User-Plane Algorithms for Stateless and Stateful Inference in Programmable Networks

Aristide Tanyi-Jong Akem

IMDEA Networks Institute, Spain

Homepage: https://akemaristide.github.io/

Thesis: User-Plane Algorithms for Stateless and Stateful Inference in Programmable Networks

Advisor: Marco Fiore, IMDEA Networks Institute, Spain

Brief Biography: Akem is a research assistant at IMDEA Networks Institute and a recent graduate from the Telematics Engineering PhD program at Universidad Carlos III de Madrid, Spain. His skills and research interests cut across machine learning, network programming, and mobile networking, as well as their applications to other domains like network security, IoT, and energy. During his PhD, Akem has developed several solutions for deploying machine learning models into P4-programmable hardware for high-speed inference. He also made research visits to Orange Labs in France, Ranplan Wireless and the University of Cambridge, both in the United Kingdom.

1 Introduction

In the last decade, the complexity of networks has increased significantly to accommodate the rise of innovative applications. This growing complexity has rendered traditional human-in-the-loop network management approaches inadequate, necessitating greater automation and flexibility in managing these networks. The introduction of Software-Defined Networking (SDN) with a programmable control plane marked a major advancement in this direction, enabling a wide range of network automation applications to be executed within the SDN control plane.

Machine Learning (ML) algorithms have become essential for automating the planning, deployment, and management of modern mobile and computer networks [1]. In SDN, ML models are typically executed in the control plane or on external servers [2]. However, these models require back-and-forth communication with the user plane to run real-time inference, which leads to significant latency, often in the range of tens to hundreds of milliseconds [3]. This delay makes them inadequate for applications with stringent latency demands, such as augmented and virtual reality (AR/VR) [4].

Recent advancements in programmable user planes, with hardware such as Intel Tofino ASICs [5] and NVIDIA BlueField DPUs [6], and network programming languages like P4 [7], have spurred interest in offloading ML models to the user plane for line-rate inference with potentially sub-microsecond latency. However, implementing ML models directly in the user plane presents significant challenges due to three key limitations [8]. First, user plane devices like switches use expensive memory like TCAM and SRAM which are available in very limited amounts, restricting data storage capacity. Second, to ensure high-speed packet processing, the number of operations that can be performed on each packet is highly constrained, often only one. Third, the supported mathematical operations are limited to addition, subtraction, bit shifts, and logical operations. This means that floating point operations cannot be performed within these devices. The P4 programming language introduces additional constraints, such as the absence of loops and the inability to inspect packet payloads, which further complicates ML deployment in the user plane. These limitations make it infeasible to fully train ML models within the switch using P4, restricting user-plane applications to deploying pre-trained models for line-rate inference [8]. Consequently, the fundamental question arises: *How can ML models be effectively deployed in programmable user-plane equipment for high-speed inference while accounting for the above constraints?*

The primary aim of this PhD project is to address the aforementioned question, contributing to the broader goal of realizing self-driving networks [9]. This is achieved through the development of solutions for embedding data-driven models into the user plane, facilitating the acceleration of offloadable network management tasks. The research question is divided into three specific sub-questions: (*i*) Which ML models are most suitable for user plane inference? (*ii*) Which user-plane components should be targeted for model deployment? and (*iii*) How can the constraints of user plane devices be effectively accommodated?

During the PhD, answers were sought to the above questions leading to several design choices for the contributions that were made. The questions were tackled as follows.

(i) Which ML models are most suitable for deployment in the user plane?
A key aim of this thesis is to maintain simplicity in models designed for user-plane inference to ensure compatibility with user-plane environments. Therefore, Decision tree (DT) and Random Forest (RF) models are chosen. Unlike complex neural networks (NNs) requiring multiple operations, tree-based models rely on simple comparisons between feature values and thresholds at each tree node to reach an inference decision. More so, these models either match or surpass NNs in tabular data inference tasks, achieving a better accuracy-complexity tradeoff [10]. This makes them the ideal models for user-plane deployment.

Copyright is held by author/owner(s). Publication rights licensed to ACM. The definitive Version of Record was published in ACM SIGMETRICS Performance Evaluation Review, 52(3), DOI: 10.1145/3712170.371217.

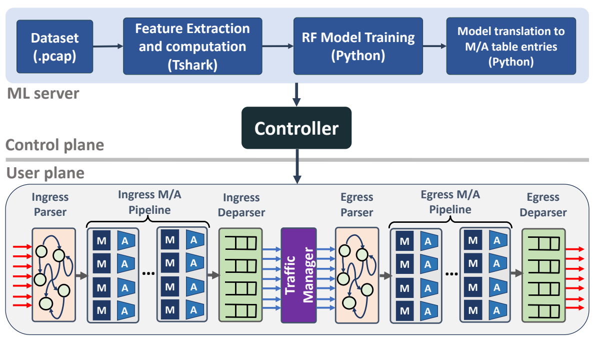

Figure 1: Overview of the user plane inference workflow.

(ii) Which user-plane components should be targeted for model deployment?
A range of off-the-shelf programmable user-plane targets is available, including ASICs like Intel Tofino [5], and SmartNICs such as Netronome Agilio [11]. Each has distinct advantages and limitations. SmartNICs installed in dedicated servers could support complex models like NNs but only allow inference at specific locations. Switches on the other hand, while widely deployed and capable of in-network inference, are more constrained in memory and operations, restricting model complexity. However, switches excel in performance, throughput, and cost-effectiveness, supporting high data rates, making them ideal for user-plane inference, and justifying why they are chosen in this thesis.

(iii) How can the constraints of user plane devices be effectively accommodated?
The design of solutions for in-switch inference must take into account the various constraints of programmable user planes. Many existing approaches face performance and scalability challenges primarily because they fail to accommodate the stringent limitations of specific user-plane targets. This thesis presents several techniques and adaptations aimed at overcoming these constraints, enabling scalable solutions for performing line-rate inference in programmable switches using DT and RF models.

User-plane inference with ML is a relatively new area of research, with most existing works proposed over the last five years, as recently surveyed [12, 13]. The majority of prior work focuses on classification, used for tasks like network intrusion detection, device identification, and service fingerprinting. This thesis also focuses on classification, which runs fully in the user plane and is analogous to the forwarding role of the switch, which in essence, is the classification of packets to different forwarding ports. The rest of this work summarizes the thesis which is available in full online [14].

2 Methodology

We focus on deploying Decision Tree (DT) and Random Forest (RF) models in programmable switches because their simplicity makes them ideal for in-switch operation. These models rely solely on logical comparisons between feature values and node thresholds, which fits well with the highly constrained environments of switches. Switches, in turn, can facilitate widespread, high-speed inference throughout the network. The following sections detail the workflow adopted for our methodology (§2.1), the software tools and frameworks used (§2.2), and the experimental hardware setup we used to test our solutions (§2.3).

2.1 In-switch inference workflow

Figure 1 shows the adopted workflow for deploying models into the user plane. It is divided into three parts based on where the processes are running.

ML Server. As in all ML tasks, the ML model preparation starts with the acquisition of the target dataset typically as packet captures in *.pcap* format. We employ Tshark [15] to extract packet header fields which are used as features in packet-level (PL) inference or to compute features in the case of flow-level (FL) inference. With the computed features now saved in *.csv* files, the dataset is used to train the models using Scikit-Learn [16] Python-based libraries. The final model is then transformed into Match & Action (M/A) table entries that will be sent to the controller as shown in Figure 1.

Controller. The trained model is injected into the switch P4 program by the controller via a control plane specification *e.g.*, the P4Runtime API. The controller does not take part in the inference process which instead happens fully in the switch. However, it receives information from the switch via packet digests which bring statistics or results from the switch, based on which it can modify table entries to change the behaviour of the switch program.

User Plane. The entire inference process takes place in the switch *i.e.*, in the user plane which is portrayed at the bottom of Figure 1 as a Protocol Independent Switch Architecture (PISA) pipeline which is adopted by most user plane targets. A P4 program is written based on the model that is to be deployed. The parser is programmed to extract all header fields that serve as model features. In the first part of the M/A pipeline which is the ingress pipeline, M/A tables are defined for the model and any logic required is implemented too. The nature of tables and/or computations depends on which inference solution is being deployed, as we will show in §3 and §4. Upon deployment, packets arriving the switch are parsed and features are extracted. In the ingress, the model is applied and a decision is reached. The packet is then forwarded or dropped based on the result. Additional processing could be applied in the egress pipeline if needed.

2.2 Software tools

As described in §2.1, we use TShark [15], to extract features from *.pcap* files. We then use Python for the data analysis, feature selection, model hyper-parameter tuning, model training (using Scikit-Learn [16]) and model translation into M/A table entries. For writing programs that implement the models in the switch, we use P4. At the early stages of the PhD when we had no hardware components, we used BMv2 as a software target to debug and test our P4 code. This switch was deployed within Mininet [17] alongside a few hosts for sending traffic through the switch via Tcpreplay [18] from one host and capturing it on another using Tcpdump. To generate background traffic and increase the heterogeneity of our test environments, we used the Moongen [19] traffic generator to produce and inject Gbps traffic. Background traffic is however not inferred upon and only serves to generate a traffic mix that better resembles real-world scenarios.

2.3 Hardware setup

After a year of experimentation with BMv2 switches, we set up a hardware testbed consisting of two servers and three programmable switches. The servers are equipped with Intel 8-core Xeon processors running at 2GHz, 48GB of RAM, and 100Gbps QSFP28 interfaces. The switches are Edgecore Wedge 100BF-QS models featuring Intel Tofino BFN-T10-032Q chipsets with 32 100-GbE QSFP28 ports. They operate on the Open Network Linux (ONL) OS, along with Intel's Software Development Emulator (SDE) for compiling P4 programs for the Tofino Native Architecture (TNA) [5]. In our experiments, we implement models as P4 programs within switches. We also run a controller instance and a traffic sink on one server while using another server as a traffic source to inject test traffic and background traffic into the switch via Tcpreplay.

3 Stateless inference

In stateless inference, the in-switch model infers on individual packets using features extracted directly from packet headers without storing any information from previous packets. This thesis proposed two solutions for stateless inference in programmable switches.

3.1 Rapid cyberattack detection in smart grids

We demonstrate how in-switch inference enables the rapid detection of cyberattacks on SDN-based Smart Grid (SG) networks. Modern power grids are "smart", connecting millions of devices through data networks, which exposes them to cyberattacks that could result in outages or data breaches. Hence, timely cyberattack detection is crucial. ML models are commonly employed in SDN-based SGs for detecting cyberattacks, but these models often run on external servers or within the network's control plane, leading to millisecond-level detection delays. The application presented in this thesis shows how ML inference at the Packet Level (PL) in programmable switches accelerates the detection and mitigation of attacks in SGs, achieving line-rate performance with sub-microsecond delays [20]. This work introduces the concept of user-plane inference into SDN-based SGs for the first time, deploying a trained DT model within the switch pipeline to perform real-time inference on live network traffic. The results produced by this thesis demonstrate that a fully user-plane solution achieves up to 99% accuracy in attack detection and classification while operating up to four orders of magnitude faster than control-plane methods.

3.2 Henna

The above solution and all prior work on PL inference in the user-plane focus on flat classification, and have significant structural limitations that prevent them from scaling when handling complex inference tasks. To tackle these limitations, this thesis proposes `Henna`, the pioneer implementation of an in-switch multi-stage hierarchical classification system. The concept upon which `Henna` hinges is that of splitting a difficult classification task into easier cascaded inference tasks, which can then be addressed with separate resource-efficient tree-based classifiers. The design of `Henna` aligns with the internal organization of programmable switch architecture and integrates state-of-the-art strategies for mapping decision trees to switch hardware. `Henna` is then implemented into a real-world testbed with off-the-shelf Intel Tofino programmable switches using the P4 language. Experiments with a complex 21-category classification task based on measurement data exhibit how `Henna` improves the F1-score of an advanced single-stage model by 21%, while maintaining usage of switch resources at 8% on average.

4 Stateful inference

Although PL inference models are substantially easier to embed in programmable user planes, stateful inference is much more relevant in the majority of networking tasks since most packets are related to each other and hence can be processed together as flows which provides a more contextual understanding of network traffic [21] by leveraging interesting insights about the relationships between the packets. However, deploying models in switches for stateful inference constitutes a significant challenge as it requires maintaining state and computing stateful features in resource-constrained devices. This challenge is tackled in this thesis with the proposal of two solutions.

4.1 Flowrest

Despite the advancements brought about by Henna and other pre-2023 solutions, it was still subject to the accuracy barriers suffered by PL solutions, and it also did not employ any FL features. As such, there was still no practical solution for running inference at FL in hardware switches with RFs. Prior FL works were either not fully tested in hardware (pForest [22] and SwitchTree [23]), or where use-case specific (pHeavy [24]). In response, we proposed Flowrest [25], the first comprehensive framework for deploying FL models into hardware switches, accounting for the constraints of the switches right from the design phase of the models. The proposed solution builds on (*i*) novel guidelines for tailoring RF models to operation in programmable switches right from the design phase, (*ii*) an original framework to embed flow-level (FL) machine learning models into programmable switch ASICs, and (*iii*) efficient strategies for maintaining state within switches to compute, store and employ FL features for inference. `Flowrest` sets a new standard for FL inference in the user plane. To validate this claim, a thorough evaluation of the proposed solution is conducted in an experimental platform based on Intel Tofino switches in two steps; (*i*) `Flowrest` is evaluated on unencrypted traffic, comparing it to major existing proposals for in-switch inference which all target unencrypted traffic, and (*ii*) it is then evaluated on encrypted traffic classification [26]. Results from the evaluation with tasks of unprecedented complexity show how `Flowrest` achieves accuracy gains in the $10\% - 39\%$ range over previous approaches to implement DT and RF models in real-world equipment.

4.2 Jewel

Despite the improved performance resulting from FL classification, a major dichotomy still exists between works for in-switch inference, based on whether they operate at PL or FL. The former relies on simple features from packet headers that are simple to implement but limit accuracy in challenging use cases; the latter exploits richer flow-based statistical features to improve accuracy, but leaves early packets in each flow unclassified. To bridge this gap, NetBeacon [27] was proposed as a hybrid solution for simultaneous PL+FL inference, using multiple PL and FL models deployed in the switch. However, deploying multiple models greatly increased switch resource consumption. Instead, this thesis presents `Jewel` [28], an in-switch ML solution based on a fully joint PL and FL design, which offers the best of both worlds by classifying early flow packets individually at PL and shifting to FL inference as soon as possible. The proposed solution involves (*i*) a single RF model trained to classify both packets and flows, and (*ii*) hardware-aware model selection and training techniques for resource footprint minimization. `Jewel` is implemented in P4 and deployed in a testbed with Intel Tofino switches, where extensive experiments are conducted with a variety of real-world use cases. Results from experiments conducted in this thesis reveal how `Jewel` outperforms four state-of-the-art benchmarks, with absolute accuracy gains in the $2.0\% - 5.3\%$ range, while consuming a modest amount of switch resources.

5 Future work

Future work could focus on integrating user-plane inference into real networks for tasks like traffic classification, routing optimization, real-time anomaly detection, load balancing, and much more. Exploring other hardware targets such as FPGAs and SmartNICs, could offer more flexibility and enable the deployment of more complex models. Additionally, evaluating stateful FL inference at high speeds will help assess scalability. Reducing memory usage remains an essential aspect to investigate. Also, distributed inference scenarios where models are split across devices present a promising path but require solutions for model coordination and communication between model components. Lastly, new technologies like eBPF could be leveraged to further push the boundaries of network management by deploying applications for network security, observability, and management in the Linux kernel space, as is already being done by large corporations, *e.g.*, Google, Cloudflare, Android, and Netflix.

6 Representative Papers

[1] Flowrest: Practical Flow-Level Inference in Programmable Switches with Random Forests (IEEE INFOCOM 2023) with M. Gucciardo, and M. Fiore

[2] Jewel: Resource-Efficient Joint Packet and Flow Level Inference in Programmable Switches (IEEE INFOCOM 2024) with B. Bütün, M. Gucciardo, and M. Fiore

[3] Henna: hierarchical machine learning inference in programmable switches (Native'NI 2022) with B. Bütün, M. Gucciardo, and M. Fiore

[4] Ultra-Low Latency User-Plane Cyberattack Detection in SDN-based Smart Grids (ACM e-Energy 2024) with M. Gucciardo, and M. Fiore

7 References

[1] C. Kilinc et al. 5G development: Automation and the role of artificial intelligence. *Wiley 5G Ref*, 5 2020.

[2] A. A. Gebremariam et al. Applications of artificial intelligence and machine learning in the area of SDN and NFV: A survey. *SSD*, 2019.

[3] K. He et al. Measuring control plane latency in SDN-enabled switches. In *SOSR*. ACM, 2015.

[4] P. Lincoln et al. From motion to photons in 80 microseconds: Towards minimal latency for virtual and augmented reality. *IEEE Trans. Vis. Comput. Graph.*, 22(4):1367–1376, 2016.

[5] Tofino Programmable Ethernet Switch ASIC. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html.

[6] NVIDIA BlueField Networking Platform. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[7] P. Bosshart et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, jul 2014.

[8] A. Sapio et al. In-network computation is a dumb idea whose time has come. In *HotNets*, 2017.

[9] N. Feamster and J. Rexford. Why (and how) networks should run themselves. *CoRR*, abs/1710.11583, 2017.

[10] L. Grinsztajn et al. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS 2022*, November 2022.

[11] Netronome. Netronome Agilio SmartNICs.

[12] R. Parizotto et al. Offloading machine learning to programmable data planes: A systematic survey. *ACM Comput. Surv.*, jun 2023.

[13] C. Zheng et al. In-network machine learning using programmable network devices: A survey. *IEEE Commun. Surv. Tutor.*, 2023.

[14] A. T.-J. Akem. *User-Plane Algorithms for Stateless and Stateful Inference in Programmable Networks*. PhD thesis, Universidad Carlos III de Madrid, Spain, 2024. https://hdl.handle.net/20.500.12761/1853.

[15] Wireshark. https://www.wireshark.org/tshark.

[16] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12, 2011.

[17] Bob Lantz et al. Mininet: An instant virtual network on your laptop. http://mininet.org/, 2017.

[18] A. Turner and F. Klassen. Tcpreplay. https://tcpreplay.appneta.com/.

[19] P. Emmerich et al. Moongen: A scriptable high-speed packet generator. In *IMC*, NY, USA, 2015.

[20] A. T.-J. Akem et al. Ultra-low latency user-plane cyberattack detection in sdn-based smart grids. In *ACM e-Energy*, pp. 676–682, 2024.

[21] M. Hayes et al. Scalable architecture for SDN traffic classification. *IEEE Systems Journal*, 12, 2018.

[22] C. Busse-Grawitz et al. pForest: In-network inference with random forests. *CoRR*, abs/1909.05680, 2019.

[23] J. Lee and K. P. Singh. SwitchTree: in-network computing and traffic analyses with random forests. *Neural. Comput. Appl.*, 2020.

[24] X. Zhang et al. pHeavy: Predicting heavy flows in the programmable data plane. *IEEE Trans. Netw. Service Manag.*, 18(4):4353–4364, 2021.

[25] A. T.-J. Akem et al. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM*, 2023.

[26] A. T.-J. Akem et al. Encrypted traffic classification at line rate in programmable switches with machine learning. In *IEEE/IFIP NOMS*, 2024.

[27] G. Zhou et al. An efficient design of intelligent network data plane. In *USENIX Security*, 2023.

[28] A. T.-J. Akem et al. Jewel: Resource-efficient joint packet and flow level inference in programmable switches. In *IEEE INFOCOM*, 2024.