Demonstrating Flow-Level In-Switch Inference

Michele Gucciardo*, Aristide Tanyi-Jong Akem*†, Beyza Bütün*†, Marco Fiore*
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain
{michele.gucciardo, aristide.akem, beyza.butun, marco.fiore}@imdea.org

***Abstract*—Existing approaches for in-switch inference with Random Forest (RF) models that can run on production-level hardware do not support flow-level features and have limited scalability to the task size. This leads to performance barriers when tackling complex inference problems with sizable decision spaces. `Flowrest` is a complete RF model framework that fills existing gaps in the existing literature and enables practical flow-level inference in commercial programmable switches. In this demonstration, we exhibit how `Flowrest` can classify individual traffic flows at line rate in an experimental platform based on Intel Tofino switches. To this end, we run experiments with real-world measurement data, and show how `Flowrest` yields improvements in accuracy with respect to solutions that are limited to packet-level inference in programmable hardware.**

I. Context and motivation

Machine learning (ML) models have become key enablers of automation in networking scenarios with applications in areas such as traffic classification, quality of service (QoS) prediction, and routing optimization. In the traditional software-defined network (SDN) paradigm, the ML models are trained and executed in the control plane. However, such models do not operate at line rate and hence do not meet the very low latency requirements typical of many and varied next-generation network functions.

The current availability of off-the-shelf programmable data planes, like Intel Tofino ASICs [1], and of domain-specific languages like P4 [2] brings new prospects for low-latency and high-throughput inference in networks. There have been multiple proposals to encode trained ML models directly into network devices like switches or Smart Network Interface Cards (SmartNICs). Embedding ML models into such devices is a challenging task, especially in the case of programmable switches, owing to their high constraints in terms of memory, support for mathematical operations, and the amount of allowed per-packet operations.

Previous works [3]–[5] have shown the potential of in-switch inference based on Decision Tree (DT) and Random Forest (RF) models. Yet, due to the constraints above, existing approaches fall short in at least three aspects. First, most solutions employ only stateless packet-level features and do not use stateful flow-level features such as inter-arrival times and flow counts that are essential for an effective inference in challenging use cases. Second, they support limited scalability in terms of ML model complexity and of the difficulty of the classification tasks that can be effectively handled. Lastly, many solutions are only tested in emulation environments and are unsuitable for real hardware deployment.

II. Proposed solution

`Flowrest` [6] addresses the shortcomings of previous solutions by proposing a practical framework that can run RF models at flow level in real-world programmable switches. It enables the embedding of large RF models into production-grade hardware, for challenging inference tasks on individual traffic flows and at line rate.

`Flowrest` is implemented as open-source software using the P4 language. It is designed for the Protocol Independent Switch Architecture (PISA) and is fully described in [6]. It enables a synergic integration of in-switch inference with the legacy forwarding functions of the switch. `Flowrest` effectively exploits flow-level metrics that are computed and stored in the switch to be used as features for complex inference tasks. In our solution, we take into account the constraints of the programmable switches and tailor the RF models' features and hyper-parameters from the design phase.

Packets arriving in the switch are parsed and header information is extracted. If the packets are already classified or do not have to be, they are forwarded normally. Otherwise, they go through flow management which comprises three phases that rely on a flow management table: (i) *flow tracking*, where CRC32 and CRC16 hashes of a 5-tuple of header information (source and destination IP addresses, source and destination ports, and protocol) are used to identify flows and retrieve their associated data; (ii) *flow-level feature handling*, in which per-flow features are read and updated via Tofino's `RegisterAction` extern; (iii) *early forwarding*, in which already classified flows use the classification result stored in the switch to get forwarded without going through inference.

After going through flow management, packets reach the inference module. Here, `Flowrest` employs a mapping scheme based on that proposed by Planter [5] to encode RF models into the switch pipeline. The scheme exploits range matches to compare the computed features against feature tables, one for each feature of the forest, and compose unique codewords that identify a specific leaf on a tree. The resulting codewords are then compared with ternary matches against codeword tables, one for each tree. A voting table then produces the final class as a majority vote of the different tree outputs.

III. Demonstration

The demonstration has two main goals. The first one is to showcase how `Flowrest` outperforms a benchmark solution based on packet-level inference in a challenging classification use case. We use the UNIBS-2009 dataset,

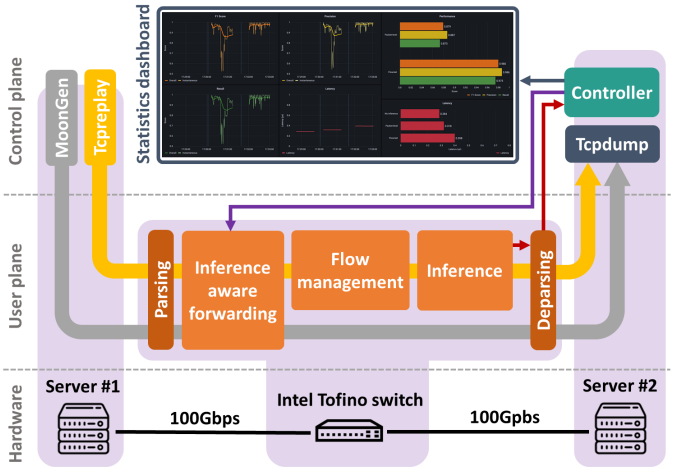

Figure 1: Demonstration workflow, and mapping of the logical components of the control and user planes into the testbed hardware.

capturing real-world traffic including peer-to-peer applications (BitTorrent, Edonkey), mail (POP3, IMAP4, SMTP), web traffic (HTTP/HTTPS), and other protocols (FTP, SSH). In a classification task that aims at identifying these 8 traffic categories, Flowrest improves the classification accuracy of the benchmark by 10%. The second goal of the demo is to prove experimentally that in-switch inference can be fully performed at line-rate, with minimum delay with respect to normal forwarding. To this end, we implement our demonstration in a real-world testbed with one Edgecore switch with an Intel Tofino BFN-T10-032Q chipset and two off-the-shelf servers with Intel 8-core Xeon processors at 2GHz and 48GB of RAM. The hardware is interconnected with QSFP28 interfaces, resulting in a full 100-Gbps platform, as shown at the bottom of Figure 1.

We perform three different evaluations. The first one aims at estimating the end-to-end latency when no inference is performed in the switch, running normal forwarding in the switch. In the second and third evaluations, we estimate the end-to-end latency and the classification performance of `Flowrest` and the benchmark solution respectively, in a challenging classification task. We estimate the classification performance with three different metrics: F1-score, precision, and recall. The evaluations are performed by running three experiments one after the other, for about 60 seconds each. During each iteration, we calculate and display in the dashboard the relevant metrics live. After each iteration, the results are kept to provide a comparison between the other evaluations.

The demonstration workflow builds on five logical components: (i) packet source ; (ii) programmable switch; (iii) sink for the traffic; (iv) controller that interacts with the switch, with the traffic sink and with the statistics dashboard; (v) dashboard that displays the statistics. These components are mapped into the testbed hardware as shown in Figure 1.

Traffic source. We implement the traffic source into a dedicated server. This component generates and sends traffic packets to the switch. The packets that are targeted by the inference, highlighted in yellow in Figure 1, are injected into the switch by replaying pcap traces via Tcpreplay. We also inject normal traffic with MoonGen packet generator.

Programmable switch. We use an Intel Tofino switch to forward traffic from the source to the sink. After parsing, the traffic is categorized as generic or target in the inference-aware forwarding block, as shown in Figure 1. While generic traffic is forwarded without any further action, the target packets are first identified as belonging to specific flows, then classified and finally forwarded according to their class, as described in detail in Section II. After each classification, a digest is sent to the controller with the class of the flow.

Traffic sink. The traffic forwarded by the switch is collected by a Tcpdump instance running on a second server. Once all the target packets have been received, we can analyze the resulting pcap trace in order to find the time of arrival of the packets. With this information, we can easily calculate the end-to-end latency, from the source to the sink, by subtracting the original timestamp of the packets.

Controller. We run a controller instance in the second server of the testbed. It is a very important element since it plays multiple roles. First of all, it configures the switch at each iteration by injecting the P4 code of the relevant evaluation to perform. Second, it analyzes the traces collected by the traffic sink in order to calculate the end-to-end latency. Third, it calculates the inference performance in terms of F1-score, precision, and recall by comparing the classification results with the ground truth. Finally, the controller interacts with the statistics dashboard and feeds data to it.

Statistics dashboard. We show the performance of `Flowrest` in a dashboard connected to the Controller. The dashboard is organized into two main areas, as shown in Figure 1. In the first one, we show live results while each evaluation is running. In the second one, we summarize and compare the results of each evaluation with each other.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 101017109 "DAEMON", which supported the work of M. Gucciardo, and the Marie Skłodowska-Curie grant agreement no. 860239 "BANYAN", which supported the work of A.T.-J. Akem. This work was also funded by the CHIST-ERA grant no. CHIST-ERA-20-SICT-001 "ECOMOME", via grant PCI2022-133013 of Agencia Estatal de Investigación, which supported the work of B. Bütün.

References

[1] Intel, "Tofino Switch." [Online]. Available: https://tinyurl.com/ycx79k4z
[2] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Computer Communication Review*, vol. 44, no. 3, jul 2014.
[3] Z. Xiong et al., "Do switches dream of machine learning? toward in-network classification," in *HotNets 2019*. NY, USA: ACM, 2019.
[4] C. Busse-Grawitz et al., "pForest: In-network inference with random forests," *CoRR*, vol. abs/1909.05680, 2019.
[5] C. Zheng et al., "Planter: Seeding trees within switches," in *SIGCOMM '21*. NY, USA: ACM, 2021.
[6] A.T-J. Akem et al., "Flowrest: Practical flow-level inference in programmable switches with random forests," in *IEEE INFOCOM 2023*.