Towards Data-Driven Management of Mobile Networks through User Plane Inference

Aristide Tanyi-Jong Akem*† and Marco Fiore*‡
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain
{aristide.akem, marco.fiore}@imdea.org
‡*Supervisor*

***Abstract*—Growing network complexity has rendered human-in-the-loop network management approaches obsolete. The advent of Software-Defined Networking (SDN) has enabled network automation, with Machine Learning (ML) models running in the control plane. However, such control plane models do not run at line rate and would not satisfy the stringent latency requirements of time-sensitive next-generation applications. In this PhD project, we exploit recent advances in programmable switches and associated languages like P4 to enable data-driven management of networks by running ML models for inference in programmable switches at line rate, with high throughput and low latency. Resulting contributions include solutions for in-switch classification at packet level, flow level, or both, with use cases in network security, service identification, and device fingerprinting in commercial off-the-shelf switches.**

***Index Terms*—In-switch inference, machine learning, P4**

I. Introduction

Machine Learning (ML) algorithms have become key players for automatic planning, deployment and management of modern high-speed networks mobile and computer networks [1]. Such automation is required to meet up with the growing complexity of these networks which has made human-in-the-loop solutions to network issues impractical. In the context of Software-Defined Networking (SDN), ML models for network automation are traditionally run in the control plane or in offline ML severs [2]. However, such control plane models require communication with the user plane to run inference in real-time on live network traffic, thereby inducing tens of hundreds of miliseconds of latency [3]. Hence, these models fall short of meeting the strict latency requirements of complex applications with extremely low latency requirements such as augmented and virtual reality (AR/VR) [4].

In recent years, SDN user planes have become programmable with commercial off-the-shelf products like Intel Tofino ASICs [5] and NVIDIA BlueField data processing units (DPU) [6] becoming available on the market, alongside domain-specific network programming languages such as P4 [7]. This has sparked a strong interest in offloading ML models to the user plane in order to achieve line rate inference that can potentially reduce latency to sub-microsecond levels. Yet, porting models to user planes is a daunting task owing to three inherent constraints of such equipment [8]. First, user planes like switches have limited memory, typically in the order of tens of megabytes of SRAM and hundreds of kilobytes of TCAM. This limits the amount of data that can be stored on them. Second, in order to support high-speed packet processing, most operations can only be performed once per packet. Last, there is limited support for mathematical operations, with only addition, subtraction, bit shifts and logical operations supported. In addition, the P4 language also has its own constraints like the absence of loops and the inability to inspect packet payloads which further complicate the deployment of ML in the user plane.

The constraints above rule out the possibility of training ML models fully in the switch with P4, and limit user plane ML applications to the deployment of pre-trained ML models into the user plane for line rate inference [8]. This gives rise to a key question: *how can machine learning models be deployed in programmable user plane equipment for high-speed inference, while accounting for the constraints of these equipment?*

The main objective of this PhD project is to answer the above question and contribute to the global vision of self-driving networks [9], through the design of solutions for embedding data-driven models into the user plane for the acceleration of offloadable network management tasks. To that effect, the research question is split into three related sub-questions; (i) what user plane components should be targeted for model deployments? (ii) what ML models are most appropriate for user plane inference? and (iii) how can the constraints of user plane equipment be accommodated?

Over the past three years, we have sought answers to the above questions by exploring the literature (§II), designing and adopting suitable methodologies (§III), developing inference solutions and techniques for accommodating hardware constraints in model training and deployment (§IV-A−§IV-C), and demonstrating our solutions in realistic use cases (§IV-D).

II. Related Work

User plane inference with ML is a relatively new area of research, with most existing works carried out over less than five years ago as recently surveyed [10], [11]. Majority of prior work focuses on classification, which is used for a variety of tasks like attack detection, device identification, and service fingerprinting. We also focus our attention on solutions for classification that run fully in the user plane, leaving out those that split the task between planes *e.g.*, Flowlens [12], or those focusing on other tasks like clustering *e.g.*, Clustreams [13]. User plane solutions differ in terms of the target user plane equipment *e.g.*, switches, FPGAs or SmartNICs; the nature of the ML model *e.g.*, decision tree or neural network; and the inference target *i.e.*, packets or flows. We discuss these next.

FPGAs. One of the earliest proposals for user plane inference is IISy [14] which implemented a DT model in a NetFPGA-SUME [15] hardware and in a the BMv2 software switch. However, NetFPGA-SUME has since reached its end-of-life and development in BMv2 is oblivious of hardware constraints. Taurus [16] proposes to use FPGA hardware to enhance the capabilities of switches and run complex Deep Neural Network (DNN) models. By offloading the model execution from switches to the external FPGA, the constraints of switches are eliminated, and more possibilities are unleashed. Yet, adopting Taurus in networks will entail significant added costs to purchase and deploy additional equipment alongside already expensive switches and NICs.

Switches. Most of the work on in-network inference has focused on programmable switches, with different kinds of models deployed. One of the earliest works is N2Net [17], which is a patented technology [18] for in-switch deployment of Binarized Neural Networks (BNNs), which are Neural Networks (NNs) with $+1/-1$ weights and sign activations [19]. While BNNs are simpler than full NNs, deploying them into commodity switches is not practical, as a basic BNN with two layers of 64 and 32 neurons completely exhausts the resources of a Tofino ASIC [17].

The vast majority of works have focused on in-switch inference with Decision Tree (DT) and Random Forest (RF) ML models, which are multiple DTs in parallel. Major proposals include IIsy [14], pForest [20], SwitchTree [21], Planter [22], Xavier *et al.* [23], Mousika [24], Henna [25], Soter [26], Bütün *et al.* [27], Flowrest [28], NetBeacon [29], Akem *et al.* [30], and Jewel [31]. Given the relatively low complexity of DTs and RFs, these solutions all opt to deploy these trained models in a an off-the-shelf programmable switch to process packets at line rate with use cases in security and traffic classification on individual packets or flows[1].

We remark that Ilsy and Planter also discuss implementing other models in switches *e.g.*, SVM, XGBoost, Naive Bayes, K-Means, or Isolation Forest but DTs and RFs are reported to offer higher scalability and better performance. Programmable switches also emerge as the favourite target due to their ability to enable ubiquitous inference in the network with multiple high-speed ports. As such, this PhD project also focuses on deploying DTs and RFs into switches for in-network inference.

SmartNICs. The inherent constraints of switches which make NN deployment in them not practical have led to the exploration of alternative user plane hardware which are more adpted to such models. In that light, N3IC [32] presents a comprehensive approach for integrating of BNNs on SmartNICs, using both the micro-C language (for Netronome system-on-chip NICs) and P4 (for NetFPGAs NICs configured with a PISA architecture). SmartNICs mitigate the constraints of programmable switches, since they offer much more computational resources and increased memory size. Using them, N3IC implements a 3-layer BNN that runs at line-rate inference while consuming a relatively small amount of the available resources. However, SmartNICs are almost as costly as switches, while offering much fewer ports. In addition, they are located within appliances in the network and can only enable inference at those locations, unlike switches which instead are everywhere in the network. This further reinforces the choice of switches as target for inference tasks.

[1]A flow is a group of packets with the same 5-tuple of source and destination IP addresses, source and destination ports, and transport protocol.

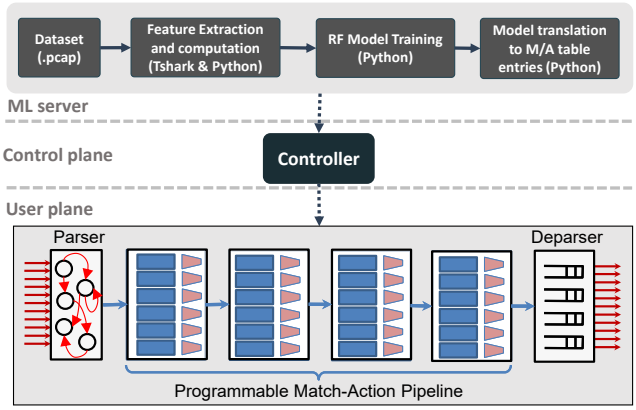

Figure 1: Overview of the user plane inference workflow.

III. Methodology

We recall from §II that we focus on the deployment of DT and RF models in programmable switches due to the simplicity of these models that make them more adapted to in-switch operation, and the potential of switches to enable ubiquitous high-speed inference in the network. In fact, DTs and RFs only require logical comparisons of feature values to the thresholds at tree nodes. They are therefore the most suitable for highly constrained environments like the switch. Next we describe the workflow adopted for our methodology (§III-A), the software tools and frameworks we employ (§III-B), and lastly the experimental hardware tested we setup (§III-C).

A. 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 [33] 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 [34] 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 [35]. 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 §IV. 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.

B. Software tools employed

As seen in §III-A, we use TShark [33], 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 [34]) 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 [36] alongside a few hosts for sending traffic through the switch via Tcpreplay [37] 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 [38] 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.

C. Hardware testbed setup

After a year of learning and experimenting on BMv2 switches, we built a hardware testbed comprising two off-the-shelf servers and three programmable switches. The servers have Intel 8-core Xeon processors at 2GHz clock frequency, with 48GB of RAM, and 100Gbps QSFP28 interfaces. The switches are the Edgecore Wedge 100BF-QS model, with Intel Tofino BFN-T10-032Q chipsets and 32 100-GbE QSFP28 ports each. They run the Open Network Linux (ONL) operating system, and Intel's Software Development Environment (SDE) that we use for compiling P4 programs for the Tofino Native Architechture (TNA) [5]. During experiments, we implement the model in the switch as a P4 program and then run a controller instance as well as a traffic sink in one of the servers. We then employ another server as a traffic source to inject the test traffic to the switch via Tcpreplay.

IV. Proposed Solutions

Recent surveys [10], [11] reveal that the bulk of user plane ML inference solutions were developed between $2022-2024$.

Figure 2: Publication timeline of proposed solutions.

Thus, at beginning of the PhD in March 2021, the only available solutions were N2Net [17], IIsy [14], pForest [20], and SwitchTree [21]. Soon after that, Planter [22] was released as an extension of IISy [14], followed by Xavier *et al.* [23], pHeavy [39] and Mousika [24]. These solutions made contributions that advanced the state-of-the-art but left many gaps which we proposed several solutions to fill over the past years as shown in Figure 2. Next, we detail these proposed solutions.

A. Hierarchical packet-level inference

Most of the above solutions were designed for PL inference *i.e.*, to infer on individual packets. As these models employ only simple header fields as features, they often fail to achieve high accuracy in complex classification tasks, or lead to models that are too resource-hungry to be deployed in the switch. To remedy the situation, we proposed Henna [25], a framework for two-stage hierarchical ML inference in switches with DT and RF models. We tackle difficult tasks by splitting them into smaller tasks which individually are easier to solve with simpler models that are easier to deploy in the switch. Results from experiments on our testbed showed that in a device identification task, Henna outperformed a fully grown single-stage classify by up to 21% in terms of F1 score, while consuming less than 10% on average of switch resources, thereby setting a new standard for solving complex problems with inherent hierarchical relationships between the classes.

B. Practical flow-level inference

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 [20] and SwitchTree [21]), or where use-case specific (pHeavy [39]). In response, we proposed Flowrest [28], 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. We implemented Flowrest in the P4 language in our hardware testbed and conducted multiple experiments to evaluate it against a PL benchmark which was representative of previous PL solutions like Planter [22] or Mousika [24]. Results demonstrated the superiority of Flowrest over PL solutions in all use cases, achieving F1-scores of up to 99% in some cases.

C. Joint packet- and flow-level inference

While PL solutions hit performance barriers in complex tasks, FL solutions perform better but leave the first few packets of flows unclassified during the FL feature computation

stage. These missed packets could have diverse effects depending on the use case, including a possible malware infection in security applications. To bridge this gap, NetBeacon [29] 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, we proposed Jewel [31] to tackle joint PL+FL inference using a single RF model trained to infer on early packets at PL and then switch to FL once FL statistics are good enough. We conducted experiments and evaluated Jewel against Mousika, Planter, Flowrest and NetBeacon, which were all outperformed on 4 different tasks, while consuming just about the same resources as Flowrest.

D. Use case-specific solutions

In our first use case paper [27], we explored how our user-plane inference solutions could can be leveraged to identify malicious IoT traffic in the Metaverse at line rate at at PL, ensuring a faster reaction than existing approaches where attack detection happens in the control plane. Building on Henna [25], we implemented and validated the solution through experiments with an IoT-based cyberattack dataset. Results showed how our solution achieved up to 99% accuracy.

Building on Flowrest [28], we proposed a second use case paper [30] in which we describe an in-switch encrypted traffic classification solution, that uses only features based on packet size and packet interarrival times. This ensures that the models remain robust in the face of encryption since the derived features are typically unaffected by encryption. We evaluated the solution over 3 use cases, including an encrypted instant messaging application classification use case with 6 classes, where it achieves up to 90% F1-score.

V. Conclusions and Future Work

The goal of this PhD project is to develop data-driven solutions for network management, which run at line rate in the data plane. We situated the project within the context of the low latency requirements of next-generation network applications, and presented its progress through the results that have been published to date. The results demonstrate the potential of in-switch machine learning to contribute towards automatic network management, and take us closer to the ultimate goal of self-driving mobile and computer networks.

Future research will address the following questions: (i) can other data plane targets and model types be exploited to enable inference in heterogeneous and/or decentralized scenarios with a mix of targets? (ii) can we identify other use cases of in-switch inference that are still unexplored? (iii) what will it take for the developed solutions to be practical for deployment in production mobile networks?

Acknowledgment

This work has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under grant agreement no. 101139270 "ORIGAMI" and the Marie Skłodowska-Curie grant agreement no. 860239 "BANYAN".

References

[1] C. Kilinc et al. 5G development: Automation and the role of artificial intelligence. *Wiley 5G Ref*, pp. 1–29, 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*, NY, USA, 2015. ACM.
[4] P. Lincoln et al. From motion to photons in 80 microseconds: Towards minimal latency for virtual and augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 22(4):1367–1376, 2016.
[5] Tofino Programmable Ethernet Switch ASIC. https://shorturl.at/mqtV2.
[6] NVIDIA BlueField Networking Platform. https://shorturl.at/fmDKO.
[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'17*, NY, USA, 2017. ACM.
[9] N. Feamster and J. Rexford. Why (and how) networks should run themselves. *CoRR*, abs/1710.11583, 2017.
[10] R. Parizotto et al. Offloading machine learning to programmable data planes: A systematic survey. *ACM Comput. Surv.*, jun 2023.
[11] C. Zheng et al. In-network machine learning using programmable network devices: A survey. *IEEE Commun. Surv. Tutor.*, 2023.
[12] D. Barradas et al. Flowlens: Enabling efficient flow classification for ML-based network security applications. In *NDSS*, 2021.
[13] R. Friedman et al. Clustreams: Data plane clustering. In *SOSR*, 2021.
[14] Z. Xiong and N. Zilberman. Do switches dream of machine learning? toward in-network classification. In *HotNets 2019*. ACM, 2019.
[15] N. Zilberman et al. NetFPGA SUME: Toward 100 Gbps as research commodity. *IEEE Micro*, 34(5):32–41, September 2014.
[16] T. Swamy et al. Taurus: A data plane architecture for per-packet ML. *ASPLOS*, 2022.
[17] G. Siracusano and R. Bifulco. In-network neural networks. *CoRR*, 2018.
[18] NEC Laboratories Europe. Using programmable switching chips as artificial neural networks engines. *US Patent: US20190102672A1*, 2019.
[19] M. Courbariaux et al. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NeurIPS*, volume 28, 2015.
[20] C. Busse-Grawitz et al. pForest: In-network inference with random forests. *CoRR*, abs/1909.05680, 2019.
[21] J. Lee and K. P. Singh. SwitchTree: in-network computing and traffic analyses with random forests. *Neural. Comput. Appl.*, 2020.
[22] C. Zheng and N. Zilberman. Planter: Seeding trees within switches. In *SIGCOMM '21*, pp. 12–14, NY, USA, 2021. ACM.
[23] B. M. Xavier et al. Programmable switches for in-networking classification. In *IEEE INFOCOM 2021*, pp. 1–10.
[24] G. Xie et al. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *INFOCOM 2022*.
[25] A. T.-J. Akem et al. Henna: Hierarchical machine learning inference in programmable switches. In *NativeNI 22*, pp. 1–7. ACM, 2022.
[26] G. Xie et al. Soter: Deep learning enhanced in-network attack detection based on programmable switches. In *SRDS*, 2022.
[27] B. Bütün et al. Fast detection of cyberattacks on the metaverse through user-plane inference. In *IEEE MetaCom*, pp. 350–354, 2023.
[28] A. T.-J. Akem et al. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM 2023*.
[29] G. Zhou et al. An efficient design of intelligent network data plane. In *32nd USENIX symposium on security*, 2023.
[30] A. T.-J. Akem et al. Encrypted traffic classification at line rate in programmable switches with machine learning. In *NOMS 2024*.
[31] A. T.-J. Akem et al. Jewel: Resource-efficient joint packet and flow level inference in programmable switches. In *INFOCOM*, 2024.
[32] G. Siracusano et al. Re-architecting traffic analysis with neural network interface cards. In *NSDI*, 2022.
[33] Wireshark. https://www.wireshark.org/docs/man-pages/tshark.html.
[34] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.
[35] P4Runtime Specification. https://shorturl.at/nryEX.
[36] Bob Lantz et al. Mininet: An instant virtual network on your laptop (or other pc). http://mininet.org/, 2017.
[37] A. Turner and F. Klassen. Tcpreplay. https://tcpreplay.appneta.com/.
[38] P. Emmerich et al. Moongen: A scriptable high-speed packet generator. IMC '15, pp. 275–287, NY, USA, 2015. ACM.
[39] X. Zhang et al. pHeavy: Predicting heavy flows in the programmable data plane. *IEEE Trans. Netw. Service Manag.*, 18(4):4353–4364, 2021.