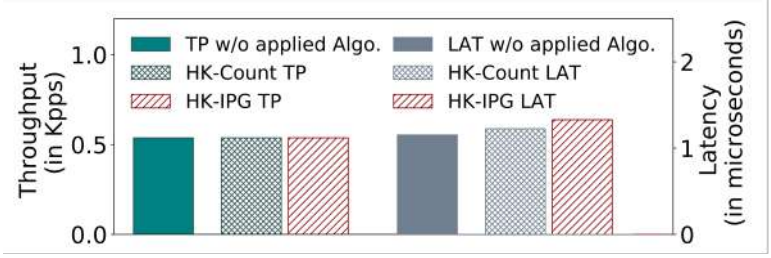


Figure 34 – Performance on *Tofino Switch ASIC*.

Table 7 – Additional HW *Tofino resources* used by HK-IPG, running on top of a baseline *switch.p4*.

Resource	Baseline (%)	HK-IPG with Baseline(%)	Additional usage(%)
Exact Match Input Crossbar	16.9	21.6	4.7
Hash Bit	20.1	28.5	8.4
Hash Distribution Unit	16.7	45.8	29.1
Meter ALU	12.5	22.9	10.4
SRAM	23.6	25.1	1.5
TCAM	21.9	22.2	0.3
VLIW Instruction	12.8	16.7	3.9
Exact Match Result Bus	18.8	27.6	8.8

4.5.3.3 Resource Utilization

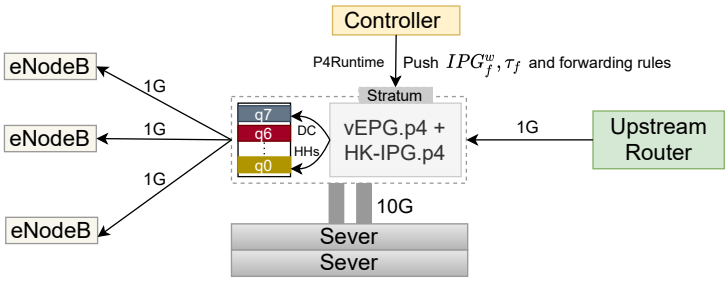
We observe no significant difference in maintaining per user state in the P4 switch ASIC compared with existing algorithms. As discussed in Section 4.5.1, for maintaining per user state in P4 switch ASIC, 72 bits are required for our proposed approach, while other existing algorithms generally need 32 bits for flow Id and 32 bits for counters. We run our algorithm on top of a baseline switch.p4 (Open-Source P4 Switch, 2017) (P4 Studio SDE 9.3.1) compiled on Tofino switch ASIC. In Table 7 (additional usage), we can see that all resources use around less than 10%, except for the hash distribution unit. Hash distribution unit is used to map hash output to PHV containers without using any table lookup. Stateful ALU is used in each stage to read and write operations in the register array. SRAM is used for storing the metrics of the HK-IPG algorithm.

4.6 Use Cases

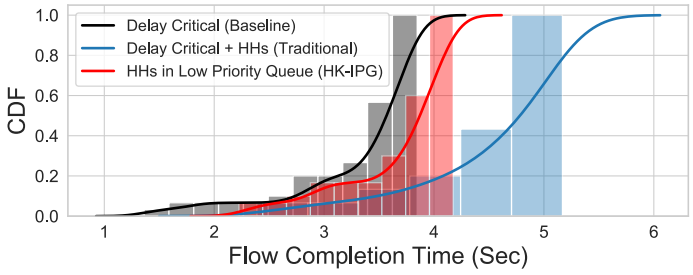
After HH detection, different traffic management and control actions are typically carried at different time scales, e.g., Data Center TCP (DCTCP) (ALIZADEH *et al.*, 2010), Distributed Denial-of-Service (DDoS) attack (LAPOLLI *et al.*, 2019), IntSight (MARQUES *et al.*, 2020).

4.6.1 Offloading Heavy-Hitters to Improve QoS.

To analyze the proposed approach, we evaluate the Flow Completion Time (FCT) of the pipeline configured to send all IPG detected HHs to lower priority queues. At the same time, delay-critical flows go to a higher priority queue.



(a) QoS-HH Use-case Scenario



(b) Flow completion time with and without HHs offloading to lower priority queue

Figure 35 – Mobile network scenario (a) and distribution of Flow Completion Times (b) of delay critical flows showcasing the gains when HHs detected by HK-IPG (red) are sent to lower priority queues compared to traditional pipelines (blue).

As shown in Figure 35(a), we consider a use case scenario of a mobile network implemented by adding our `HK-IPG.p4` functional blocks to a mobile gateway P4 pipeline implementation (`vEPG.p4`) (SINGH *et al.*, 2019). The P4 code is compiled in a Tofino hardware connected to three different hosts acting as eNodeB, Application Server, and

Upstream Router facing the Internet.

We use Stratum (STRATUM, 2022) to configure the port shaping rate from 10G to 1G interface to create a congestion environment. The required entries, such as match-action parameters (i.e., IPG_f^w and τ_f) discussed in Section 4.3.2 and other forwarding entries, are pushed using P4 Runtime(P4RNTIME, 2016). 20 different TCP flows sharing the available link bandwidth, considered as delay critical, are taken to download the 15 Mbytes data from the Internet. We evaluate FCT for each delay critical TCP flow.

We consider three different test-case scenarios. First, as a baseline, we consider the FCT of delay-critical flows without background traffic. Then, in a second scenario representing a traditional setup, we observe the FCT of delay-critical flows with concurrent HH flows (occupy 40% of link bandwidth with around 80 Mbps flow rate). In the third case, we demonstrate the FCT gains when HH are detected and sent to lower priority queues. Since the higher priority queues are served as a strict priority over lower priority queues in Tofino, we can see the difference in FCT for the different scenarios, as shown in Figure 35(b).

4.7 Concluding Remarks

This chapter presented a different approach using IPG analytics to detect HH entirely in the dataplane. Our proposed method supports a push-based approach, where the data-plane reports the controller simultaneously when the flows turn into heavy hitters. Our design has been implemented within the constraints of a programmable switch ASIC. Also, it is easy to adapt to most of the existing counter-based algorithms with high accuracy. We evaluated the Tofino switch ASIC using real traces and achieved high accuracy and low control channel overhead compared to the state-of-the-art algorithms. We showcase the QoS benefits of blending dataplane IPG-based HH detection with dynamic queue allocation.

Some challenges are reserved for future work and are not addressed in this chapter. A few of the difficulties are given below:

- This chapter describes how to identify the flows and store the necessary data for HH detection using a hash-based method. Rather than using straightforward hash tables on bit space, it could be worthwhile to investigate probabilistic data structures like sketches. Observing if the probabilistic data structure could offer greater accuracy while freeing up additional resources for the other routing functions would be interesting.

- In this chapter, IPG is mostly utilized for HH detection. As a next step, we can use In-band Network Telemetry (INT) for IPG metric observability to explore how IPG-based techniques could be extended beyond HH detection applications for network-wide anomaly detection. Another piece of research for innovative applications, such as root cause analysis, is the use of appropriate queries of IPG metrics (per device and network-wide after INT collections).

5 Ml-based Cloud Gaming Traffic Detection

5.1 Introduction

Cloud gaming (CG) is server-based gaming solution that runs games on remote servers in data centers. CG tries to bring high Quality of Experience (QoE) to users by incorporating streaming technologies and network infrastructure optimization. With CG's current growing popularity, it is anticipated that demand for it will continue to grow in the years to come (STATISTA, 2024). Nvidia GeForce Now, Microsoft xCloud, PlayStation Now, and Amazon Luna are a few well-known CG platforms. However, prior research (MARCHAL *et al.*, 2023) also indicates that QoE may be adversely affected by network conditions including bandwidth, latency, jitter and packet loss. Even with high bandwidth network conditions, CG can suffer QoE degradation primarily due to latency and jitter. To overcome this issue, CG traffic must be continuously monitored and given priority over non-latency-sensitive applications. This shows the significance of accurately and quickly classifying CG traffic.

To prioritize latency-sensitive traffic across queues in network devices, we typically employ packet header information (BOMMISETTI *et al.*, 2014), such as the ECN bit in the packet's IP header. Nevertheless, these classification methods might be less precise because of errors in configuration or malevolent actions by the end-host. To circumvent these issues, we respond by employing an intelligent classifier that analyzes traffic and discerns latency-sensitive traffic from millions of flows. According to current initiatives aimed at using ML for traffic monitoring and categorization, particular application flows can be identified by their features and actions (GRAFF *et al.*, 2023). There are procedures that must be followed in order to accomplish this, including data collection, feature extraction, labeling, model selection, and evaluations. These information can be gathered from various cloud gaming platforms and utilized to extract pertinent properties for the purpose of training the model. The trained model can then be used to identify CG traffic

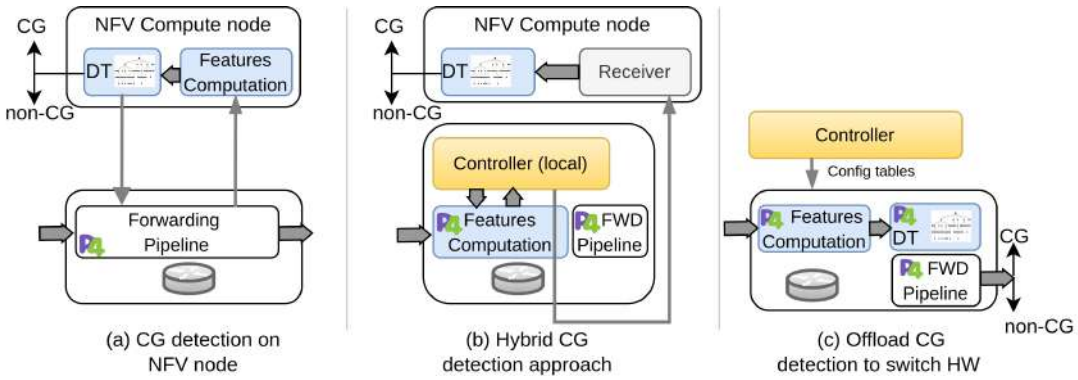


Figure 36 – Cloud Gaming detection approaches on different targets.

after that. The aforementioned procedures are typically carried out on NFV nodes running on commodity servers because they involve complicated functions and calculations that are necessary for the majority of ML models (GRAFF *et al.*, 2023). These software-based approaches take longer to identify lower latency traffic because they need more CPU cycles to implement rules, apply ML models, and gather dataplane information(KUCERA *et al.*, 2020b). Whereas, in-networking computing solutions can significantly enhance the CG experience by shortening the detection time.

Recent developments in P4 (BOSSHART *et al.*, 2014) and programmable dataplanes have made it possible for network devices to do basic operations and computations, enabling high-speed line-rate processing. These benefits, combined with the match-action pipeline, allow supervised and unsupervised ML algorithms like Decision Tree (DT), K-means, Support Vector Machine (SVM), and Naïve Bayes to be mapped with switch pipelines for classification (XIONG; ZILBERMAN, 2019). Even though P4 for TNA is capable of executing these algorithms, fitting ML models with features that need to be calculated and not just taken from packet headers calculation and forwarding action within the restricted switch resources remains a challenging task. Additionally, choosing features is a crucial step in avoiding complicated P4 calculations like mean value and standard deviation.

As seen in Figure 36(a) and 36(b), the majority of state-of-the-art proposals (GRAFF *et al.*, 2023) either use a hybrid approach or execute the entire process on an NFV node (KY *et al.*, 2023). In this chapter, we propose MATADOR, in which the DT method

and feature calculation are performed on a programmable Tofino HW switch for CG classification, as shown in Figure 36(c). To make processing compatible with P4, we take into consideration EWMA for feature computation instead of relying on mean or standard deviation. Furthermore, we optimize the DT method to fit into fewer Tofino stages by reducing dependencies between match-action tables. MATADOR performs the quick and accurate classification of CG traffic at line rate, with an accuracy of around 97%.

In summary, the main contributions of this chapter are:

- We present MATADOR as a CG classification method entirely in Tofino switch hardware and demonstrate its feasibility using two real datasets. Our proposal includes features necessary for CG detection that are compatible with P4-TNA and deliver high detection accuracy.
- We discuss P4 language's constraints and implementation difficulties as well as target-specific subtleties.
- We optimize the ML model to fit all processing, including feature computation and packet processing, within the limited number of stages in the Tofino.
- For reproducibility, we share all pertinent code and documentation available in an open-source repository[1].

5.2 Background

Programmable data planes. Programmable dataplanes permits customized packet processing logic, which aids in achieving low-latency data processing. Furthermore, data plane programming languages, such as P4 and NPL, offer high abstraction levels and are simple to develop code in even for non-programmers with no prior programming knowledge. In P4 (BOSSHART *et al.*, 2014), there are three primary steps involved in processing network packets: parser, match-action, and deparser. Packets are broken down during the parser step to extract pertinent data, like fields and protocol headers. In a match-action

[1] <https://github.com/intrig-unicamp/cg-classifier>

table, packets are compared to predetermined patterns and, depending on the match outcome, the appropriate action is applied to the matched packets. Next, we use the deparser stage to recreate the packets that were sent out. Using these feature, numerous applications, including 5G UPF (SINGH *et al.*, 2019) and HH detection (SINGH *et al.*, 2023), are specified in P4.

In Networking ML. ML has been progressively included in networking to address intricate issues, improve network efficiency, fortify security, and automate related management tasks (RIDWAN *et al.*, 2021). ML is applied in a number of crucial areas to improve network efficiency. One example is the monitoring of network traffic. In order to detect anomalies in network traffic or classify vehicles in intelligent transportation systems (KIM *et al.*, 2022), ML algorithms can examine trends in these types of data. Also, SDN routing applications need to be improved with ML in order to optimize routing in a network efficiently. Additionally, ML is utilized in security to detect insider risks, authenticate users, and identify security threats in real-time.

Recent attempts to successfully offload ML algorithms (AKEM *et al.*, 2023) to programmable dataplanes allow to satisfy the requirements for 5G and 6G use cases. Nevertheless, there are a few challenges. Deploying ML models directly onto dataplane devices can be difficult because to their limited computing and memory resources. This is because ML models can be resource-intensive. Additionally, ML models that are employed in dataplane applications need to balance complexity and accuracy. Resources-constrained contexts may pose challenges for the deployment and management of complex models.

5.3 Related Work

ML is widely utilized in traffic classification, namely in CG, AR, heavy hitting flows, and Denial-of-Service (DoS) attacks. For CG categorization, software and hybrid-based methods are employed. Using flow-level characteristics (e.g., mean packet size, average and standard deviation of inter-arrival times), DT and RF classifiers are used for CG classification in software solutions like that found in (GRAFF *et al.*, 2023). The suggested

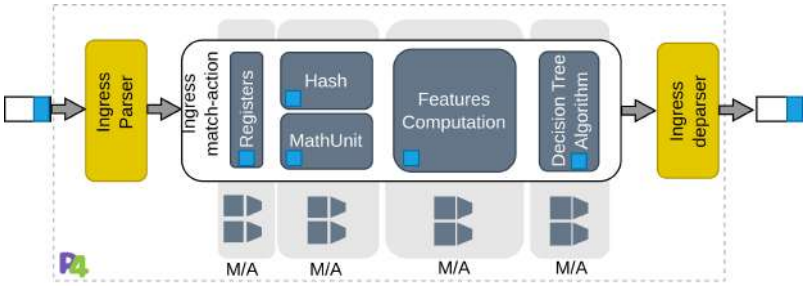


Figure 37 – Proposed P4-TNA pipeline.

classifiers are run at a line rate of roughly 10 Gbps as a VNF. However, an unsupervised machine learning model is employed to categorize CG traffic using a hybrid P4/NFV technique in hybrid solutions like in (KY *et al.*, 2023). The dataplane does the feature calculation (e.g., packet size and inter-arrival times), and it reports the results to the local P4 controller on a regular basis. The NFV node does further calculations to determine CG flows. Sending a feature report for each packet is not feasible in this scenario due to the long processing times. As a result, a report is sent via a digest message every 33 milliseconds. However, due to the limited size digest message, only a restricted amount of features can be reported. Currently, no fully programmable hardware classification for CG is available due to the difficulties in executing ML models and simultaneously carrying out complicated feature calculations.

In addition to CG classification, a hybrid approach to attack detection is suggested in (NASCIMENTO *et al.*, 2023). On the P4 switch, pre-trained lightweight ML models can be used to detect threats. An ML classifier operating at the network edge is utilized for complex operations. However, a simulated environment is used to assess the solution. The similar, albeit non-hybrid, approach to long-lived flow classification used in (KAMATH; SIVALINGAM, 2021). P4 (v1model) switches (Bmv2) are used in the Mininet environment to assess the detection. In (AKEM *et al.*, 2023) and (XIE *et al.*, 2024), an Intel Tofino switch ASIC is used to implement the RF model and binary decision tree (DT), respectively, for traffic classification, including elephant flows and IoT device identification. However, compared to performing complex calculations for feature computations, packet header information are used as an input for the ML model, and thus fits easily within the constraints of programmable hardware.

Our work differs from previous implementation attempts in the following ways. Compared to earlier work, Figure 37 illustrates that MATADOR focuses on CG classification and feature selections, and allows both feature computation (rather than just extracting features from packet headers) and ML model execution within the limited resources on Tofino ASICs. For feature computations, MathUnit, hash functions, and registers are utilized.

5.4 MATADOR Design

The suggested architecture of MATADOR is shown in Figure 38. It comprises of several stages, which will be covered in more detail below, and is coordinated by means of a programmable switch and an SDN controller that can identify CG flows.

5.4.1 Line Rate Feature Extraction

Feature selection is a crucial step in ML-based classification. An ML model's performance can be directly impacted by the features' relevancy and quality. The selection of features is primarily based on CG traffic. However, the same features may also be considered for similar delay critical application traffic (e.g., virtual and augmented reality). As was covered in (GRAFF *et al.*, 2023), feature choice ought to be determined by the CG traffic's uplink and downlink directions. Large packet sizes and frequent packet arrivals in the downlink direction are expected for CG traffic. Multimedia flows are regarded as downlink, and player commands are regarded as uplink. Six features are taken into consideration: the number of packets per TW of a flow for both directions, as well as the moving average of the Packet Size (PS) and the IPG for both directions.

To compute these features in the dataplane, we consider a set of network flows $f \in \mathcal{F} = \{f_1, f_2, ..., f_N\}$, where N indicates the total number of flows. The IPG and PS metrics are updated for each incoming packet entering the switch of $f \in \mathcal{F}$. The last ingress timestamp ($TS_f^l \in \mathbb{N}^+$) is deducted from the current timestamp ($TS_c \in \mathbb{N}^+$) to compute the current IPG_f^c of network flow f. Then, an EWMA metric IPG_f^w is calculated:

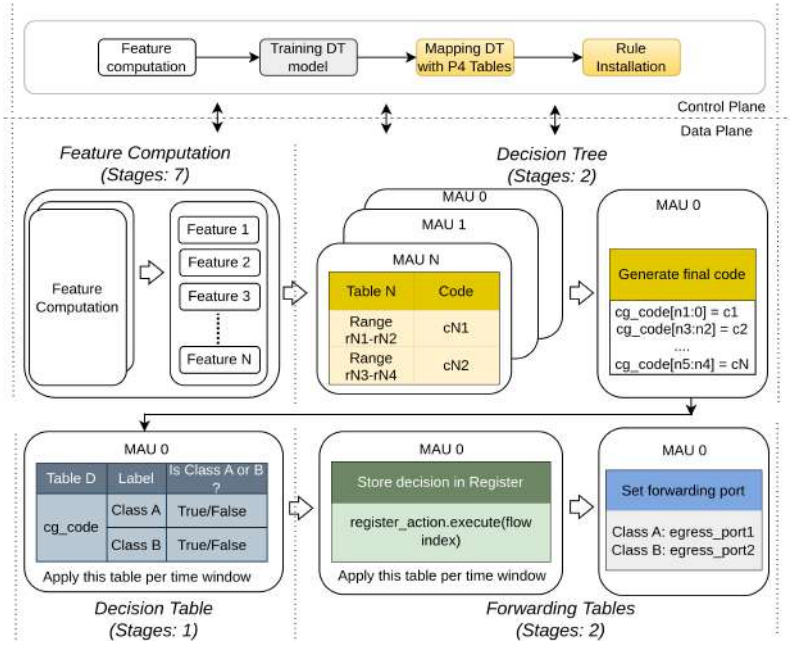


Figure 38 – MATADOR overview.

$$IPG_f^w = \alpha \cdot IPG_f^{w-1} + (1-\alpha) \cdot IPG_f^c, \quad f \in \mathcal{F} \tag{5.1}$$

The same approach is considered for PS EWMA:

$$PS_f^w = \alpha \cdot PS_f^{w-1} + (1-\alpha) \cdot PS_f^c, \quad f \in \mathcal{F} \tag{5.2}$$

where $\alpha \in [0, 1]$ is the smoothing factor and IPG_f^{w-1} and PS_f^{w-1} are the last noted weighted IPG and PS, respectively. The packet header information is used to compute PS_f^c. We compute the EWMA of IPG and PS for both directions using Equations 1 and 2. The Tofino switch ASIC has seven stages dedicated to feature processing, as depicted in Figure 38.

5.4.2 P4-TNA Switch Decision Tree Realization

The proposed design of the DT algorithm for P4-TNA is inspired by the idea presented in (XIONG; ZILBERMAN, 2019). Due to the limited amount of stages in Tofino switch hardware, it is not feasible to execute if-else conditions on each DT node in P4. We match each feature with all of its possible values in a single Tofino stage as a solution. As an

example, Figure 38 shows how, once all feature computations are finished, matches are made for each feature and its possible ranges, resulting in a unique code. The produced codes are kept in metadata, which is then utilized to categorize the traffic into distinct groups. The final two blocks, as shown in Figure 38, can be used to map the traffic class to distinct priority queues or to configure the egress port. Furthermore, no modifications to the DT are necessary for the P4 pipeline. All that is needed is to map the model in the dataplane at runtime and configure the CP.

5.4.3 Control Plane

As seen in Figure 38, training the DT model and setting up the P4 table for mapping with the match-action tables are done in the CP. There are two sets in the dataset: test and training. The decision tree with all the nodes and feature values on each node with criteria is generated by CP using the training data to train the DT model. The DT algorithm is applied using Python scripts, and the Barefoot runtime (BFRT) APIs are called in order to map match-action tables in P4 switch.

5.4.4 Deployment Limitations with P4-TNA

EWMA Computation. According to Equation 1 and 2, the switch HW presents a challenge for EWMA computation. P4 switch hardware, like P4 Tofino, only supports basic division and multiplication operations. For these kinds of computations, there is a feature called Low Pass Filter (LPF) available. LPF comes in two flavors: Sample and Rate. Rate mode offers the traffic rate within a given time interval, whereas sample mode provides an approximation EWMA calculation. Fixing the smoothing factor in sample mode is difficult, but it is necessary for improving accuracy. Without utilizing LPF, our technique is predicated on an approximative EWMA computation. As a next step, we examine the application of LPF with a variable smoothing factor.

Number of Stages. To ensure reduced latency, P4 switch HW (Tofino) has a packet forwarding pipeline with a restricted number of stages. The number of stages is contingent

upon the feature computation, DT algorithm, and forwarding operations, as expounded upon in section 5.4.2. We are unable to parallelize both processes because the DT algorithm applies only after feature computations. Although Tofino 2 has more flexibility than Tofino 1 in terms of the number of stages and resources, we still need to carefully choose the features and attempt to limit them without sacrificing overall accuracy.

5.5 Implementation Details

5.5.1 Datasets

CG training dataset. The ML model is trained using the datasets[2] (in PCAP format, 20% of the dataset) in the same manner as described in (GRAFF *et al.*, 2023)(KY *et al.*, 2023). The four main CG platforms—xCloud (cg_xc), PlayStation (cg_psn), GeForce (cg_gfn), and Stadia (cg_std)—are used to train the ML model. The CG dataset contains a wide range of frame rates and resolutions. While user input has led to moderate packet sizes for the uplink route, multimedia streams have resulted in enormous packet sizes for the downstream direction. For the non-CG traffic class, the following applications are considered: video streaming (live streaming: non_cg_ls, game: non_cg_g, remote desktop protocol: non_cg_rdp, youtube: non_cg_vod), video conferencing (non_cg_vc), and Facebook navigation (browsing: non_cg_b). These applications usually have higher bitrates than CG flows, but their latency needs are less stringent.

CG test dataset. The evaluation process makes use of two distinct datasets. One is same as previously mentioned (i.e., 80% of the dataset). Another test dataset, collected using PCAP format and entirely taken in a different lab setup, displays CG activity from the Xbox CG platform. This dataset contains two scenarios: one using 5G wireless technology and the other using cable internet technology. Over the course of two days, Leris Lab gathered the data, which includes single-player and multiplayer games like Fortnite, Mortal Combat, and Forza. To see the data files and for more details, go to the

[2] <https://cloud-gaming-traces.lhs.loria.fr>

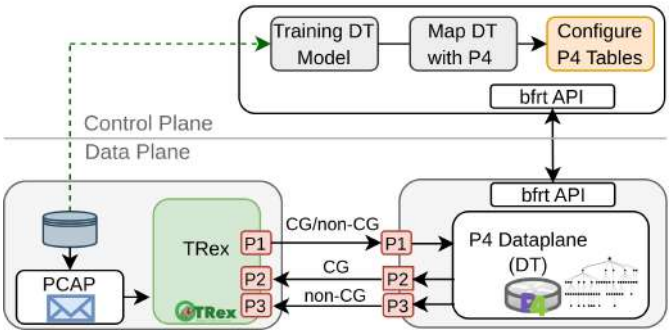


Figure 39 – Tesbed for evaluating CG classification.

GitHub site[3].

5.5.2 Experiment Setup

Testbed. An x86 server equipped with an Intel Xeon D-1518 processor (4 CPU cores, 2.20 GHz) and a *Barefoot Tofino switch ASIC* (Edgecore Wedge 100BF-32X) for CG classification are used to analyze the performance. A testbed with an x86 server connected to a P4 Dataplane (Tofino) via 10G SFP+ ports is depicted in Figure 39. The traffic is sent via TRex, which is operating on an x86 server. A local Python-based controller maps the learned DT model with match-action tables in Tofino in order to configure the necessary table entries.

Evaluation Metrics. Accuracy, F1 Score and detection time. TP are CG flows that are correctly detected, while False Positive (FP) are those flows that are non-CG but detected as CG, and False Negative (FN) are the flows that are CG but detected as non-CG. True Negative (TN) are non-CG flows that are correctly classified. We evaluate the Precision ($Pr = \frac{TP}{TP+FP}$) and Recall ($R = \frac{TP}{TP+FN}$) metrics. Accuracy is evaluated as ($\frac{TP+TN}{TP+FP+TN+FN}$) and the F1 score as $\frac{2\cdot(R\cdot Pr)}{(R+Pr)}$. Also, we use a microsecond scale to measure CG detection time.

5.5.3 Test Execution

For evaluation, we employ the CG test dataset given in section 5.5.1. To evaluate MATADOR, we replay these traces at the same rate that pcap recorded them. TRex sends and

[3] <https://github.com/dcomp-leris/VR-AR-CG-network-telemetry>

receives CG and non-CG packets via switch hardware ports in order to measure performance. Upon receiving traffic (CG or non-CG), the switch uses the source and destination IP addresses to identify the flow. We then compute the features and store them in the registers. Switches identify flows using the decision tree method at the end of each time interval and send packets to the designated hardware port. Receiving both CG and non-CG packets on distinct ports enables the computation of the MATADOR's performance.

5.6 Evaluation

5.6.1 Feasibility and Performance Test

F1-Score and Accuracy. We assess the accuracy of the suggested method using a simulator built on Python in order to compare MATADOR's accuracy. Using a simulator, we map the trained DT model and identify CG flows for each time window. The assessment is carried out for both CG and non-CG streams. Because of more precise computations on an x86 server to support floating-point numbers, the simulator performs slightly better in terms of accuracy and F1-score for both CG and non-CG scenarios, as expected and illustrated in Figure 40. MATADOR does better sometimes. This may be because, there aren't many differences in packet sizes or IPG, which lessens the influence of approximate calculations. As MATADOR represents the first attempt to be fully implemented in switch hardware, we compare it to the state-of-the-art, which is CG hybrid. Results are comparatively close (Figure 41(a)). MATADOR's overall performance on Tofino HW is encouraging, ranging from 92% to 99.99%.

Additionally, we evaluate our trained model using a new dataset (as mentioned in section 5.5.1) that was obtained in a completely different lab configuration. Findings (Figure 42) verify that the suggested model can accurately (80-97%) categorize CG flows—a completely separate dataset from the one used to train it. Further work is being done to optimize the depth of the ML model in order to further increase overall accuracy.

Detection Time. When CG classification is offloaded to the Tofino HW instead of

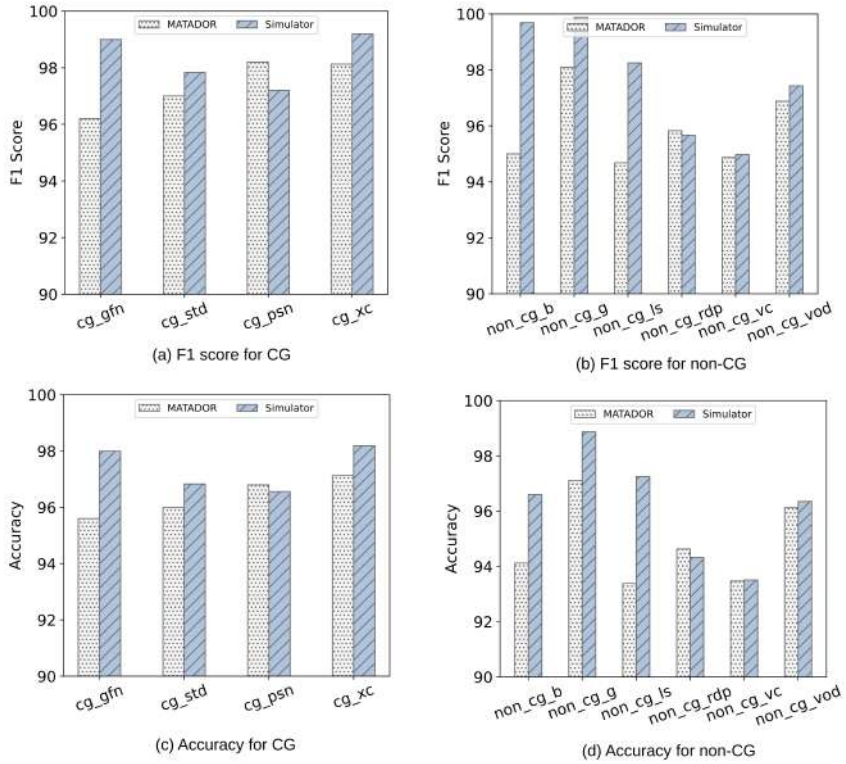


Figure 40 – Performance comparison of MATADOR with simulator outcomes.

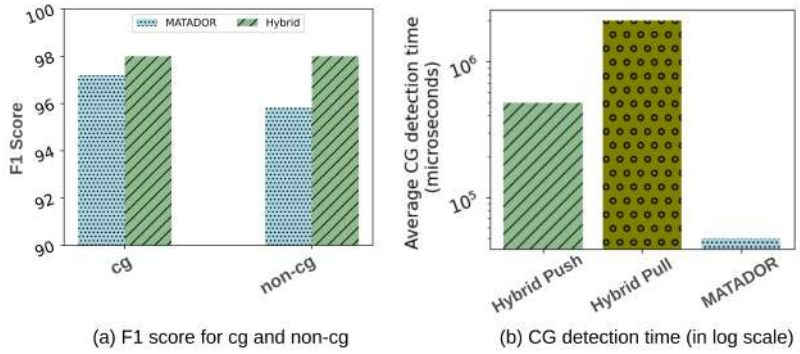


Figure 41 – Performance comparison with the state-of-the-art.

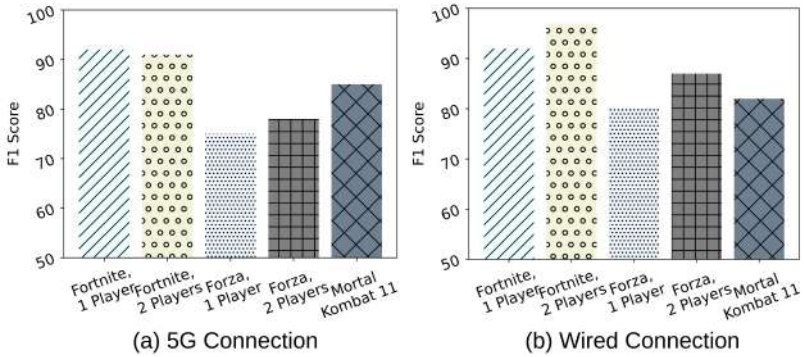


Figure 42 – MATADOR performance using a new cloud gaming dataset (Games: Fortnite, Forza Horizon 5, and Mortal Kombat 11).

the x86 servers, detection times are lowered. Three distinct time frame sizes—0.3, 0.5, and 1 seconds—are used to assess MATADOR. MATADOR computes features for every incoming packet and applies DT at the end of each time interval. According to the current implementation, as indicated in Figure 41(b), MATADOR can classify each flow in 0.03 second into CG or non-CG. When using a hybrid push or pull strategy, the controller reads all of the register values at regular intervals or digest messages (maximum 48 bytes are permitted) are used to send feature calculations. Subsequently, the trained machine learning model is applied by the NFV node to classify flows and send the necessary entries to the switch. A few seconds are needed for the entire processing, as Figure 41(b) illustrates (ZHANG *et al.*, 2021)(KUCERA *et al.*, 2020b).

5.6.2 Resource Utilization

The P4 pipeline of MATADOR is approximately one thousand lines of code for the Tofino switch ASIC. Code blocks for parsers, decision trees, feature computation, header definitions, and constants are among them. We examine the resources for the DT and feature computations independently, as indicated in Table 8, in order to assess the resource usage on the Tofino switch. Because DT uses five feature tables and one decision table with range match, TCAM is filled by about 12%. Additionally, as mentioned in section 5.4.2, DT occupies the three number of stages—two for feature tables and one for decision tables. Seven stages and 4.7% of hash bits are needed for feature computation in order to identify the flows. MATADOR fits into ten stages total, with two more stages for action. Additionally, a maximum of 16 logical tables for both exact and ternary matches are supported by each match-action unit. 19% of the logical table id is consumed because several tables with actions are run to compute features and determine the uplink or downlink direction.

Table 8 – Resource usage of MATADOR.

Resource	DT	Features	CG (DT+Features)
Exact Match Result Bus	0.0	14.1	14.1
Exact Match Search Bus	0.0	8.9	9.4
Hash Bit	0.0	4.7	4.7
Logical Table Id	3.1	19.3	22.4
SRAM	0.5	3.4	4.1
TCAM	12.2	0.0	12.2
Stages	3	7	10

5.7 Concluding Remarks

This chapter investigates machine learning in a programmable dataplane to improve detection times and line-rate classification of cloud gaming traffic. It is essential to choose features that fit the overall design model, given the restricted CPU capacity and resource limits of the switch hardware.

While programmable dataplanes have limited computational and memory resources, there are still many areas to explore that are not discussed in this chapter. Here are a few of them:

- Future activities will consider creating a multi-class classification that takes into account not only cloud gaming but also other application traffic types, like DoS traffic and virtual and augmented reality.

- According to the current implementation, we maintain the flow's necessary information for classification in a hash table. We store both CG and non-CG flows in the hash table. As of right now, our proposed design can manage about a million flows. In order to free up more switch resources for other tasks, it will be necessary to try to keep only specific application flows in the hash table in future development.

- This chapter assesses the suggested approach's performance; however, it does not cover the use cases where the suggested strategy can be applied to observe a real-time improvement in QoE. To further improve the user experience, it would be interesting to investigate how the suggested model might be applied in various scenarios, such as

traffic prioritization (e.g., video streaming) to higher priority queues in the network switch.

- While we primarily concentrate on the DT algorithm in this chapter, other machine learning techniques, such as RF and unsupervised learning, which need more computations but may yield higher accuracy, may also be used on switch hardware in the future.

6 Conclusion & Future Work

In conclusion, we provide a concise summary of each aspect of enhancing mobile user plane functions with programmable dataplanes, highlighting their significant impact on network performance, flexibility, and scalability. Additionally, we address the shortcomings and corresponding solutions for the various sections of this thesis, as well as explore further potential related opportunities in the future work section.

6.1 Conclusions

We typically select NFV nodes for complex network functions because general-purpose processors provide more flexibility and programmability. However, we are unable to guarantee the performance. In this thesis, we optimize the efficiency of mobile networks by using a programmable dataplane to offload mobile core user plane functions to programmable switch hardware. We focus on two main aspects of the mobile core network: 1. performance of the user plane function regarding latency, throughput, and scalability, and 2. traffic classification in the user plane function for priority. Our Hybrid-UPG shows how we can use numerous programmable targets to make mobile networks more adaptable and scalable without compromising performance.

Furthermore, we show that inter-packet gap can be used as a metric for traffic classification, which has benefits over traditional packet counters. Additionally, we show how programmable dataplanes can benefit from machine learning by monitoring specific application flows to improve the quality of service in mobile networks. We validate each proposal with real-time programmable hardware and compare it with the state-of-the-art. Moreover, we analyze the constraints and limitations of P4 language and programmable targets for each design and approach we have showcased. We have also added functionality, developed an open-source tool, and assisted with open-source projects, among other contributions. By better understanding how to use a programmable dataplane with re-

strictions and limitations to boost the power and flexibility of SDN networking and satisfy the expectations of 5G vital applications, the research community will gain from the work and conclusions in this thesis.

6.2 Future Work

Several topics have been covered in the development of mobile core user plane functions, including design for increased efficiency and traffic handling to improve the quality of service. We designed each of our suggested solutions to answer particular research questions. During the implementation process, we encountered new ideas and challenges that we will discuss here. These contributions are organized into four main categories.

Adding Missing UPF Features. The implementation of UPF, as covered in Chapter 2, indicates that a few key features and activities remain to be completed, such as traffic management (TM), set rate limiting for billing purposes, optimization code to scale the number of users, and flow prioritization based on traffic behavior. TM necessitates mapping the QoS Class Identifier (QCI) values, as per the 3GPP standard, to priority queues in programmable switches and configuring traffic classes optimized by the SDN controller. In this manner, we can enhance QoS by prioritizing traffic according to its classes. In addition, metering in P4 must be implemented to restrict the traffic rate, aiding operator billing. Additionally, there is still an opportunity to optimize the UPF P4 code to free up more memory for other purposes or to allow for greater handling for users as they scale up.

Optimizing Traffic handling in Hybrid-UPF. We covered the use of the HH detection approach in Chapter 3 to determine which flows to keep in switch HW or offload to SW for processing. However, relying solely on these methods may have a detrimental effect on the quality of service. We also need to keep non-HH flows in switch HW for quick processing since they can also need reduced latency. We need more optimization to dynamically manage traffic among several programmable targets according to their characteristics, behavior, and QoS needs per flow or traffic class. Additionally, for hybrid design, we

currently concentrate on x86-based servers and Tofino switches; however, we also need to consider other hardware, like FPGA and smartNICs, which can offer greater flexibility to create more dynamic solutions.

IPG for Capturing more Dataplane Events. Chapter 4 discusses how we utilize the IPG metric to detect HH flows in programmable switch HW. Not only do HH flows need to be monitored, but several other applications, including global iceberg identification, change detection, entropy estimates, and DDoS detection, must also be monitored. A denial-of-service attack occurs when an attacker overloads a server with traffic to stop users from accessing online services. A global iceberg is a distributed denial of service attack in which specific components repeatedly occur globally throughout the distributed streams but do not manifest as a denial of service locally. Moreover, finding the flows that most influence traffic change over two successive time intervals is the change detection method. Capturing these events is critical to prevent lower network performance. Furthermore, it is challenging to integrate the numerous applications needed for capturing these events into the hardware switches due to their constrained memory and resource limits. We require a solution that allows us to capture the maximum number of events in a programmable data-plane with a single application that uses IPG.

Expanding ML-Based Methods for Traffic Analysis. We employ a machine learning strategy in Chapter 5 to categorize cloud gaming flows. The process of CG detection makes use of the decision tree. Other traffic, such as AR and VR, needs to be monitored. As was previously said, due to the memory constraints of switch hardware, we want a solution that can be applied to classify traffic into many classes accurately. Given programmable hardware, this may need more calculation, making it more difficult. Furthermore, unsupervised techniques like clustering and further machine learning algorithms like random forest must also be investigated. Although these methods might use more resources and computations, their classification accuracy could be higher.

Bibliography

ABHIK, B.; DIPTYAROOP, M.; PRATEEK, A.; NILESH, U.; RINKU, S.; MYTHILI, V. Leveraging programmable dataplanes for a high performance 5g user plane function. In: *5th Asia-Pacific Workshop on Networking*. New York, NY, USA: Association for Computing Machinery, 2021. (APNet '21), p. 1–8. ISBN 9781450385879. Available from Internet: <https://doi.org/10.1145/3469393.3469400>. Cited 2 times on pages 54 and 57.

AKEM, A. T.-J.; GUCCIARDO, M.; FIORE, M. Flowrest: Practical flow-level inference in programmable switches with random forests. In: *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2023. p. 1–10. Cited 2 times on pages 105 and 106.

AL-FARES, M.; RADHAKRISHNAN, S.; RAGHAVAN, B.; HUANG, N.; VAHDAT, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In: *Networked Systems Design and Implementation (NSDI)*. [S.l.]: USENIX, 2010. Cited 3 times on pages 10, 71, and 73.

ALIZADEH, M.; GREENBERG, A.; MALTZ, D. A.; PADHYE, J.; PATEL, P.; PRABHAKAR, B.; SENGUPTA, S.; SRIDHARAN, M. Data center tcp (dctcp). In: *Proceedings of the ACM SIGCOMM 2010 Conference*. New York, NY, USA: Association for Computing Machinery, 2010. (SIGCOMM '10), p. 63–74. ISBN 9781450302012. Cited on page 99.

ANTICHI, G.; SHAHBAZ, M.; GENG, Y.; ZILBERMAN, N.; COVINGTON, A.; BRUYERE, M.; MCKEOWN, N.; FEAMSTER, N.; FELDERMAN, B.; BLOTT, M.; MOORE, A. W.; OWEZARSKI, P. Osnt: open source network tester. *IEEE Network*, v. 28, n. 5, p. 6–12, 2014. Cited 2 times on pages 46 and 93.

ASIATICI, M.; IENNE, P. Stop crying over your cache miss rate: Handling efficiently thousands of outstanding misses in fpgas. In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: Association for Computing Machinery, 2019. (FPGA '19), p. 310–319. ISBN 9781450361378. Available from Internet: <https://doi.org/10.1145/3289602.3293901>. Cited on page 25.

BASAT, R. B.; CHEN, X.; EINZIGER, G.; ROTTENSTREICH, O. Designing Heavy-Hitter Detection Algorithms for Programmable Switches. In: *Transactions on Networking, Volume:28, Issue:3*. [S.l.]: IEEE/ACM, 2020. Cited 4 times on pages 58, 59, 71, and 87.

BASAT, R. B.; EINZIGER, G.; FRIEDMAN, R.; KESLASSY, I. Heavy Hitters in Streams and Sliding Windows. In: *International Conference on Computer Communications (INFOCOM)*. [S.l.]: IEEE, 2016. Cited 2 times on pages 74 and 76.

BASAT, R. B.; Einziger, G.; Friedman, R.; Kassner, Y. Poster abstract: A sliding counting bloom filter. In: *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.: s.n.], 2017. p. 1012–1013. Cited on page 76.

BASAT, R. B.; EINZIGER, G.; KESLASSY, I.; ORDA, A.; VARGAFTIK, S.; WAISBARD, E. Memento: Making Sliding Windows Efficient for Heavy Hitters. In: *Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. [S.l.]: ACM, 2018. Cited 2 times on pages 74 and 76.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Network traffic characteristics of data centers in the wild. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA: Association for Computing Machinery, 2010. (IMC '10), p. 267–280. ISBN 9781450304832. Cited 4 times on pages 79, 80, 93, and 96.

BENSON, T.; ANAND, A.; AKELLA, A.; ZHANG, M. MicroTE: Fine Grained Traffic Engineering for Data Centers. In: *COnference on Emerging Networking EXperiments and Technologies (CoNEXT)*. [S.l.]: ACM, 2011. Cited on page 71.

BOMMISETTI, S.; ANNAPPA, B.; TAHILIANI, M. P. Extended ecn mechanism to mitigate ecn-based attacks. In: *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. [S.l.: s.n.], 2014. p. 1105–1110. Cited on page 102.

BOSSHART, P.; DALY, D.; GIBB, G.; IZZARD, M.; MCKEOWN, N.; REXFORD, J.; SCHLESINGER, C.; TALAYCO, D.; VAHDAT, A.; VARGHESE, G.; WALKER, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, Association for Computing Machinery, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2656877.2656890>. Cited 7 times on pages 25, 27, 40, 42, 53, 103, and 104.

CAIDA. *Anonymized Internet Traces*. 2016. <http://www.caida.org/data/passive/passive_dataset.xml>. Cited 6 times on pages 64, 69, 72, 80, 93, and 95.

CASCONE, C.; CHAU, U. *Offloading VNFs to programmable switches using P4*. March 2018. Cited 2 times on pages 42 and 43.

CAVIUM. *Cavium-XPliant family of programmable Ethernet switches*. 2014. Available from Internet: <https://www.openswitch.net/cavium/>. Cited on page 71.

CHEN, X.; FEIBISH, S. L.; KORAL, Y.; REXFORD, J.; ROTTENSTREICH, O.; MONETTI, S. A.; WANG, T.-Y. Fine-Grained Queue Measurement in the Data Plane. In: *Conference on Emerging Networking Experiments And Technologies (CoNEXT)*. [S.l.]: ACM, 2019. Cited 2 times on pages 72 and 74.

CISCO. *Cisco. Internet of Things*. 2016. Available from Internet: <https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf>. Cited on page 61.

Cisco NetFlow. *Cisco IOS Netflow*. 1996. <http://cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html>. Cited on page 75.

Cisco SFlow. *SFlow*. 1996. <https://sflow.org/>. Cited on page 75.

COSTA-REQUENA, J.; LIYANAGE, I.; O.L.PEREZ, M. Y.; ITZAZELAIA, M.; OCA, E. de. SDN and NFV Integration in Generalized Mobile Network Architecture. *IEEE European Conference on Networks and Communications (EuCNC)*, June 2015. Cited 2 times on pages 39 and 42.

CURTIS, A. R.; MOGUL, J. C.; TOURRILHES, J.; YALAGANDULA, P.; SHARMA, P.; BANERJEE, S. DevoFlow: Scaling Flow Management for High-performance Networks. In: *Special Interest Group on Data Communication (SIGCOMM)*. [S.l.]: ACM, 2011. Cited on page 71.

DANG, H. T.; WANG, H.; JEPSEN, T.; BREBNER, G.; KIM, C.; REXFORD, J.; SOULé, R.; WEATHERSPOON, H. Whippersnapper: A p4 language benchmark suite. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 2017. (SOSR '17), p. 95–101. ISBN 9781450349475. Available from Internet: <https://doi.org/10.1145/3050220.3050231>. Cited on page 53.

ERUNKULU, O. O.; ZUNGERU, A. M.; LEBEKWE, C. K.; MOSALAOSI, M.; CHUMA, J. M. 5g mobile communication applications: A survey and comparison of use cases. *IEEE Access*, v. 9, p. 97251–97295, 2021. Cited on page 26.

ESTAN, C.; VARGHESE, G. New Directions in Traffic Measurement and Accounting. In: *Special Interest Group on Data Communication (SIGCOMM)*. [S.l.]: ACM, 2002. Cited on page 71.

FELDMANN, A.; GREENBERG, A.; LUND, C.; REINGOLD, N.; REXFORD, J.; TRUE, F. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In: *Special Interest Group on Data Communication (SIGCOMM)*. [S.l.]: ACM, 2000. Cited on page 71.

FU, Y.; Li, D.; Shen, S.; Zhang, Y.; Chen, K. Clustering-preserving network flow sketching. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2020. p. 1309–1318. Cited on page 75.

GIANNI, A.; MUHAMMAD, S.; YILONG, G.; NOA, Z.; ADAM, C.; MARC, B.; NICK, M.; NICK, F.; BOB, F.; MICHAELA, B. *et al.* Osnt: Open source network tester. *IEEE Network*, IEEE, v. 28, n. 5, p. 6–12, 2014. Cited 2 times on pages 45 and 63.

GRAFF, P.; MARCHAL, X.; CHOLEZ, T.; MATHIEU, B.; FESTOR, O. Efficient identification of cloud gaming traffic at the edge. In: *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2023. p. 1–10. Cited 5 times on pages 102, 103, 105, 107, and 110.

GUPTA, A.; JHA, R. K. A survey of 5g network: Architecture and emerging technologies. *IEEE Access*, v. 3, p. 1206–1232, 2015. Cited on page 23.

HAO, F.; KODIALAM, M.; LAKSHMAN, T. V. *Line-rate, real-time-traffic detector*. [S.l.]: Google Patents, 2011. US Patent 8,054,760. Cited on page 75.

HASEGAWA, G.; MURATA, M. Joint bearer aggregation and control- data plane separation in LTE EPC for increasing M2M communica- tion capacity. *IEEE Glob. Commun. Conf. (GLOBECOM)*, p. 1–6, 2015. Cited on page 42.

HATTACHI, E. R. *Next Generation Mobile Networks, NGMN*. 2015. Available from Internet: <https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf>. Cited 2 times on pages 39 and 52.

HE, M.; Alba, A. M.; Basta, A.; Blenk, A.; Kellerer, W. Flexibility in softwarized networks: Classifications and research challenges. *IEEE Communications Surveys Tutorials*, v. 21, n. 3, p. 2600–2636, thirdquarter 2019. Cited 2 times on pages 39 and 42.

HOGAN, M.; LANDAU-FEIBISH, S.; ARASHLOO, M. T.; REXFORD, J.; WALKER, D. Modular switch programming under resource constraints. In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, 2022. p. 193–207. ISBN 978-1-939133-27-4. Available from Internet: <https://www.usenix.org/conference/nsdi22/presentation/hogan>. Cited on page 24.

IBANEZ, S.; BREBNER, G.; MCKEOWN, N.; ZILBERMAN, N. The p4->netfpga workflow for line-rate packet processing. In: *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: Association for Computing Machinery, 2019. (FPGA '19), p. 1–9. ISBN 9781450361378. Available from Internet: <https://doi.org/10.1145/3289602.3293924>. Cited on page 55.

INTEL. *Home - DPDK*. 2010. https://www.dpdk.org [Online; accessed 7-June-2021]. Available from Internet: <https://www.dpdk.org>. Cited 2 times on pages 27 and 52.

Intel. *Tofino chip*. 2017. Available from Internet: <https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html>. Cited on page 71.

JIN, X.; LI, X.; ZHANG, H.; SOULÉ, R.; LEE, J.; FOSTER, N.; KIM, C.; STOICA, I. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In: *Symposium on Operating Systems Principles (SOSP)*. [S.l.]: ACM, 2017. Cited on page 75.

KAIPPALLIMALIL, J.; CHAN, H. A. Network virtualization and direct Ethernet transport for packet data network connections in 5G wireless. *IEEE Glob. Commun. Conf. (GLOBECOM)*, IEEE Press, p. 1836–1841, 2014. Cited on page 42.

KALJIC, E.; MARIC, A.; NJEMCEVIC, P.; HADZIALIC, M. A survey on data plane flexibility and programmability in software-defined networking. *IEEE Access*, v. 7, p. 47804–47840, 2019. Cited on page 27.

KAMATH, R.; SIVALINGAM, K. M. Machine learning based flow classification in dcns using p4 switches. In: *2021 International Conference on Computer Communications and Networks (ICCCN)*. [S.l.: s.n.], 2021. p. 1–10. Cited on page 106.

KAMMENHUBER, N.; KENCL, L. Efficient statistics gathering from tree-search methods in packet processing systems. In: *IEEE International Conference on Communications, 2005. ICC 2005. 2005.* [S.l.: s.n.], 2005. v. 3, p. 1483–1489 Vol. 3. Cited on page 76.

KANNAN, R. J. P. G.; CHAN, M. C. Precise time-synchronization in the data-plane using programmable switching ASICs. *ACM Symp. SDN Res*, IEEE Press, n. 2, p. 8–20, 2019. Cited 2 times on pages 42 and 48.

KATTA, N.; ALIPOURFARD, O.; REXFORD, J.; WALKER, D. Cacheflow: Dependency-aware rule-caching for software-defined networks. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 2016. (SOSR '16). ISBN 9781450342117. Available from Internet: <https://doi.org/10.1145/2890955.2890969>. Cited on page 57.

KEMPF B. JOHANSSON, S. P. H. L. J.; NILSSON, T. Moving the mobile Evolved Packet Core to the cloud. *Wireless and Mobile Computing, Networking and Communications (WiMob)*, v. 103, p. 784–791, Oct, 2012. Cited on page 39.

KFOURY, E. F.; CRICHIGNO, J.; BOU-HARB, E. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, v. 9, p. 87094–87155, 2021. Cited on page 53.

KIM, J.; NAKASHIMA, M.; FAN, W.; WUTHIER, S.; ZHOU, X.; KIM, I.; CHANG, S.-Y. A machine learning approach to anomaly detection based on traffic monitoring for secure blockchain networking. *IEEE Transactions on Network and Service Management*, v. 19, n. 3, p. 3619–3632, 2022. Cited on page 105.

KREUTZ, D.; Ramos, F. M. V.; Veríssimo, P. E.; Rothenberg, C. E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, Jan 2015. Cited on page 39.

KUCERA, J.; POPESCU, D. A.; WANG, H.; MOORE, A.; KOrENEK, J.; ANTICHI, G. Enabling Event-Triggered Data Plane Monitoring. In: *Symposium on SDN Research (SOSR)*. [S.l.]: ACM, 2020. Cited 5 times on pages 71, 72, 74, 75, and 76.

KUCERA, J.; POPESCU, D. A.; WANG, H.; MOORE, A.; KOrENEK, J.; ANTICHI, G. Enabling event-triggered data plane monitoring. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 2020. (SOSR '20), p. 14–26. ISBN 9781450371018. Available from Internet: <https://doi.org/10.1145/3373360.3380830>. Cited 2 times on pages 103 and 114.

KUNDEL, R.; MEUSER, T.; KOPPE, T.; HARK, R.; STEINMETZ, R. User Plane Hardware Acceleration in Access Networks: Experiences in Offloading Network Functions in Real 5G Deployments. In: *55th Hawaii International Conference on System Sciences*. [S.l.: s.n.], 2022. Cited on page 56.

KUNDEL, R.; NOBACH, L.; BLENDIN, J.; MAAS, W.; ZIMBER, A.; KOLBE, H.; SCHYGUDA, G.; GUREVICH, V.; HARK, R.; KOLDEHOFE, B.; STEINMETZ, R. Openbng: Central office network functions on programmable data plane hardware. *Int. J. Netw. Manag.*, John Wiley Sons, Inc., USA, v. 31, n. 1, jan. 2021. ISSN 1099-1190. Available from Internet: <https://doi.org/10.1002/nem.2134>. Cited on page 57.

KUNDEL, R.; NOBACH, L.; BLENDIN, J.; KOLBE, H.; SCHYGUDA, G.; GUREVICH, V.; KOLDEHOFE, B.; STEINMETZ, R. P4-BNG: Central Office Network Functions on Programmable Packet Pipelines. *IEEE International Conference on Network and Server Management (CNSM)*, IEEE Press, p. 21–25, October 2019. Cited on page 42.

KY, J.; GRAFF, P.; MATHIEU, B.; CHOLEZ, T. A hybrid p4/nfv architecture for cloud gaming traffic detection with unsupervised ml. In: *2023 IEEE Symposium on Computers and Communications (ISCC)*. Los Alamitos, CA, USA: IEEE Computer Society, 2023. p. 733–738. Available from Internet: <https://doi.ieeecomputersociety.org/10.1109/ISCC58397.2023.10217863>. Cited 3 times on pages 103, 106, and 110.

LAKE, D.; WANG, N.; TAFAZOLLI, R.; SAMUEL, L. Softwarization of 5g networks – implications to open platforms and standardizations. *IEEE Access*, PP, p. 1–1, 04 2021. Cited 2 times on pages 23 and 52.

LAKHINA, A.; CROVELLA, M.; DIOT, C. Diagnosing Network-wide Traffic Anomalies. In: *Computer Communication Review, Volume: 34, Issue: 4*. [S.l.]: ACM, 2004. Cited on page 71.

LAPOLLI, C.; MARQUES, J. A.; GASPARY, L. P. Offloading real-time ddos attack detection to programmable data planes. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. [S.l.: s.n.], 2019. p. 19–27. ISSN 1573-0077. Cited on page 99.

LI, Y.; CHEN, M. Software-defined network function virtualization: A survey. *IEEE Access*, IEEE, v. 3, p. 2542–2553, 2015. Cited on page 39.

LI, Z. M. L.; REXFORD, J. Toward software-defined cellular networks. *Software Defined Networking (EWSDN)*, p. 07–12, 2012. Cited on page 42.

LIN, Y.-B.; HUANG, T.-J.; TSAI, S.-C. Enhancing 5g/iot transport security through content permutation. *IEEE Access*, v. 7, p. 94293–94299, 2019. Cited on page 56.

LINGUAGLOSSA, L.; Lange, S.; Pontarelli, S.; Rétvári, G.; Rossi, D.; Zinner, T.; Bifulco, R.; Jarschel, M.; Bianchi, G. Survey of performance acceleration techniques for network function virtualization. *Proceedings of the IEEE*, v. 107, n. 4, p. 746–764, April 2019. Cited 2 times on pages 40 and 48.

LIU, Y.; TOWSLEY, D. F.; YE, T.; BOLOT, J.-C. An Information-theoretic Approach to Network Monitoring and Measurement. In: *Internet Measurement Conference (IMC)*. [S.l.]: USENIX, 2005. Cited on page 73.

LIU, Z.; MANOUSIS, A.; VORSANGER, G.; SEKAR, V.; BRAVERMAN, V. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In: *Special Interest Group on Data Communication (SIGCOMM)*. [S.l.]: ACM, 2016. Cited 5 times on pages 71, 72, 74, 75, and 84.

MACDAVID, R.; CASCONE, C.; PINGPING, L.; PADMANABHAN, B.; THAKUR, A.; PETERSON, L.; REXFORD, J.; SUNAY, O. A P4-based 5G User Plane Function. In: *Symposium on SDN Research (SOSR)*. [S.l.]: ACM, 2021. Cited 2 times on pages 54 and 57.

MARCHAL, X.; GRAFF, P.; KY, J.; CHOLEZ, T.; TUFFIN, S.; MATHIEU, B.; FESTOR, O. An analysis of cloud gaming platforms behaviour under synthetic network constraints and real cellular networks conditions. In: *2023 Journal of Network and Systems Management*. [S.l.]: Springer, 2023. p. 1573–7705. Cited on page 102.

MARQUES, J.; LEVCHENKO, K.; GASPARY, L. Intsight: Diagnosing slo violations with in-band network telemetry. In: *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*. New York, NY, USA: Association for Computing Machinery, 2020. (CoNEXT '20), p. 421–434. ISBN 9781450379489. Cited on page 99.

MATIAS, J.; GARAY, J.; TOLEDO, N.; UNZILLA, J.; JACOB, E. Toward an sdn-enabled nfv architecture. *IEEE Communications Magazine*, v. 53, n. 4, p. 187–193, 2015. Cited on page 24.

MAWI. *Working Group Traffic Archive*. 2020. <https://mawi.wide.ad.jp/mawi/ditl/ditl2020-G/>. Cited 4 times on pages 79, 80, 93, and 96.

METWALLY, A.; AGRAWAL, D.; ABBADI, A. E. Efficient Computation of Frequent and Top-k Elements in Data Streams. In: *International Conference on Database Theory (ICDT)*. [S.l.]: Springer-Verlag, 2005. Cited 2 times on pages 75 and 89.

MIAO, R.; ZENG, H.; KIM, C.; LEE, J.; YU, M. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. New York, NY, USA: Association for Computing Machinery, 2017. (SIGCOMM '17), p. 15–28. ISBN 9781450346535. Cited on page 75.

MOHAMMADKHAN, A.; RAMAKRISHNAN, K. K.; RAJAN, A. S.; MACIOCCO, C. CleanG: A clean-slate EPC architecture and control plane protocol for next generation cellular networks. *ACM CoNEXT Workshop Cloud Assist. Netw. (CAN)*, IEEE Press, p. 31–36, 2016. Cited on page 42.

MOHAMMADKHAN, A.; RAMAKRISHNAN, K. K.; RAJAN, A. S.; MACIOCCO, C. Considerations for re-designing the cellular infrastructure exploiting software- based networks. *In ICNP*, p. 397–413, 2016. Cited on page 40.

NASCIMENTO, A.; ABREU, D.; RIKER, A.; ABELéM, A. Aid-sdn: Advanced intelligent defense for sdn using p4 and machine learning. In: *2023 IEEE Latin-American Conference on Communications (LATINCOM)*. [S.l.: s.n.], 2023. p. 1–6. Cited on page 106.

NESHATPOUR, K.; MALIK, M.; GHODRAT, M. A.; SASAN, A.; HOMAYOUN, H. Energy-efficient acceleration of big data analytics applications using fpgas. In: *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*. USA: IEEE Computer Society, 2015. (BIG DATA '15), p. 115–123. ISBN 9781479999262. Available from Internet: <https://doi.org/10.1109/BigData.2015.7363748>. Cited on page 55.

NEUGEBAUER, R.; ANTICHI, G.; ZAZO, J. F.; AUDZEVICH, Y.; LóPEZ-BUEDO, S.; MOORE, A. W. Understanding pcie performance for end host networking. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. New York, NY, USA: Association for Computing Machinery, 2018. (SIGCOMM '18), p. 327–341. ISBN 9781450355674. Available from Internet: <https://doi.org/10.1145/3230543.3230560>. Cited on page 52.

NGUYEN, V.; BRUNSTROM, A.; GRINNEMO, K.; TAHERI, J. SDN/NFV- based Mobile Packet Core Network Architectures: A Survey. *IEEE Communications Surveys & Tutorials*, IEEE Press, v. 19, n. 3, p. 1567–1602, 2017. Cited on page 42.

NPLSPEC. *Specification*. 2021. Available from Internet: <https://github.com/nplang/NPL-Spec>. Cited on page 27.

ONF. Aether enterprise-5g/lte-edge-cloud-as-a-service. In: *Aether White Paper*. [S.l.: s.n.], 2020. Cited on page 57.

Open-Source P4 Switch. 2017. [Available]:<https://github.com/p4lang/switch>. Cited 2 times on pages 88 and 98.

P4-HH. 2022. Available from Internet: <https://github.com/intrig-unicamp/P4-HH>. Cited on page 62.

P4-HH. *HH detection*. 2022. Available from Internet: <https://github.com/intrig-unicamp/P4-HH>. Cited 5 times on pages 82, 87, 90, 93, and 95.

P4RNTIME. *A control plane specification for controlling the data plane elements of a device or program defined by a P4 program*. 2016. Available from Internet: <https://github.com/p4lang/p4runtime/>. Cited on page 100.

PATRA, G.; Rothenberg, C. E.; Pongracz, G. Macsad: High performance dataplane applications on the move. In: *2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*. [S.l.: s.n.], 2017. p. 1–6. Cited 4 times on pages 41, 42, 45, and 63.

PLANCHER, B.; NEUMAN, S. M.; BOURGEAT, T.; KUINDERSMA, S.; DEVADAS, S.; REDDI, V. J. Accelerating robot dynamics gradients on a cpu, gpu, and fpga. *IEEE Robotics and Automation Letters*, v. 6, n. 2, p. 2335–2342, 2021. Cited on page 24.

RAJAN, A. S.; GOBRIEL, S.; MACIOCCO, C.; RAMIA, K. B.; KAPURY, S.; SINGHY, A.; ERMANZ, J.; GOPALAKRISHNANZ, V.; JANAZ, R. Understanding the bottlenecks in virtualizing cellular core network functions. *Proc. 21st IEEE Int. Workshop Local Metropolitan Area Netw. (LANMAN)*, IEEE Press, p. 1–6, 2015. Cited on page 39.

RAUMER, D.; WOHLFART, F.; SCHOLZ, D.; CARLE, G. Performance Exploration of Software-based Packet Processing Systems. In: *Proceedings of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 6. GI/ITG-Workshop MMBnet 2015*. Hamburg, Germany: [s.n.], 2015. Cited 2 times on pages 23 and 25.

RICART-SANCHEZ, R.; MALAGON, P.; ALCARAZ-CALERO, J. M.; WANG, Q. P4-netfpga-based network slicing solution for 5g mec architectures. In: *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. [S.l.: s.n.], 2019. p. 1–2. Cited on page 56.

RICART-SANCHEZ, R.; MALAGON, P.; SALVA-GARCIA, P.; PEREZ, E. C.; WANG, Q.; Alcaraz Calero, J. M. Towards an fpga-accelerated programmable data path for edge-to-core communications in 5g networks. *Journal of Network and Computer Applications*, v. 124, p. 80–93, 2018. ISSN 1084-8045. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1084804518302923>. Cited on page 55.

RIDWAN, M. A.; RADZI, N. A. M.; ABDULLAH, F.; JALIL, Y. E. Applications of machine learning in networking: A survey of current issues and future challenges. *IEEE Access*, v. 9, p. 52523–52556, 2021. Cited on page 105.

RODRIGUEZ, F.; PATRA, G.; CSIKOR, L.; ROTHENBERG, C.; LAKI, P. V. S.; PONGRáCZ, G. Bb-gen: A packet crafter for p4 target evaluation. In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. New York, NY, USA: ACM, 2018. (SIGCOMM '18), p. 111–113. ISBN 978-1-4503-5915-3. Available from Internet: <http://doi.acm.org/10.1145/3234200.3234229>. Cited on page 46.

ROSTAMI, S.; HEISKA, K.; PUCHKO, O.; TALVITIE, J.; LEPPANEN, K.; VALKAMA, M. Novel wake-up signaling for enhanced energy-efficiency of 5g and beyond mobile devices. In: *2018 IEEE Global Communications Conference (GLOBECOM)*. [S.l.: s.n.], 2018. p. 1–7. Cited on page 62.

ROTTENSTREICH, O.; TAPOLCAI, J. Optimal Rule Caching and Lossy Compression for Longest Prefix Matching. In: *Transactions on Networking, Volume:25, Issue:2*. [S.l.]: IEEE/ACM, 2017. Cited on page 71.

SHAH, R.; KUMAR, V.; VUTUKURU, M.; KULKARNI, P. Turboepc: Leveraging dataplane programmability to accelerate the mobile packet core. In: *Proceedings of the Symposium on SDN Research*. New York, NY, USA: Association for Computing Machinery, 2020. (SOSR '20), p. 83–95. ISBN 9781450371018. Available from Internet: <https://doi.org/10.1145/3373360.3380839>. Cited 2 times on pages 53 and 56.

SILVA, J. Santiago da; STIMPFLING, T.; LUINAUD, T.; FRADJ, B.; BOUGHZALA, B. One for all, all for one: A heterogeneous data plane for flexible p4 processing. In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2018. p. 440–441. Cited on page 57.

SINGH, S. K.; ROTHENBERG, C.; LUIZELLI, M. C.; ANTICHI, G.; PONGRACZ, G. Revisiting heavy-hitters: Don't count packets, compute flow inter-packet metrics in the data plane. In: *Proceedings of the SIGCOMM '20 Poster and Demo Sessions*. New York, NY, USA: Association for Computing Machinery, 2020. (SIGCOMM '20), p. 49–51. ISBN 9781450380485. Available from Internet: <https://doi.org/10.1145/3405837.3411388>. Cited 2 times on pages 58 and 62.

SINGH, S. K.; ROTHENBERG, C. E.; LUIZELLI, M. C.; ANTICHI, G.; GOMES, P. H.; PONGRáCZ, G. Hh-ipg: Leveraging inter-packet gap metrics in p4 hardware for heavy hitter detection. *IEEE Transactions on Network and Service Management*, v. 20, n. 3, p. 3536–3548, 2023. Cited on page 105.

SINGH, S. K.; ROTHENBERG, C. E.; PATRA, G.; PONGRACZ, G. Offloading virtual evolved packet gateway user plane functions to a programmable asic. In: *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*. New York, NY, USA: Association for Computing Machinery, 2019. (ENCP '19), p. 9–14. ISBN 9781450370004. Cited 4 times on pages 53, 56, 99, and 105.

SIVARAMAN, A.; KIM, C.; KRISHNAMOORTHY, R.; DIXIT, A.; BUDIU, M. DC.P4: Programming the Forwarding Plane of a Data-center Switch. In: *Symposium on Software Defined Networking Research (SOSR)*. [S.l.]: ACM, 2015. Cited on page 75.

SIVARAMAN, V.; NARAYANA, S.; ROTTENSTREICH, O.; MUTHUKRISHNAN, S.; REXFORD, J. Heavy-Hitter Detection Entirely in the Data Plane. In: *Symposium on SDN Research (SOSR)*. [S.l.]: ACM, 2017. Cited 6 times on pages 58, 71, 72, 74, 75, and 93.

STATISTA. *Cloud Gaming Worldwide report and prediction for upcoming years*. 2024. Https://fr.statista.com/outlook/amo/media/games/cloud-gaming/worldwide. Available from Internet: <https://fr.statista.com/outlook/amo/media/games/cloud-gaming/worldwide>. Cited on page 102.

STRATUM. *An open source silicon-independent switch operating system for software defined networks*. 2022. Available from Internet: <https://opennetworking.org/stratum/>. Cited on page 100.

SULTANA, N.; SONCHACK, J.; GIESEN, H.; PEDISICH, I.; HAN, Z.; SHYAMKUMAR, N.; BURAD, S.; DEHON, A.; LOO, B. T. Flightplan: Dataplane disaggregation and placement for p4 programs. In: *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021. p. 571–592. ISBN 978-1-939133-21-2. Available from Internet: <https://www.usenix.org/conference/nsdi21/presentation/sultana>. Cited on page 54.

SUN, Y.; XU, V.; AHLAWAT, S.; DAFTARY, K. *Measuring network performance using inter-packet gap metric*. [S.l.]: Google Patents, 2015. US Patent 9,178,783. Cited on page 75.

TANG, F.; Zhang, H.; Yang, L. T.; Chen, L. Elephant flow detection and differentiated scheduling with efficient sampling and classification. *IEEE Transactions on Cloud Computing*, p. 1–1, 2019. Cited on page 75.

TRex. *Cisco TRex Traffic Generator* . 2024. Available from Internet: <https://trex-tgn.cisco.com/>. Cited on page 93.

TURKOVIC, B.; OOSTENBRINK, J.; KUIPERS, F. Detecting Heavy Hitters in the Data-plane. In: *CoRR, abs/1902.06993*. [S.l.]: Cornell University, 2019. Cited on page 76.

VIEIRA, M. A. M.; CASTANHO, M. S.; PACÍFICO, R. D. G.; SANTOS, E. R. S.; JúNIOR, E. P. M. C.; VIEIRA, L. F. M. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 1, fev. 2020. ISSN 0360-0300. Cited on page 27.

VöRöS, P.; HORPáCSI, D.; KITLEI, R.; LESKó, D.; TEJFEL, M.; LAKI, S. T4p4s: A target-independent compiler for protocol-independent packet processors. In: *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. [S.l.: s.n.], 2018. p. 1–8. Cited on page 60.

WANG, Q.; ALCARAZ-CALERO, J.; RICART-SANCHEZ, R.; WEISS, M. B.; GAVRAS, A.; NIKAEIN, N.; VASILAKOS, X.; GIACOMO, B.; PIETRO, G.; RODDY, M.; HEALY, M.; WALSH, P.; TRUONG, T.; BOZAKOV, Z.; KOUTSOPOULOS, K.; NEVES, P.; PATACHIA-SULTANOIU, C.; IORDACHE, M.; OPROIU, E.; YAHIA, I. G. B.; ANGELO, C.; ZOTTI, C.; CELOZZI, G.; MORRIS, D.; FIGUEIREDO, R.; LORENZ, D.; SPADARO, S.; AGAPIOU, G.; ALEIXO, A.; LOMBA, C. Enable advanced qos-aware network slicing in 5g networks for slice-based media use cases. *IEEE Transactions on Broadcasting*, v. 65, n. 2, p. 444–453, 2019. Cited on page 56.

WEAVER D. TERPSTRA, S. M. V. M. Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations. *Int'l Symp. on Performance Analysis of Systems and Software*, 2013. Cited on page 48.

WHITEPAPER. *Intel, Kaloom Create P4-programmable Network Solutions*. 2020. Cited 2 times on pages 57 and 59.

XIAO, Q.; Tang, Z.; Chen, S. Universal online sketch for tracking heavy hitters and estimating moments of data streams. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2020. p. 974–983. Cited on page 75.

XIE, G.; LI, Q.; DUAN, G.; LIN, J.; DONG, Y.; JIANG, Y.; ZHAO, D.; YANG, Y. Empowering in-network classification in programmable switches by binary decision tree and knowledge distillation. *IEEE/ACM Transactions on Networking*, v. 32, n. 1, p. 382–395, 2024. Cited on page 106.

XIONG, Z.; ZILBERMAN, N. Do switches dream of machine learning? toward in-network classification. In: *ACM HotNets*. [s.n.], 2019. ISBN 9781450370202. Available from Internet: <https://doi.org/10.1145/3365609.3365864>. Cited 2 times on pages 103 and 108.

YANG, T.; JIANG, J.; LIU, P.; HUANG, Q.; GONG, J.; ZHOU, Y.; MIAO, R.; LI, X.; UHLIG, S. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In: *Special Interest Group on Data Communication (SIGCOMM)*. [S.l.]: ACM, 2018. Cited 8 times on pages 58, 71, 72, 74, 75, 84, 86, and 89.

YANG, T.; ZHANG, H.; LI, J.; GONG, J.; UHLIG, S.; CHEN, S.; LI, X. HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows. In: *Transactions on Networking, Volume:27, Issue:5*. [S.l.]: IEEE/ACM, 2019. Cited 5 times on pages 71, 72, 75, 86, and 89.

YOUSAF, F. Z.; LESSMANN, J.; LOUREIRO, P.; ; SCHMID, S. SoftEPC: Dynamic instantiation of mobile core network entities for efficient resource utilization. *Proc. IEEE Int. Conf. Commun. (ICC)*, IEEE Press, p. 3602–3606, 2013. Cited on page 42.

ZHANG, D.; ZHOU, Y.; XI, Z.; WANG, Y.; XU, M.; WU, J. Hypertester: High-performance network testing driven by programmable switches. *IEEE/ACM Transactions on Networking*, v. 29, n. 5, p. 2005–2018, 2021. Cited on page 114.