Marina: Realizing ML-driven Real-time Network Traffic Monitoring at Terabit Scale

Michael Seufert, Katharina Dietz, Nikolas Wehner, Stefan Geißler, Joshua Schüler, Manuel Wolz, Andreas Hotho, Pedro Casas, Tobias Hoßfeld, Anja Feldmann

***Abstract*—Network operators require real-time traffic monitoring insights to provide high performance and security to their customers. It has been shown that artificial intelligence and machine learning (ML) can improve the visibility of telemetry systems, especially with encrypted traffic. However, current solutions cannot cope with high traffic rates and volumes in large-scale networks. To realize the ML-driven network intelligence paradigm at terabit scale, we design *Marina*, a system that spreads monitoring over a highly efficient data plane, which can extract traffic statistics at line rate, and a powerful ML server, which can run monitoring inference using complex ML models. We apply temporal microaggregation into sub-second time slots and extract moment-based statistics. These allow to flexibly obtain accurate ML-based monitoring decisions during the next time slot. To demonstrate the scalability of our design, we implement and evaluate a *Marina* data plane prototype on a Barefoot Wedge 100BF-65X P4 switch, which can monitor more than 520,000 concurrent flows at full switching capacity of 6.4 Tbps. We validate the analytics capabilities enabled by our *Marina* implementation for four ML-driven real-time monitoring tasks with a broad set of standard ML models, achieving comparable or better than state-of-the-art results.**

***Index Terms*—Network monitoring, artificial intelligence, machine learning, encrypted traffic, real-time monitoring, P4, programmable data plane.**

I. Introduction

The growing number of users, devices, and applications, as well as the increasing complexity of networks, push network operators to deploy broader and more efficient network monitoring solutions to improve their visibility, to quickly detect and resolve performance or security issues, as well as to optimize resources. While flow-level network telemetry approaches (e.g., NetFlow [1], sFlow [2], IPFIX [3]) provide valuable insights on how a network is operating, they are limited with respect to the monitoring capacity (amount of traffic/flows), expressiveness (set of monitored statistics), and accuracy (sampling). In addition, their coarse temporal granularity (typically, export intervals are 1 minute or higher) does not align well with real-time monitoring tasks. Thus, packet-level monitoring capabilities and small temporal granularity are desired for deriving actionable management decisions in real-time.

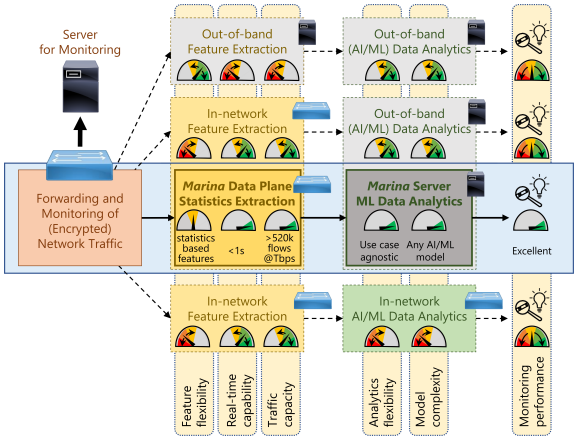

Figure 1: Design principles for the deployment of ML-based network telemetry systems.

Traditionally, deep packet inspection (DPI) has been used on the full packet stream to identify applications or threats in the network, and even to obtain application-layer information regarding the health of applications such as voice or video streaming. However, the increasing network and application complexity and the wide adoption of end-to-end traffic encryption are drastically limiting the visibility of operators on the performance of services consumed by their customers. This has given rise to a wider adoption of artificial intelligence and machine learning (ML) technology to improve traffic monitoring at scale. These approaches have been successfully applied to flow-level data [4], [5], [6] and packet-level (time series) data [7], [8], [9], even for the case of encrypted network traffic [10], [11], [12], [13]. Still, running ML-driven monitoring applications in real-time and at line rate is challenging. The combination of fine-grained, per-flow and per-session monitoring requirements with the high volume of data observed in modern wide area networks (WANs), data center networks (DCNs), or enterprise networks impose difficult-to-meet targets, and the performance of deployable solutions often falls short in large-scale networks.

In general terms, a typical ML-driven network traffic monitoring workflow consists of three consecutive steps, as depicted in Figure 1. It starts by analyzing the *(encrypted) network traffic*, followed by a *feature extraction* of this data into a vector representation, which serves as input to the *ML data analytics model*. Different design principles are applied

M. Seufert was with University of Würzburg, Würzburg, Germany for this work, but is now with University of Augsburg, Augsburg, Germany. K. Dietz, N. Wehner, S. Geißler, M. Wolz, A. Hotho, and T. Hoßfeld are with University of Würzburg, Würzburg, Germany. J. Schüler was with University of Würzburg, Würzburg, Germany for this work, but is now with Tesat-Spacecom GmbH & Co. KG, Backnang, Germany. P. Casas is with AIT Austrian Institute of Technology, Vienna, Austria. A. Feldmann is with Max Planck Institute for Informatics, Saarbrücken, Germany.

Manuscript received May 26, 2023; revised December 18, 2023.

in the literature to integrate this ML workflow with traditional network telemetry approaches. *Offline or out-of-band processing* (top part of Figure 1) is typically the starting point for model development and academic endeavors (e.g., [14], [15]). For this, the monitored traffic has to be captured or mirrored, and forwarded on an external server. Stream processors are software-based and provide high flexibility in terms of features and analytics. They can reach excellent monitoring performance given sufficient compute resources and time for computing features, as well as training and applying complex models. However, limited traffic processing capacities and absence of real-time monitoring capabilities typically make this approach infeasible for deployment in operational networks.

Many current network telemetry approaches leverage novel capabilities in softwarized and virtualized networks (SDN/NFV), using sophisticated software packet processing technologies [16] and programmable data planes (PDP) [17], [18]. They have opened the door for the conception of more flexible, real-time monitoring capabilities using programmable switches [19], [20] or commodity hardware [21]. These systems can analyze the massive traffic volumes in real-time, i.e., at line rate without impacting packet forwarding, and thus, network performance. At the same time, they can be used to implement *in-network feature extraction* to reduce the amount of data forwarded to an out-of-band stream processor (middle part of Figure 1). By deploying a powerful server, the data analysis can be scaled to support complex models in real-time, and can, in theory, leverage the full analytics flexibility of software-based stream processors. In practice, however, this flexibility in terms of number and types of realizable monitoring tasks is limited or dictated by the extracted feature set. Here, hardware and processing constraints of the network device and potential service disruption during reprogramming of the device significantly reduce the feature flexibility. Therefore, the in-network feature extraction system has a pivotal role.

In-network ML (bottom part of Figure 1) integrates the ML model directly into the data plane, either offloading feature extraction to an external system [22], [23], or by integrating both the feature extraction process and the ML model [24], [25], [26], in which case the monitoring system might provide high capacities and real-time capabilities. However, in-network ML requires heavy tailoring and simplification of the specific ML model, given the limited operations supported by high-speed programmable hardware, losing flexibility in terms of feature and ML analytics capabilities.

To effectively manage large-scale traffic while offering rich analytics capabilities, our focus is on efficient in-network extraction of monitoring information and out-of-band processing on a powerful server, as shown in the middle part of Figure 1. However, in contrast to existing network telemetry systems, we design our system ***Marina*** **(MAchine-learning-based Real-time Network traffic Analytics)** explicitly for ML-based real-time network traffic monitoring at terabit scale. The rationale of *Marina* is to devote and max out the limited data plane resources to extract statistics, which are useful for ML-based traffic analytics, even in case of encrypted traffic. All subsequent and more complex workflow tasks, i.e., feature generation from the extracted statistics and model inference, are executed on a powerful ML server. This is possible, as ML models can be trained to leverage the generated feature sets to achieve excellent performance for different real-time traffic monitoring tasks. This includes traffic/device classification, application health (e.g., Quality of Experience), and fault/anomaly detection (e.g., intrusion detection), which we demonstrate in this work.

We distinguish from existing solutions, which often apply techniques like sampling (e.g., [1], [2], [27], [28]), sketching (e.g., [29], [30], [31]), or filtering/querying (e.g., [32], [19], [20], [21], [33]), and do not explicitly consider the ML-based network intelligence paradigm. Instead, *Marina* implements a temporal microaggregation of packets using sub-second time slots and extracts moment-based traffic statistics. This allows a fine-grained tracing of traffic characteristics for each monitored flow at sub-second granularity. These statistics can subsequently be mapped into a high-dimensional feature space, which offers high visibility and discriminative power to the downstream analytics and provides high flexibility for applying arbitrarily complex ML models for different monitoring tasks. By controlling placement and compute resources of the ML server, we can obtain actionable results for all monitored flows within the next time slot, i.e., with sub-second delay.

The contributions of this paper are as follows:

① ***Marina*** **system design (§2)**: we present a novel concept for data plane extraction of moment-based statistics at line rate in combination with ML-based analytics on a powerful server. Our design relies on temporal microaggregation of packets into sub-second time slots and allows to flexibly realize different ML-driven real-time monitoring tasks with high accuracy at scale, even for encrypted traffic.

② ***Marina*** **data plane implementation (§3) and evaluation (§4)**: our implementation of the *Marina* data plane on a Barefoot Wedge 100BF-65X P4 switch maxes out the data plane resources to monitor up to $6.4\,\mathrm{Tbps}$ of traffic in 524,288 concurrent flows over 65 QSFP $100\,\mathrm{Gbps}$ ports. It generates less than $385\,\mathrm{Mbps}$ monitoring traffic, and can keep monitoring granularity and delay until obtaining monitoring results for all flows as low as $500\,\mathrm{ms}$. For the sake of reproducibility and as an additional contribution, we make *Marina*'s code publicly available at: https://github.com/lsinfo3/Marina

③ ***Marina*** **ML-based real-time traffic monitoring (§5)**: we validate the analysis capabilities enabled by *Marina* for four different use cases, namely, encrypted traffic classification, video streaming application health/Quality of Experience, intrusion detection, and IoT device classification. We achieve comparable or better than state-of-the-art results with standard ML models.

II. *Marina* System Design

Monitoring large networks with high traffic volumes and rates, such as ISP, data center, or enterprise networks, faces well known challenges (e.g., [34], [35], [33]). In particular, our system design addresses the following challenges to enable ML-driven real-time traffic monitoring at terabit scale.

Scalability: the monitoring system must be able to cope with high traffic volumes in a large number of flows within its hardware resources. Extracting the monitored information from packets must happen at line rate, while ensuring that the monitoring does not negatively impact the basic forwarding mechanism, and thus, the network performance or stability.

Overhead: the telemetry system must keep the monitoring overhead as low as possible. Resource allocation of the system should be constant, i.e., constant memory per flow, and independent of the traffic load. Traffic generated by the monitoring system must not overload the management network or any out-of-band consumer of the monitored data, such as compute or storage servers.

Expressiveness: the monitoring system must provide sufficient compute resources to support a wide range of performance- and security-related telemetry tasks with high accuracy. It must be able to provide monitoring results on different time scales like real-time insights, records of terminated flows, or periodic aggregate reports. Moreover, it must allow to consider relationships between flows and provide monitoring results for sets of related flows, e.g., a browser session using several connections. As traffic encryption hides important information contained in inner protocol header fields and payloads, the monitoring system must make the best use of information that is always available (i.e., information which cannot be hidden by encryption). Consequently, monitoring must be independent of traffic encryption by design, providing the same accurate insights if the traffic is encrypted or not. Nevertheless, aiming to overcome the limited visibility due to encryption, it must allow to obtain valuable monitoring results for many different monitoring tasks.

Flexibility: network operators require network telemetry to be flexible, such that they can add or change monitoring tasks at any time without affecting the network. They need to be able to deploy *Marina* at different vantage points and combine the monitoring results to obtain network-wide insights. Moreover, it must be possible to store monitored information, e.g., to revisit or analyze the historical state of the network, to detect changes, or to forecast trends.

Trade-offs: while scalability and extracted monitoring data are limited by the data plane capabilities, there is a trade-off between the number of monitored flows and the generated traffic and load at the server. Additionally, the number and complexity of generated features and executed ML models impact the processing time at the server, depending on the server's compute resources. This results in a trade-off with the monitoring granularity and real-time capabilities of our system. We investigate these trade-offs in §4.

A. Design Principles

We design our system *Marina* for real-time traffic monitoring at terabit scale by implementing three design principles:
① The data plane must do the heavy lifting work and carry most of the monitoring burden. Its task is to reduce the data volume as early as possible, but at the same time extract valuable information. To be able to observe Tbps traffic, our system is designed to be deployed at a core network element or gateway where many high speed links interconnect. Data planes are built to forward high volumes of traffic, but offer limited computational or storage resources. Nevertheless, specialized devices (e.g., programmable switches, FPGAs, or ASICs) can execute packet operations at line rate, such as arithmetic calculations, without affecting the forwarding and the network performance. We require such a specialized device for *Marina* and install a data plane program to efficiently forward and monitor at line rate for all ports. We allocate constant memory to each flow, aiming to max out the data plane resources to monitor as many flows in parallel as possible. The controller should be located on the same device for efficient communication with the data plane. It instructs the data plane which flows to monitor and where to forward packets by installing appropriate flow rules, and exports the monitored data.
② We move all complex operations to a powerful server. It can efficiently take over the more sophisticated work on the reduced data, namely, generating feature sets and running model inference for a large number of monitored flows and ML-based monitoring models. The server can be equipped with appropriate and specialized compute and storage resources, which allows to further speedup tasks on many CPUs or GPUs in parallel. To avoid overloading the server, its service rate (processing of transmitted data and ML inference) has to be higher than the arrival rate (export of monitored data from the data plane). Controlling server placement, server resources, and complexity of features and models allows to enforce an upper bound on the service time, and thus, makes real-time monitoring possible. A single server can serve one or more data planes, or multiple servers can be combined into a server cluster. This allows to merge data from multiple vantage points and obtain network-wide insights. Finally, the server (cluster) allows to offer a rich API for network operators to inspect and visualize the stored data and monitoring results, as well as to flexibly change or add monitoring tasks. As the monitoring flexibility is on the controller or server side, we avoid having to restart the data plane device to change the data plane program, which would result in network downtime. The monitoring results can then be sent to, e.g., a network management system. Note that the development of a server API and the selection of traffic engineering decisions based on the inferred results are beyond the scope of this work.
③ The key to the success of our system is how to link the first two design principles. In particular, we need to extract a small amount of valuable information, which allows to realize the monitoring tasks with high accuracy. To achieve this, we perform temporal microaggregation on the data plane. We temporally divide the packets per flow into fixed-length time slots. We use statistical descriptors to characterize the traffic within each time slot in constant memory. We read out and reset the descriptors after each time slot, and export them to the server. The stream of descriptors provides a fine-granular approximation of the observed traffic, which allows the ML models at the server to accurately infer the monitoring metrics. However, it is important to configure the time slot duration appropriately. As discussed above, to avoid overloading the

server and allow for real-time monitoring, the time slot duration has to be higher than the end-to-end monitoring delay consisting of data plane register read, transmission between controller and server, feature generation, and ML inference. Additionally, the time slot duration not only defines the real-time capability of the entire monitoring system, but it also impacts the temporal granularity of the stream of descriptors, and consequently, the accuracy of the monitoring. Thus, we minimize the slot length to below $1\,\mathrm{s}$.

B. Monitoring of Encrypted Traffic

As end-to-end encryption of Internet applications is becoming the norm, enabled by protocols such as transport layer security (TLS), we have no longer access to application-layer information. This renders DPI ineffective without requiring additional privacy-invasive methods (e.g., TLS interception), and thus, substantially limits the visibility of many network telemetry systems. Traffic encryption allows to only extract information from the IP and transport protocol (TCP/UDP) headers. Available information includes the flow 5-tuple (src/dst IP addresses and port numbers plus protocol type), packet size, TCP flags, or header options. In addition, a network element can track the inter-arrival time (IAT) of packets, i.e., the time since the last packet of the same flow has been forwarded. Despite this limited information, we can still derive temporal and volumetric traffic information on a per-flow basis, i.e., how much traffic is transmitted over time. For this, it is sufficient to inspect the 5-tuple and track packet sizes and IATs. This information is highly valuable for performance- and security-related network monitoring, as it allows, for example, to identify characteristic device/application traffic patterns or to detect when device/application traffic patterns deviate from normal.

Since extracting and storing full time series of packet sizes and IATs is infeasible at terabit scale, we employ temporal microaggregation as discussed above and characterize their distributions within each time slot using statistical descriptors. We decide to use moments, i.e., sample moments about the origin, as they provide valuable insights and can be computed on the data plane in constant memory using only simple arithmetic operations. To compute the k-th sample moment, it is required to raise the observed values to the power of k, sum the resulting powers, and divide by the total number of packets. Note that this computation naturally allows to obtain packet count and traffic volume. These raw moments may then be converted into central moments and standardized moments, allowing to compute the most important named properties of the distribution (mean, variance, standard deviation, coefficient of variation, skewness). It also allows to accurately approximate the observed distributions (cf. truncated Hausdorff moment problem [36]).

While higher-order moments are computationally expensive, especially when considering the limited resources on the data plane, we advocate for computing at least the first three moments. The reason is that skewness (third standardized moment) describes where the distribution mass is concentrated, e.g., towards small/large packets or towards bursty/isolated packets. Skewness plays an important role in many monitoring scenarios, also being one of the reasons why sketching emerged decades ago (cf. AMS sketches [37]). The suitability and generality of considering sample moments and derived features is also confirmed when looking at related work, where they are widely adopted for a multitude of use cases, such as real-time anomaly/intrusion detection [38], [39], [40], traffic classification [41], QoE inference [12], [10], or IoT device fingerprinting [42], [43], [15].

We are not limited to use only the meta-information (timestamp, src/dst IP address and port number, protocol type) and statistics (moments and derived features) of a single time slot and flow as input for the ML-based monitoring prediction. Instead, we can perform derivations/augmentations (e.g., infer IP ranges, domain name, or service type from the meta-information in the 5-tuple) and aggregations at the server side. Considering the temporal dimension, for example, we can aggregate the statistics of consecutive time slots (e.g., in a sliding window fashion [12]) to cover larger time spans. For this, the corresponding moments have to be multiplied with their packet counter to obtain summed powers again, which can then be added together. Moreover, we can consider a time series of statistics from consecutive time slots as input to sequential ML models, such as popular recurrent neural networks (RNNs). Besides the temporal dimensions, aggregations can also be performed in the spatial dimension. For example, the summation statistics of a matching pair of unidirectional flows can be added or combined to derive features for bidirectional traffic, e.g., ratio of uplink/downlink packets or volume, or total traffic volume. It is also possible to merge or concatenate feature sets of a larger set of related flows, such as multiple TCP connections of a single application having the same source IP address (e.g., using hierarchical embeddings [44]). Similarly, feature sets from different vantage points, e.g., considering the same flows or the same types of service, can be merged or concatenated to infer network-wide insights. In short, as all data is gathered on the server, we can process the collected meta-information and statistics as well as derived and augmented information from single/multiple time slots, single/multiple flows, and/or single/multiple vantage points on the server as needed in order to obtain accurate monitoring results for our desired monitoring tasks.

Depending on the generated features, many ML models can be applied to realize the monitoring task. This ranges from shallow ML (e.g., decision trees) to elaborate methods such as deep learning (DL), or (deep) reinforcement learning (RL), which can provide accurate traffic insights. The limitations to feature generation and model inference are given by the expressiveness of the extracted statistics and the time consumption for processing on the ML server, which adds to the end-to-end monitoring delay, and thus, affects time slot length.

III. *Marina* Data Plane Prototype

Figure 2 gives an overview of the *Marina* implementation. To fulfill the hardware requirements of the *Marina* data plane, we implement the prototype on a Barefoot Wedge 100BF-65X P4-enabled Tofino switch. As current P4 hardware

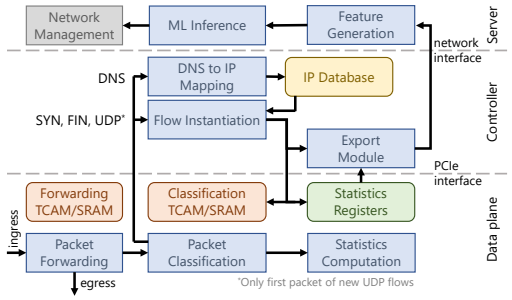

Figure 2: Implementation of *Marina* prototype.

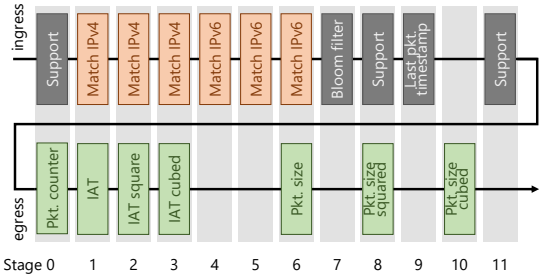

Figure 3: Data plane stages: register/table allocation. Support registers are shown in gray.

exhibits several limitations in terms of stateful processing, memory capacity, and the complexity of available operations, we highlight workarounds and approximations that went into developing the prototype implementation.

Packet Forwarding: the switch performs statistics extraction in addition to (a) bridging the traffic, (b) L2 switching, or (c) L3 routing. To do this, the Packet Forwarding component (depicted at the bottom left) uses the Forwarding TCAM (depicted above the packet forwarding component) or metadata information attached to the ingress ports to identify the corresponding egress port. The available TCAM memory limits the number of forwarding flow rules to roughly 1.5 million flow rules. If the switch is used as a bridge (man-in-the-middle device), the number of forwarding rules is of no concern, but the number of available ports is halved. Note that the installation of forwarding rules is not part of the P4 controller developed in this work.

Packet Classification and Flow Instantiation: after identification of the egress ports, packets pass through the Packet Classification module, which uses the Classification TCAM to identify flows requiring further processing. For this, we reuse concepts known from reactive OpenFlow [45] applications. We maintain the set of relevant flows to be monitored in the Classification TCAM. If a packet is not matched, it either belongs to an irrelevant flow and can be ignored, or is part of a new flow. In the latter case, it needs to be sent to the controller. Our design is not impacted by the well known problems of reactive flow processing [46], [47], [48], as the controller application runs on the switch's host controller, which communicates with the data plane over the internal PCIe interface, offering minimal delay and a throughput of up to 31 Gbps (PCIe Gen3 x4). For TCP, new flows can be easily detected based on TCP SYN flags. For UDP, we need to track which flows have been already seen. We employ a combination of a partitioned Bloom filter [49], [50] in the data plane and a counting Bloom filter [51] at the controller. They ensure that we can efficiently identify previously seen, but irrelevant UDP flows with an acceptably small false positive rate of below 1%, when packets of 400,000 irrelevant UDP flows are present. This is expected, due to the probabilistic nature of the applied Bloom filter. A false positive hit results in missed relevant flows that are falsely classified as irrelevant by the Bloom filter. Finally, all TCP SYN packets and the first packets of unknown UDP flows are forwarded to the controller.

For each new flow, the controller checks whether it is relevant for monitoring. If so, it assigns a flow id, inserts the corresponding rules into the data plane Classification TCAM, and allocates register slots for the statistic computation. Additionally, it stores the mapping between 5-tuple and register slots, such that the 5-tuple can be exported together with the corresponding statistics as meta-information. Likewise, flow ids and allocated register slots are freed when a flow is considered ended. The removal of TCP flows is straightforward, based on the tracking of packets with the TCP FIN flag or a flow timeout. To remove a UDP flow from the counting Bloom filter, we use a probabilistic aging technique [52]. It decrements the corresponding entries of the counting Bloom filter with an eviction probability that controls the retention time. We configured it to obtain an average flow retention time of 25 s. Any changes in the counting Bloom filter are mirrored to the binary Bloom filter on the data plane.

The controller decides on the relevance of flows in the Flow Instantiation module, e.g., based on flow 5-tuple or IP range. Additionally, we might be interested to selectively monitor the traffic of a certain application, e.g., to monitor application health. To identify which flows may belong to these applications, we rely on the hostnames of the contacted servers, which we obtain from parsing DNS requests [53]. Therefore, all DNS responses are forwarded to the controller's DNS to IP Mapping module, where an IP Database with relevant IP addresses is constructed. For example, to monitor YouTube, the database is filled with all IP addresses for *googlevideo.com* – the domain used by YouTube's video chunk HTTP requests. In case of DNS over TLS (DoT) or DNS over HTTPS (DoH), where we no longer can leverage the clear text from DNS requests and responses to differentiate between flows, we use the Server Name Indication (SNI) field of TLS. For this purpose, we forward TLS Client Hello packets to the controller's SNI to IP mapping module. As a result, we obtain a similar database as for DNS. For example for YouTube with DoT or DoH enabled, we therefore look for TLS Client Hello packets having a SNI, which contain *googlevideo.com*. As some applications like video streaming use several flows in parallel, we additionally implement a session mode, which assigns all flows with the same source IP to the same registers. This allows to monitor the aggregated application traffic, which saves data plane resources and facilitates downstream analytics. The controller can seamlessly switch between flow and session mode during runtime.

Statistic Computation: following our design principle, we implement the calculation of sample moments for the distribution of packet size and packet inter-arrival time, while maxing out the data plane resources. Figure 3 shows the allocation of

registers and matching tables on the data plane stages. The packet first traverses the ingress pipeline (top), is switched to the correct egress port, and finally traverses the corresponding egress pipeline (bottom). The statistic computation is limited by the number of stages of the switch – in total 24, 12 on ingress and 12 on egress – and by the computation capabilities of the ALUs at each stage. Resources are further limited by the stages required for Packet Forwarding and Packet Classification. Thus, we are able to realize only the first three moments of the packet size and IAT distribution on our P4 switch. More specifically, as detailed in Table I, we count the number of packets and compute (for both distributions) the sums of (a) the raw values, the (b) square of the values, and (c) the cubes of the values. We also track the timestamp of the last arrival as a support statistic for the computation of the inter-arrival times.

Here again, the hardware limitations of the switch forced us to resort to certain abstractions for the computation process. Consider computing the statistical moments of the packet size distribution, i.e., the summed powers of the packet size. The packet size can be easily computed by subtracting various header lengths from the total packet size. However, recording the sum of packet sizes using byte granularity in a $32\,\mathrm{bit}$ register is likely to result in overflows, e.g., a flow sending at $100\,\mathrm{Gbps}$ would overflow the register more than three times per second. Moreover, the sum of squared packet sizes, which allows to derive the variance, would overflow after just around 2000 packets of size $1500\,\mathrm{B}$. Additionally, as the data plane supports only $32\,\mathrm{bit}$ addition and subtraction, we also need to approximate multiplications by pre-computing TCAM rules as shown in [54]. Although the P4 compiler supports $64\,\mathrm{bit}$ registers, we cannot perform arithmetic operations on them. Implementing $64\,\mathrm{bit}$ arithmetic would be possible using multiple registers with intermediate overflow detection. However, we decided for a simpler approach, based on the assumption that the magnitude of packet sizes and IATs could still provide valuable information.

This was realized by using the logarithm of the packet sizes and IATs instead of the actual value. This way, a $32\,\mathrm{bit}$ register is sufficient to record the summed powers of logarithmic values. As the logarithm is not natively supported by the ALUs of the switch, it is computed using a ternary match table in TCAM, constructed as described in [54], mapping input values using a longest prefix match on the binary representation to their approximate logarithm. Hence, the implementation column in Table I denotes the approximated logarithm as log^*. Note that register overflows can still occur on long-running or large volume flows for these statistics. We investigate the impact of these approximations on the performance of the monitoring tasks in §5.

Monitored Flows: The number of concurrent flows that can be monitored is effectively limited by the memory capacity of the data plane and the selected statistics. The Barefoot Wedge 100BF-65X has 4 parallel pipelines with 12 stages. Each stage contains 80 blocks of SRAM with $128\,\mathrm{kbit}$ each. 48 of those blocks, i.e., $6\,\mathrm{Mbit}$, are available as stateful memory. A register always occupies whole blocks, at most 35 blocks, and requires one additional block for organizational purposes. If the selected statistics are $32\,\mathrm{bit}$ values, the maximum number of slots in a register is $35 \cdot \frac{128 \cdot 1024}{32} = 143,360$. This is then the theoretical maximum number of flows that can be monitored in a single pipeline. However, there is a trade-off between the number of monitored flows and the number of selected statistics.

To simplify register addressing we use a power of two, resulting in a flow capacity of $2^{17} = 131,072$ flows per data plane pipeline, thus, $4 \cdot 2^{17} = 524,288$ unidirectional flows at most, as the switch has four independent data plane pipelines. In session mode, we summarize all uplink/downlink flows together by recording separate values for the aggregated uplink/downlink traffic only. This allows us to monitor $2^{18} = 262,144$ bidirectional sessions in parallel. These numbers are derived using the memory capacities of the Intel Tofino 1 chipset. Next generation devices are expected to be equipped with more memory for both SRAM and TCAM, and could hence monitor more flows or compute additional statistics.

Export Module: the controller reads all data plane statistics registers at regular time intervals, appends the corresponding meta-information (5-tuple in flow mode, 3-tuple in session mode), and transmits them via the $1\,\mathrm{Gbps}$ management interface of the switch to an external ML server. This results in $328\,\mathrm{bit}$ generated monitoring data per flow per time slot. Our first approach of using the Apache Thrift interface provided by the Tofino SDE failed, as it needed on average $3\,\mathrm{s}$ for a complete read of all registers. Thus, we implemented a custom controller library with access to internal functions based on the suggestions of Yu et al. [55]. This allows us to circumvent the Thrift layer and achieve a much smaller read duration of $268\,\mathrm{ms}$, independently of the number of monitored flows. Nevertheless, the number of monitored flows affects the transmitted data volume, and thus, the minimal end-to-end monitoring delay. As the end-to-end monitoring delay has to be smaller than the time slot length for real-time monitoring, there is a trade-off on how small the time slot can be set, which we analyze in §IV.

Feature Generation and ML Inference: the server receives the extracted statistics and meta-information collected from the data plane and generates feature sets as input to ML models. Depending on the monitoring task, different features can be derived as discussed in §II-B and depicted in Table I. The computed features are assembled into feature sets and forwarded to ML models to infer the monitoring predictions. In this work, we adopt the majority of features, which have also been computed in [12], because they cover the most important characteristics of the packet stream and can be computed in an online fashion [12]. Note that, although there is basically no limit to what and how many features and models can be used, the feature generation and model inference times impact the end-to-end monitoring delay. To preserve the real-time properties of the entire monitoring system, the server enforces a fixed upper bound on the end-to-end monitoring delay to be less than the time slot length, which can be easily achieved by controlling server placement, compute resources, feature generation, and model complexity. Using a powerful

Statistic	Implementation	Derivable Features
Packet count	Packet count	Packet count, packet ratio
Last packet timestamp	Last packet timestamp	*Packet inter-arrival time (IAT)* $\rightarrow$ *used only internally*
Sum of inter-arrival time (IAT)	$\sum \log^*(\text{IAT})$	Mean of IAT
Sum of squared IAT	$\sum \log^*(\text{IAT}^2)$	Variance, standard deviation, coefficient of variation of IAT
Sum of cubed IAT	$\sum \log^*(\text{IAT}^3)$	Skewness of IAT
Sum of packet size (PS)	$\sum \log^*(\text{PS})$	Volume, volume ratio, mean of PS
Sum of squared PS	$\sum \log^*(\text{PS}^2)$	Variance, standard deviation, coefficient of variation of PS
Sum of cubed PS	$\sum \log^*(\text{PS}^3)$	Skewness of PS
Meta-information	Flow 5-tuple (3-tuple)	Src/dst IP addresses, (Src/dst port numbers), protocol type

Table I: Statistics and meta-information that can be computed at line rate by *Marina* on a Barefoot Wedge 100BF-65X. $\log^*$ denotes the approximation of the logarithm based on lookup tables.

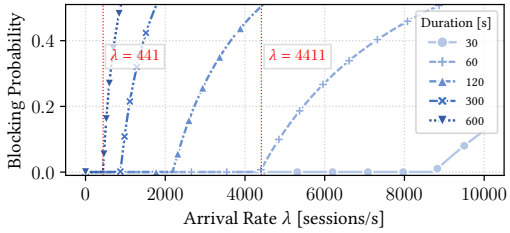

Figure 4: Blocking probability for different mean session durations for increasing session arrival rates.

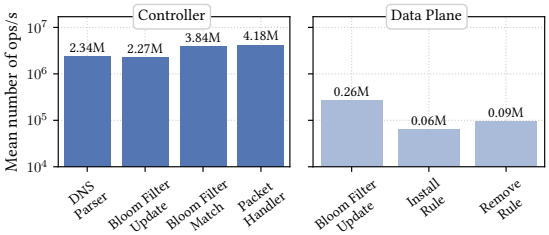

Figure 5: Mean operations per second for critical controller and data plane operations.

server also ensures that the resulting ML predictions for all sessions can be forwarded in real-time, e.g., to a network management system. In addition, the server can offer an API for network operators to inspect and visualize the stored data and monitoring results, as well as to flexibly change or add monitoring tasks at any time. This would instruct the *Marina* controller to change which flows are classified as relevant, or it would deploy another ML model on the server.

IV. Performance of *Marina* System

To showcase the real-world performance of *Marina*, we first highlight the isolated performance of all involved components and subsequently demonstrate the total end-to-end monitoring delay from collecting data plane statistics to obtaining ML predictions. As described above, we deploy the *Marina* data plane on a Barefoot Wedge 100BF-65X P4-enabled Tofino switch with 65 QSFP 100 Gbps ports for a total data rate of 6.4 Tbps. The controller application runs on the switch's host controller – an 8-core Intel Xenon CPU with 32 GB of memory running Ubuntu 18.04. Note that we do not validate the total data rate of the switch through dedicated measurements as it is a technical specification of the device. Instead, we focus only on the performance of the controller and data plane applications as well as the ML pipeline, as their operations will ultimately limit the performance of the entire Marina system to accurately monitor all relevant flows. Thus, for the evaluations in our testbed, the switch was connected to two servers – each equipped with 10-core Intel Xenon CPUs, a Mellanox ConnectX-5 series NIC offering two 100Gbps ports, 200 GB of memory, and running Ubuntu 18.04. The *Marina* ML server is equipped with a 64 core Xenon CPU, 8 GPUs (RTX 2080Ti 11GB), and 768 GB RAM and is connected via the 1 Gbps network management interface of the switch.

Flow Arrival Rate: the P4 switch has enough memory for *Marina* to support up to 524,288 unidirectional flows (or 262,144 bidirectional sessions) in parallel. We explore the impact of this limitation by assessing the probability to drop incoming sessions when the system is in a steady state, given a certain arrival rate of new, relevant sessions, for an average flow duration ranging from 30 s to 600 s, see Figure 4.

We assume that the arrival process of new flows is a superposition of multiple independent renewal processes that can be modeled as a Poisson process with a total arrival rate λ, according to the Palm-Khintchine theorem [56], [57]. Using λ and the amount of available memory, we compute the blocking probability of the system using the Erlang-B formula [56]. Thereby, we assume that the arrival rate to the four individual pipelines of the switch is equally distributed.

For a mean session duration of 60/600 seconds, *Marina* can handle 4411/441 new sessions per second while ensuring a blocking probability below 1%. To put these results into perspective, we use the data obtained in [58] on YouTube video streaming characteristics, and assume a session duration of 600 seconds. This translates into approximately 14.2 million users that can be handled by a single P4 data plane device assuming an average request rate, and 3.5 million users when considering the peak request rate reported in [58].

Controller Operations: to ensure real-time capabilities of the control plane, we explore each involved operation in isolation. To achieve this, we conducted stress tests on both the P4 switch control plane API and the controller. We employed custom benchmark scripts written in C to execute the same type of operation repeatedly in a loop. This allows us to identify the maximum number of operations of a specific type that can be supported per second, as shown in Figure 5. Controller operations are evaluated on the left side – meaning operations that do not involve the data plane (cf. Figure 2). The DNS to IP Mapping component is able to parse $2.3 \cdot 10^6$ DNS responses per second. The Bloom filter used for the Flow Instantiation component is able to handle $2.2 \cdot 10^6$ operations per second. The packet handler of *Marina*, which is responsible for processing incoming packets of new flows,

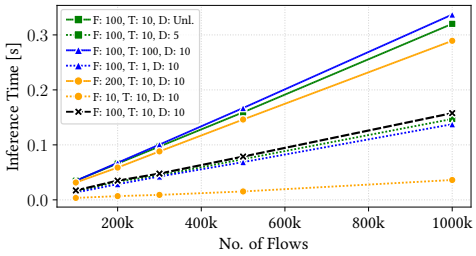

Figure 6: Random forest inference time on single GPU.

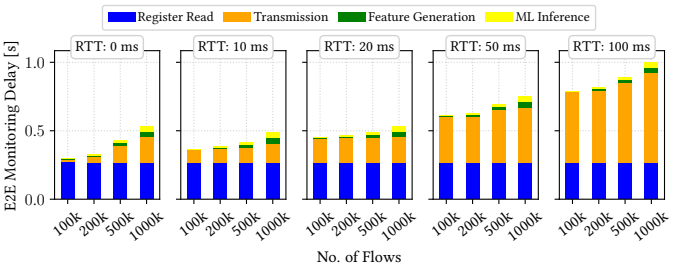

Figure 7: Total end-to-end monitoring delay from data plane statistics extraction to ML inference.

is able to handle almost $4.2 \cdot 10^6$ packets per second, showing that none of the benchmarked operations form a bottleneck for the system.

Data/Control Plane Interactions: regarding interactions with the data plane – Figure 5 on the right – the controller is able to perform around 260,000 Bloom filter write operations per second. Moreover, it can execute up to 63,000 match table insertions and 94,000 remove operations per second. Thus, 63,000 match table insertions per second are the bottleneck of the data plane API. Similarly, we evaluate the maximum rate at which flow modifications can be performed at runtime to determine the maximum burst arrival rate of new flows. For this, we used a sequence of TCP SYN and FIN packets, which forced the continuous creation and removal of flows on the data plane, and sent it over the switch at increasing rates. Our benchmarks show that the controller is able to process 50,000 flow changes per second. This means that the total set of 524,288 flows that can be monitored by the switch in parallel can be replaced roughly every 20 seconds.

ML Inference Speed: the inference speed depends on the number, type, and complexity of used ML models as well as the compute resources of the server. The specific model, which provides the highest accuracy, depends on the actual monitoring task at hand, cf. results in §5. As random forest models proved to provide a good trade-off between high accuracy and fast inference speed, we evaluate the time to infer predictions for a specific number of flows on a single GPU depending on the model complexity. To be able to run the trained scikit-learn model on GPU, we use Microsoft's Python library Hummingbird[1], which converts the Random Forest model to a PyTorch model under the hood. For the analysis, we vary the number of input features (F), number of trees (T), and tree depth (D) compared to a baseline. The results in Figure 6 show that, even when using a large set of 100 features and an expensive model with 100 trees of depth 10, we can obtain monitoring predictions for one million flows in less than $350\,\mathrm{ms}$ on just one GPU. In the following, we assume the performance of this model to perform a worst-case analysis of the end-to-end monitoring delay.

End-to-End Monitoring Delay: the total time to read the statistics from the data plane, to transmit them to the ML server, to generate features, and finally, obtain the ML predictions is shown in Figure 7. Note again that this end-to-end monitoring delay defines the real-time capabilities of *Marina* and provides a lower bound on the monitoring time slot duration at the data plane. Thus, it is the most important system parameter, which we need to minimize to improve the monitoring granularity and accuracy of *Marina*. The register read operation is performed by the controller through an API call to the data plane and is independent of the number of flows present in the system, as every call polls all registers that have been defined at compile time. Naturally, the time to transmit the polled statistics to the ML server scales with the number of available flows, shown along the x-axis. We include measurement values for one million flows to put the scalability of our approach into perspective, even if current hardware only supports 524,288 flows. Note that the register read duration will actually be different when supporting one million flows on a different hardware, which we do not reflect in the figure for comparison purposes. In addition to the register read, we investigate the transmission duration depending on the RTT between the controller and the ML server to assess the need for optimal server placement. Finally, the ML server augments the raw statistics by computing a set of 100 derived features and applies the pretrained random forest model. Here, we use all available resources of our ML server, i.e., 64 cores for feature generation and 8 GPUs for ML inference. The data shows that the transmission between controller and ML server contributes a significant fraction of the total end-to-end monitoring delay, while simultaneously being one parameter for optimization. It can be seen that even with an RTT of $100\,\mathrm{ms}$, we are able to perform sub-second predictions, while predictions in less than $500\,\mathrm{ms}$ are possible for a local server. Specifically, when monitoring 524,288 flows, *Marina* transmits a total of $21.5\,\mathrm{MB}$ for every time slot and can achieve a minimum time slot length of $429\,\mathrm{ms}$. Thus, when using a round-number time slot length of $500\,\mathrm{ms}$, *Marina* generates $385\,\mathrm{Mbps}$ of monitoring traffic (including protocol overhead) per P4 switch towards the ML server.

V. *Marina*'s ML-based Real-time Monitoring Performance

We study the resulting monitoring performance, which can be realized based on our *Marina* prototype, for four different real-time traffic monitoring use cases, namely, traffic classification, application health (video streaming), security/anomaly detection (intrusion detection), and device classification (IoT). As different ML models can benefit differently from the extracted statistical features, we compare the performance of several widely used ML models. The ML Inferencing component uses either a shallow model, e.g., Decision Tree (DT), Random Forest (RF), Extremely Randomized Trees

[1]https://github.com/microsoft/hummingbird

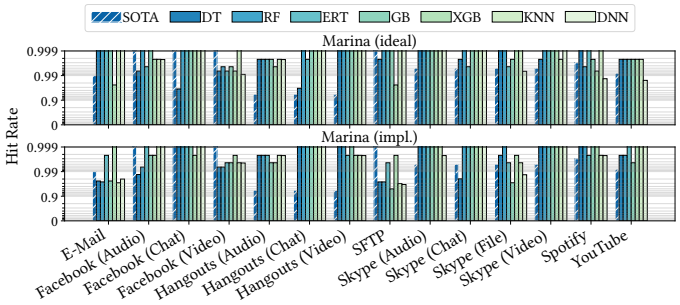

Figure 8: Hit rates for ISCXVPN2016 traffic classification into 14 applications/traffic types.

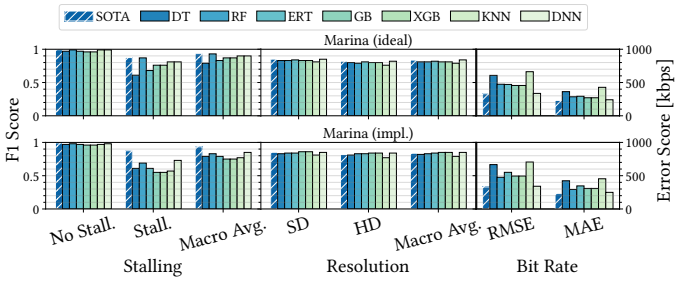

Figure 9: F1 scores for stalling/resolution classification, error scores (RMSE and MAE) for bit rate inference.

(ERT), Gradient Boosting (GB), XGBoost (XGB), or K-nearest neighbor (KNN), or a Deep Neural Network (DNN), namely, a simple feedforward network with three to eight hidden layers with 64 to 1024 neurons each and ReLU activation. Model hyper-parameters are optimized during training, using either sklearn's GridSearch [59] for the shallow models, or Optuna [60] for the DNN. We split the data sets into 80% training, 10% validation, and 10% test data, balancing the training data set by oversampling.

We use the features as described in Table I with *Marina* monitoring time slot set to 1 s. We additionally perform macroaggregations of time slots to consider these features on three different time scales. We use features for the current time slot, a trend macro time slot consisting of the three last time slots, and a session macro time slot consisting of all time slots since the beginning of the flow or session. Note that we deliberately choose to use minimal feature engineering and standard model selection to showcase a lower bound on the ML performance that can be realized with *Marina* with publicly available data sets. When deploying *Marina* in an operational network, the ML workflow (feature generation, model development) can easily be optimized for the actual monitoring tasks at hand considering the available compute resources at the server. This can be expected to further improve the performance results presented here.

We will investigate the performance for two versions of the feature set. First, we take the actual features as generated from the statistics extracted by *Marina*. Since the prototype has to use approximations due to the lack of certain ALU operations and 64-bit arithmetic, we also consider an ideal version with the same features, but without approximations. Thus, this version indicates what performance *Marina* could achieve on future data plane hardware using the same features, but overcoming current feature computation limitations. We will compare the performance of both feature sets to the best state-of-the-art (SOTA) results from literature, which were obtained using the same data sets. We are aware that ML results from literature are never perfectly comparable since they are influenced by data set splits (training, validation, test set), feature selection and preprocessing, search space for model selection and hyperparameter tuning, as well as time budget for model training. Still, SOTA results provide a numerical anchor, which helps to sort in the utility of our *Marina* system for real-time traffic monitoring. All numerical results are detailed in Tables III-VI in the appendix.

① **Encrypted Traffic Classification:** a vital part of network management is the identification of ongoing network traffic, as it enables more advanced tasks such as ensuring QoS and anomaly detection [61]. It can be carried out with varying granularity, such as identifying a specific application (e.g., Spotify versus Skype) or the traffic type (e.g., video versus chat) [61], [62], [63]. We utilize the ISCXVPN2016 data set [64] (only non-VPN traffic), which contains traces for various encrypted traffic types like email, chatting, file transfer, but also for various applications like Spotify, YouTube, Skype, and Facebook. To investigate a more challenging and fine-grained classification task, we also differentiate between audio, chat, file transfer, and video traffic for the applications Facebook, Hangouts, and Skype. We exclude some of the applications like FTPS and SCP, because their single traces are too short, not allowing us to generate a sufficient number of valid training and test slots. We split the time slots into short consecutive sequences of $30-120$ s, and distribute the sequences such that 80% of the time slots are training data, 10% validation data, and 10% test data. As SOTA comparison we choose the results reported by Xie et al. [63]. The SOTA approach here is DL-based, and instead of extracting features, packet traces are treated like a sentence and information is extracted similar to NLP tasks.

The performance of the various ML models is depicted in Figure 8, where the hit rate, i.e., true positive rate, for each traffic type and application is shown for ideal *Marina* and implemented *Marina*. For each monitoring interval, *Marina* performs a real-time prediction. The shown hit rate is computed based on these predictions after the flow has ended. Our system targets the lowest monitoring interval possible. Therefore, we do not investigate the relationship between hit rate and slot size/number of slots in this work, but leave this for future work.

We observe that all ML models perform excellent on the task and that the differences between ideal and implemented are only marginal, i.e., the approximations did not cause any performance loss. We also observe that our models' performance is on a par or even better (e.g., RF, XGB) than SOTA [63]. The results show that *Marina* can be effectively used by network operators to monitor network traffic types and applications.

② **Video Streaming Application Health/Quality of Experience:** video streaming is a prime example of an application, for which prevailing traffic encryption has substantially limited the visibility of DPI-based network telemetry systems [65]. To

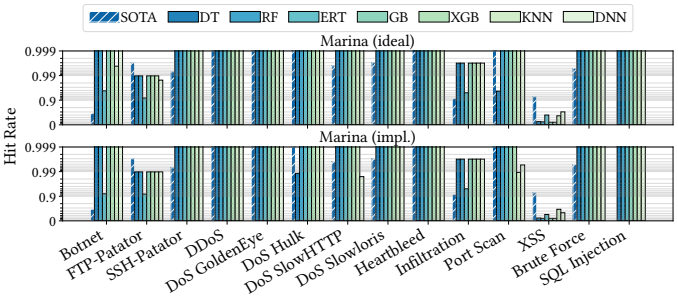

Figure 10: Hit rates for CIC-IDS2017 intrusion detection using 14 attacks.

win back visibility into video streaming application health or Quality of Experience (QoE), ML-based approaches have been conceived that seamlessly operate on traffic features extracted from the encrypted stream of bytes [66], [14], [67], [10], [11], [12]. In particular, standardized video streaming Quality of Experience (QoE) models, such as ITU-T P.1203 [68] or P.1204 [69], consider application-layer key performance indicators for visual quality, such as resolution and bit rate, as well as critical playback events such as stalling/re-buffering [70]. Accordingly, we tackle three separate ML tasks, i.e., stalling and resolution – binary classification, and bit rate – regression. We compare to the SOTA results in [12] using an extended data set that was provided by the authors. It contains the encrypted traffic of more than 16,000 YouTube video sessions labelled with ground truth information obtained at the client side. It was recorded on laptop devices from 2018 to 2019, at different geographical locations and different networks. Note that the authors retrained the model on this extended dataset compared to the initial paper. The feature set is similar to ours and the authors trained a multitude of ML models (e.g., DT, RF, KNN, and many more). The data set contains more than 5,000,000 time slots of 1 s, annotated with ground truth information obtained at the client side. We split into 80% training, 10% validation, and 10% test data on a per-session basis, balancing the training data set by oversampling.

Figure 9 summarizes the results. For stalling and video resolution classification, we show the F1 score for the individual classes (no stalling/stalling and standard (SD) and high definition (HD)) as well as the macro-averaged F1 scores, where the F1 score of each class accounts equally to the overall average F1 score independent of the number of available samples. For bit rate regression, we show root mean square error (RMSE) and mean absolute error (MAE). Overall, results for the ideal *Marina* feature set are on a par with SOTA results. For all stalling and resolution classes, multiple models can reach similar performance to the SOTA models using both ideal and implemented *Marina*. When considering the bit rate regression, the performance of ideal and implemented *Marina* is again close for DNN, but slightly worse than SOTA. In general, the performance difference between ideal and implemented *Marina* is marginal. These results confirm that *Marina* allows network operators to accurately trace the application health and subjective QoE of a large number of video streaming users in real-time, while fully preserving the encryption, and therefore, protecting end users' privacy.

③ **Intrusion Detection:** the ever-growing number and complexity of cyberattacks [71] pushes network operators beyond static defenses, such as firewalls, to deeper Intrusion Detection Systems (IDSs) [72], [73], relying on DPI technology to detect and potentially block known malicious traffic. Despite the complexities and limitations of ML in the networking security realm [74], there is a growing interest on ML for malicious traffic detection, in particular as a countermeasure against attacks hiding in encrypted traffic [75], [38], [76]. We use the CIC-IDS2017 data set [77], [78], featuring a small network with a wide variety of attacks, labeled on a per-flow basis. We are aware of the well-known limitations and issues with this data set [79]. However, we chose to consider it, as it is widely used and allows for comparison with SOTA results. For the SOTA comparison, we use the results reported by Ho et al. [80], which uses a DNN-based approach, utilizing the original 78 flow-based features provided by the publishers of the CIC-IDS2017 dataset in addition to the raw packet captures. In our case and after processing the packet captures, the data set contains more than 20,000,000 time slots with labels from 15 classes: benign or attack, including brute force, denial of service, web attack, etc. For the ML training, we require a balanced dataset, where benign and malicious slots appear equally likely so that the model is able to capture the underlying relationships. As it would be unmanageable to oversample all attacks to the size of the majority class "benign" in this highly imbalanced dataset, it is common practice to undersample the majority class instead. Thus, we first undersample the benign slots by using only 10% of the original slots. To obtain a balanced dataset, we then oversample the attack slots such that their count equals the number of benign slots. We assign sequences of consecutive time slots of $30-300\,\mathrm{s}$ length into 80% training, 10% validation, and 10% test data.

Figure 10 summarizes the results for this multi-class classification problem, reporting the hit rate per attack for all ML models and both feature sets. Except for the cross-site scripting (XSS) web attack, *Marina* detects all attacks equally well, reaching an outstanding hit rate of above 99%, which is on a par with SOTA results [80]. In this use case, we also observe that the restrictions imposed by the P4 hardware have hardly any impact, i.e., using the implemented *Marina* is close to the performance of the ideal *Marina* variant. Additionally, evaluations indicate that most ML models perform equally good for all attack classes. All in all, we see that *Marina* realizes high detection performance for different types of attacks. Together with its high capacity, it is thus well suited to support network operators' security management, monitoring and detecting intrusions at large, on high-speed links. As intrusion detection is both a security and an anomaly detection task, we expect results to transfer also to other anomaly detection tasks, such as detecting faults or changes in the network.

④ **IoT Device Classification:** the increasing number of non-standard computing devices communicating with each other on the Internet of Things (IoT) poses security and privacy risks. Device classification can help network administrators to

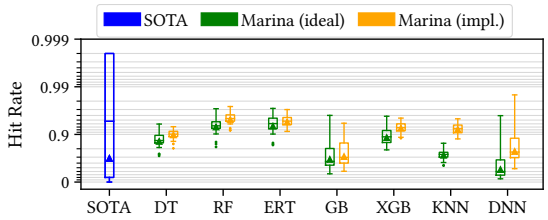

Figure 11: Distributions and mean (▲) of per-class hit rate for CIC-IOT2022 classification into 37 devices.

improve the network security by gaining an overview of the network inventory [81], [82]. ML-based solutions have seen great success for IoT device classification [15], [83], [84], [85]. We use the CIC-IOT2022 data set [86], in which 60 different IoT devices have been monitored in different scenarios and over many hours. Here, we focus on the idle and active experiments with 37 distinct devices only. The goal is to identify the IoT devices based on their traffic characteristics. This is relevant in scenarios where IP addresses change dynamically, or when multiple devices jointly communicate over a single public IP address using NAT. The considered IoT devices in this data set include various audio devices, camera devices, home automation devices, etc., which are based on different protocols like IEEE 802.11, Zigbee, and Z-Wave. We split the traffic captures by MAC address to obtain traffic flows per IoT device. We further ignore all empty time slots where devices were inactive. We split the time slots into 80% training, 10% validation, and 10% test data. For ML training, we randomly sample only 5% of the original time slots and then balance the IoT devices using oversampling. We compare our approach to the SOTA results reported by Ma et al. [85], which utilizes a DL-based approach using packet sequences.

Figure 11 depicts the hit rate distributions of the 37 classes for the various ML models in the form of boxplots. Results show that the multi-class problem of device identification can be adequately solved with *Marina*, achieving average hit rates above 93% for RF and ERT, for both the ideal and implemented *Marina* feature sets. Compared to SOTA [85], which proposes a flow-based approach based on embeddings of packet sequences, *Marina* provides a superior performance. Notably, the variance of the SOTA approach is a lot higher. It is able to identify some devices perfectly, but it fails to recognize others entirely, which also causes the high difference between median and mean. Thus, *Marina* shows a more consistent performance regarding all classes. It can also be seen that for this use case implemented *Marina* often outperforms ideal *Marina*, which seems kind of contradictory as the features are supposed to become more inaccurate due to the approximations and transformations. The transformations, however, are the root cause why implemented *Marina* performs better here. They change the underlying distribution of the features, e.g., the variance inside the features, and simultaneously change the co-variances in a beneficial way for the prediction in this case. Note that while the transformations had a positive effect here, it can also negatively affect the performance in other cases.

Additionally, providing hit rates of at least 91% for each of the 37 IoT devices and considering its high capacity, *Marina* is an appropriate tool for network operators to classify devices on the growing Internet of Things.

VI. Discussion

Denial of Service: using the controller to classify unknown flows into flows that are relevant or irrelevant for monitoring, forwarding the first packet of each flow to the controller, has the potential to make the system susceptible to denial of service attacks. For example, an attack may use SYN flooding to overload the controller. While our prototype currently does not implement any mitigation strategy, SYN flood detection strategies are well known and can be implemented. Moreover, a load dependent sampling mechanism for new flows can be added to avoid controller overload. Note that due to the use of the PCIe interface, there is ample network bandwidth between the data plane and the controller, and the controller is a reasonably powerful system by itself.

Traffic Completeness: *Marina* assumes that all traffic of a user session is observed. Thus, performance might degrade due to multi-path routing in combination with packet by packet multiplexing, asymmetric routing, or other form of link aggregation splitting the traffic. Assessing the impact of traffic (in)completeness is beyond the scope of this paper.

DNS Poisoning and Encrypted DNS: the prototype can rely on a database of IP addresses to decide which flows to selectively monitor for a given use case. This database can be either hard coded or dynamically updated using a mapping of IP addresses to specific domain names of the application, which are extracted from unencrypted DNS responses. DNS poisoning, which may insert irrelevant IP addresses into the IP database, may negatively impact the monitoring system. In addition, the current implementation does not support encrypted DNS, such as DNS over HTTPS (DoH) or DNS over TLS (DoT). Still, the assumption that a network operator has a way to identify relevant IP addresses seems reasonable, even if DoH or DoT is used within their network, since most use cases involve well known services.

Concept Drift: *Marina* relies on ML models which are trained on labelled data sets for monitoring purposes. The challenge is on the availability of such data sets. They need to be representative and available offline for training, while at the same time there is a need to detect and handle concept drift during operation, i.e., long-term changes to the data distributions over time. Possible examples for such changes are updates of the application or the end user device, or changes to network configuration. This may require the collection of new labeled data sets and re-training of the ML models, e.g., [87]. We note that this is not a problem specific to *Marina*, but a generic problem involving ML models in operational networks. As such, we did not evaluate the impact of concept drift in this work.

Hardware Requirements: although the *Marina* data plane prototype was implemented on a P4-based Tofino switch, our design can be ported to any data plane device (programmable switch, FPGA, or ASICs), which provides line rate forwarding and arithmetic packet operations. Thus, in contrast to other approaches, we are not affected by Intel's recent cut of Tofino

network solutions [88].

VII. Related Work

Traditional Solutions: classical flow-based telemetry approaches such as NetFlow [1], sFlow [2], or IPFIX [3] only provide coarse-grained monitoring, as they collect statistics over the whole flow, unsuitable for real-time applications. In detail, NetFlow timeouts are configured in seconds, but practical implementations often do not allow for finer granularities $< 30\,\mathrm{s}$ [89]. Additionally, they often rely on sampling techniques, as monitoring the whole traffic at line rate is impossible, possibly skewing/falsifying the traffic observations.

Sketch-based Solutions: to gain a more fine-grained insight into the network, sketching algorithms have been popularized [30], [31], [90], [91], [92], [93], [94], [95], [96], [29], utilizing customized data structures, e.g., for DDoS mitigation [97]. Sketches generally depict a trade-off between resource consumption and accuracy [30], may only serve a special purpose, and are not designed with ML-based traffic analytics in mind.

In-band Network Telemetry: orthogonal to sketching, INT [98] and extensions, e.g., Probabilistic INT (PINT) [28] emerged. Instead of approximations, INT makes use of piggybacking strategies and dumps meta-data onto the packets. As this imposes a significant overhead, newer proposals [99] revert back to using lightweight sketchlets with INT.

Query-driven Languages: often building on INT and/or sketches, query-driven languages [20], [100], [19], [33] aim to provide expressive interfaces to run flexible streaming queries. However, providing these interfaces often comes with significant overhead, e.g., Sonata [20] renders short ($< 3\,\mathrm{s}$) time slots undesirable due to updates of filter rules.

Software-based Solutions: to realize the above discussed telemetry systems, either software- or hardware-based implementations are possible. With *Marina*, we opt for a hardware-based solution. Though, several software-based academic (e.g., Retina [21]), commercial (e.g., Corelight [101]), or open-source (e.g., Zeek [102], nProbe [103], Tstat [104]) solutions exist. While some of them are capable of monitoring multi-Gbps traffic, they do not scale to Tbps traffic, and/or are tailored to a specific use case. Zhang et al. [105] provide an overview of such software-based solutions that operate on a Gigabit scale, but as mentioned by Sonchack et al. [106] scaling these solutions to huge networks on a Terabit scale requires a rack full of servers.

ML-driven Solutions: some monitoring solutions implement the ML network intelligence paradigm, aiming to deploy ML models or feature extraction on commodity hardware or programmable switches. Nevertheless, after a vast review of related literature (see Table II in the appendix for the full taxonomy), we conclude that existing ML-based solutions are not ready for deployment, as they have unclear performance, e.g., in terms of overall latency of the whole ecosystem. Additionally, related works often only operate on data for a specific use case, i.e., while some approaches might theoretically be applicable for more than the described use case, it was not evaluated in the corresponding paper.

VIII. Conclusion

We introduced *Marina*, a system for realizing ML-driven real-time traffic monitoring in large scale networks. *Marina* addresses the challenges of scalability up to terabit scale while minimizing the monitoring overhead and providing high flexibility, expressiveness, and accuracy for performance- and security-related traffic monitoring tasks, even in case of encrypted traffic. The design of *Marina* is based on spreading the monitoring over a highly efficient data plane on a programmable switch, FPGA, or ASIC, which can extract monitoring data at line rates, and a powerful ML server, which can run monitoring inference using diverse ML models. We link both parts by applying temporal microaggregation of packets per flow into sub-second time slots. We extract a stream of sample moments of the packet size and inter-arrival time distributions. This information is available even for encrypted traffic and provides a valuable description of the traffic in each time slot that can be leveraged by ML-based monitoring models. The time slot duration both defines the real-time capabilities and the monitoring accuracy of *Marina*, and thus, must be kept as short as possible.

We implemented a *Marina* data plane prototype on a Barefoot Wedge 100BF-65X P4 switch and made the code publicly available. As current P4 hardware exhibits several limitations, we had to utilize a P4 bag of tricks to realize *Marina* and to approximate the extracted statistics. The *Marina* prototype maxes out the data plane resources to monitor up to $6.4\,\mathrm{Tbps}$ of traffic in 524,288 concurrent flows. It generates less than $385\,\mathrm{Mbps}$ of monitoring traffic, and, in combination with a powerful ML server, it can achieve a monitoring granularity and end-to-end delay until obtaining monitoring results for all flows as low as $500\,\mathrm{ms}$.

We validated the analysis capabilities provided by *Marina* for four different and challenging ML-driven real-time monitoring applications – encrypted traffic classification, video streaming application health/Quality of Experience monitoring from encrypted traffic, intrusion detection, IoT device classification – with a broad set of ML models. For all investigated tasks, despite the approximations required due to P4 hardware limitations, the ML inference results enabled by the *Marina* prototype are on a par or better than state-of-the-art results. We found that random forest models provide a good trade-off between high monitoring accuracy and fast model inference speed. However, in an operational deployment, the best ML models can be selected and optimized depending on the actual monitoring tasks and the available compute resources, which can be expected to further improve the monitoring performance. Considering its monitoring capacity at terabit scale, this confirms that *Marina* allows to realize different ML-driven real-time monitoring tasks in large-scale networks with high accuracy.

Acknowledgment

This work was partly funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant SE 3163/3-1, project number: 500105691 (UserNet). The authors alone are responsible for the content.

REFERENCES

[1] B. Claise, "Cisco systems netflow services export version 9," Internet Requests for Comments, RFC Editor, RFC 3954, October 2004, http://www.rfc-editor.org/rfc/rfc3954.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3954.txt

[2] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 2004, pp. 628–635.

[3] B. Claise, B. Trammell, and P. Aitken, "Specification of the ip flow information export (ipfix) protocol for the exchange of flow information," Internet Requests for Comments, RFC Editor, STD 77, September 2013, http://www.rfc-editor.org/rfc/rfc7011.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7011.txt

[4] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers & Security*, vol. 70, pp. 238–254, 2017.

[5] T.-L. Huoh, Y. Luo, P. Li, and T. Zhang, "Flow-based encrypted network traffic classification with graph neural networks," *IEEE Transactions on Network and Service Management*, 2022.

[6] W. Zheng, C. Gou, L. Yan, and S. Mo, "Learning to classify: A flow-based relation network for encrypted traffic classification," in *Proceedings of The Web Conference 2020*, 2020, pp. 13–22.

[7] T. Kim and W. Pak, "Real-time network intrusion detection using deferred decision and hybrid classifier," *Future Generation Computer Systems*, vol. 132, pp. 51–66, 2022.

[8] N. Borgioli, L. Thi Xuan Phan, F. Aromolo, A. Biondi, and G. Buttazzo, "Real-time packet-based intrusion detection on edge devices," in *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, 2023, pp. 234–240.

[9] J. Kampeas, A. Cohen, and O. Gurewitz, "Traffic classification based on zero-length packets," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1049–1062, 2018.

[10] M. H. Mazhar and M. Z. Shafiq, "Real-time Video Quality of Experience Monitoring for HTTPS and QUIC," in *IEEE INFOCOM*, Honolulu, HI, USA, 2018.

[11] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time qoe detection for encrypted youtube traffic," in *ACM Multimedia Systems Conference (MMSys)*, 2019.

[12] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li, "Vicrypt to the rescue: Real-time, machine-learning-driven video-qoe monitoring for encrypted streaming traffic," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2007–2023, 2020.

[13] P. Casas, S. Wassermann, M. Seufert, N. Wehner, O. Dinica, and T. Hoßfeld, "X-ray goggles for the isp: Improving in-network web and app qoe monitoring with deep learning," in *6th Network Traffic Measurement and Analysis Conference, TMA 2022, June 27-30, 2022*, V. Bajpai, H. Haddadi, and O. Hohlfeld, Eds. IFIP, 2022.

[14] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring Video QoE from Encrypted Traffic," in *ACM Internet Measurement Conference (IMC)*, Santa Monica, CA, USA, 2016.

[15] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

[16] D. Cerović, V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "Fast packet processing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3645–3676, 2018.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[18] O. Michel, R. Bifulco, G. Rétvári, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–36, 2021.

[19] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *ACM SIGCOMM Conference*, 2017, pp. 85–98.

[20] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *ACM SIGCOMM Conference*, 2018, pp. 357–371.

[21] G. Wan, F. Gong, T. Barbette, and Z. Durumeric, "Retina: analyzing 100gbe traffic on commodity hardware," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 530–544.

[22] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Automating in-network machine learning," *arXiv preprint arXiv:2205.08824*, 2022.

[23] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, H. Haddadi, G. Antichi, and R. Bifulco, "Running neural networks on the NIC," *CoRR*, vol. abs/2009.02353, 2020. [Online]. Available: https://arxiv.org/abs/2009.02353

[24] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pforest: In-network inference with random forests," *arXiv preprint arXiv:1909.05680*, 2019.

[25] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Iisy: Practical in-network classification," *arXiv preprint arXiv:2205.08243*, 2022.

[26] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *18th ACM workshop on Hot Topics in Networks*, 2019, pp. 25–33.

[27] V. Sekar, M. K. Reiter, and H. Zhang, "Revisiting the case for a minimalist approach for network flow monitoring," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 328–341.

[28] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *ACM SIGCOMM Conference*, 2020, pp. 662–680.

[29] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 29–42.

[30] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *ACM SIGCOMM Conference*, 2016, pp. 101–114.

[31] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *ACM SIGCOMM Conference*, 2018, pp. 561–575.

[32] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao *et al.*, "Packet-level telemetry in large datacenter networks," in *ACM SIGCOMM Conference*, 2015, pp. 479–491.

[33] C. Misa, W. O'Connor, R. Durairajan, R. Rejaie, and W. Willinger, "Dynamic scheduling of approximate telemetry queries," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). USENIX Association, Renton, WA*, 2022, pp. 701–717.

[34] M. Yu, "Network telemetry: towards a top-down approach," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 1, pp. 11–17, 2019.

[35] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *NDSS*, 2021.

[36] J. A. Shohat and J. D. Tamarkin, *The problem of moments*. American Mathematical Society (RI), 1950, vol. 1.

[37] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 20–29.

[38] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," in *25th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[39] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-assisted ddos attack detection with p4 language," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[40] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-enabled ddos attacks detection in p4 programmable networks," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 1–27, 2022.

[41] M. H. Pathmaperuma, Y. Rahulamathavan, S. Dogan, and A. M. Kondoz, "Deep learning for encrypted traffic classification and unknown data detection," *Sensors*, vol. 22, no. 19, p. 7643, 2022.

[42] A. J. Pinheiro, J. d. M. Bezerra, C. A. Burgardt, and D. R. Campelo, "Identifying iot devices and events based on packet length from encrypted traffic," *Computer Communications*, vol. 144, pp. 8–17, 2019.

[43] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Managing iot cyber-security using programmable telemetry and machine learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 60–74, 2020.

[44] N. Wehner, M. Ring, J. Schüler, A. Hotho, T. Hoßfeld, and M. Seufert, "On Learning Hierarchical Embeddings from Encrypted Network Traffic," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–7.

[45] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[46] W. Braun and M. Menth, "Software-defined networking using openflow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[47] M. P. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2013, pp. 1009–1016.

[48] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software defined networking flow table management of openflow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, p. 147, 2020.

[49] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[50] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," in *European Symposium on Algorithms*. Springer, 2006, pp. 456–467.

[51] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[52] J.-J. Lim and K. G. Shin, "Gradient-ascending routing via footprints in wireless sensor networks," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE, 2005, pp. 10–pp.

[53] I. N. Bermudez, M. Mellia, M. M. Munafo, R. Keralapura, and A. Nucci, "Dns to the rescue: Discerning content and services in a tangled web," in *Internet Measurement Conference*, 2012, p. 413–426.

[54] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 67–82.

[55] L. Yu, J. Sonchack, and V. Liu, "Mantis: Reactive programmable switches," in *ACM SIGCOMM Conference*, 2020, pp. 296–309.

[56] L. Kleinrock, *Queueing Systems – Volume I: Theory*. John Wiley, 1975.

[57] D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research: Stochastic Processes and Operating Characteristics*. Dover Publications, 2003.

[58] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network–measurements, models, and implications," *Computer networks*, vol. 53, no. 4, pp. 501–514, 2009.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[60] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[61] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[62] T. Shapira and Y. Shavitt, "Flowpic: Encrypted internet traffic classification is as easy as image recognition," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 680–687.

[63] G. Xie, Q. Li, Y. Jiang, T. Dai, G. Shen, R. Li, R. Sinnott, and S. Xia, "Sam: Self-attention based deep learning method for online traffic classification," in *Proceedings of the Workshop on Network Meets AI & ML*, 2020, pp. 14–20.

[64] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[65] P. Casas, M. Seufert, and R. Schatz, "Youqmon: A system for online monitoring of youtube qoe in operational 3g networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.

[66] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward Quality-of-Experience Estimation for Mobile Apps from Passive Network Measurements," in *15th Workshop on Mobile Computing Systems and Applications (HotMobile)*, Santa Barbara, CA, USA, 2014.

[67] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "A Machine Learning Approach to Classifying YouTube QoE Based on Encrypted Network Traffic," *Multimedia Tools and Applications*, vol. 76, no. 21, pp. 22 267–22 301, 2017.

[68] International Telecommunication Union, "ITU-T Recommendation P.1203: Parametric Bitstream-based Quality Assessment of Progressive Download and Adaptive Audiovisual Streaming Services over Reliable Transport," 2017. [Online]. Available: https://www.itu.int/rec/T-REC-P.1203/en

[69] ——, "ITU-T Recommendation P.1204: Video Quality Assessment of Streaming Services Over Reliable Transport for Resolutions up to 4K," 2020. [Online]. Available: https://www.itu.int/rec/T-REC-P.1204-202001-P/en

[70] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2015.

[71] Check Point Research, "Cyber Security Report 2022," Check Point Research, Tech. Rep., 2022, accessed: 2023-02-08. [Online]. Available: https://www.checkpoint.com/downloads/resources/cyber-security-report-2022.pdf

[72] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on sdn based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[73] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[74] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 305–316.

[75] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical real-time intrusion detection using machine learning approaches," *Computer Communications*, vol. 34, no. 18, 2011.

[76] N. Gray, K. Dietz, M. Seufert, and T. Hossfeld, "High performance network metadata extraction using p4 for ml-based intrusion detection systems," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2021, pp. 1–7.

[77] Canadian Institute for Cybersecurity. (2017) Intrusion Detection Evaluation Dataset (CIC-IDS2017). (accessed 2023-02-08). [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

[78] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[79] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *2021 IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 7–12.

[80] S. Ho, S. Al Jufout, K. Dajani, and M. Mozumdar, "A novel intrusion detection model for detecting known and innovative cyberattacks using convolutional neural network," *IEEE Open Journal of the Computer Society*, vol. 2, pp. 14–25, 2021.

[81] C. Kuzniar, M. Neves, V. Gurevich, and I. Haque, "Iot device fingerprinting on commodity switches," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–9.

[82] H. Jmila, G. Blanc, M. R. Shahid, and M. Lazrag, "A survey of smart home iot device classification using machine learning-based traffic analysis," *IEEE Access*, 2022.

[83] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[84] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and {IoT} devices from (public){WiFi}," in *29th USENIX security symposium (USENIX security 20)*, 2020, pp. 55–72.

[85] J. Ma, Y. Sang, Y. Zhang, X. Xu, B. Feng, and Y. Zeng, "An adaptive ensembled neural network-based approach to iot device identification," in *Collaborative Computing: Networking, Applications and Worksharing: 18th EAI International Conference, CollaborateCom 2022, Hangzhou, China, October 15-16, 2022, Proceedings, Part II*. Springer, 2023, pp. 214–230.

[86] S. Dadkhah, H. Mahdikhani, P. K. Danso, A. Zohourian, K. A. Truong, and A. A. Ghorbani, "Towards the development of a realistic multidimensional iot profiling dataset," in *2022 19th Annual International Conference on Privacy, Security & Trust (PST)*, 2022, pp. 1–11.

[87] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[88] D. Harris, "Time to save: Intel stopped development of network switches and ended the support program for chip designers on RISC-V," 2023, accessed 2023-02-07. [Online]. Available: https://technewsspace.com/time-to-save-intel-stopped-development-of-network-switches-and-ended-the-support-program-for-chip-designers-on-risc-v/

[89] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *2014 IEEE 34th International Conference on Distributed Computing Systems*. IEEE, 2014, pp. 228–237.

[90] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *ACM SIGCOMM Conference*, 2017, pp. 113–126.

[91] Q. Huang, P. P. Lee, and Y. Bao, "Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference," in *ACM SIGCOMM Conference*, 2018, pp. 576–590.

[92] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *ACM SIGCOMM Conference*, 2019, pp. 334–350.

[93] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "Cocosketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 207–222.

[94] H. Zheng, C. Tian, T. Yang, H. Lin, C. Liu, Z. Zhang, W. Dou, and G. Chen, "Flymon: enabling on-the-fly task reconfiguration for network measurement," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 486–502.

[95] Q. Huang, S. Sheng, X. Chen, Y. Bao, R. Zhang, Y. Xu, and G. Zhang, "Toward nearly-zero-error sketching via compressive sensing," in *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*, 2021, pp. 1027–1044.

[96] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "SketchLib: Enabling efficient sketch-based monitoring on programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/namkung

[97] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *the 27th Network and Distributed System Security Symposium (NDSS 2020)*, 2020.

[98] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM Conference*, vol. 15, 2015.

[99] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang *et al.*, "Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets." in *NSDI*, 2021, pp. 991–1010.

[100] Q. Huang, H. Sun, P. P. Lee, W. Bai, F. Zhu, and Y. Bao, "Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 404–421.

[101] Corelight, "Corelight," 2022, accessed 2023-02-08. [Online]. Available: https://corelight.com/

[102] Zeek, "Zeek," 2022, accessed 2023-02-08. [Online]. Available: https://zeek.org/

[103] ntop, "nProbe," 1998, accessed 2023-12-06. [Online]. Available: https://www.ntop.org/products/netflow/nprobe/

[104] T. N. G. P. di Torino, "Tstat TCP STatistic and Analysis Tool," 2008, accessed 2023-12-06. [Online]. Available: http://tstat.polito.it/

[105] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "Flowatcher-dpdk: Lightweight line-rate flow-level monitoring in software," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1143–1156, 2019.

[106] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *13th EuroSys Conference*, 2018, pp. 1–16.

[107] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, and H. Hu, "Fastfe: Accelerating ml-based traffic analysis with programmable switches," in *Workshop on Secure Programmable Network Infrastructure*, 2020, pp. 1–7.

[108] C. Hardegen, "Scope-based flow monitoring to improve traffic analysis in programmable networks," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 254–260.

[109] P. Schmitt, F. Bronzino, R. Teixeira, T. Chattopadhyay, and N. Feamster, "Enhancing transparency: Internet video quality inference from network traffic." TPRC, 2018.

[110] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[111] ——, "Map4: A pragmatic framework for in-network machine learning traffic classification," *IEEE Transactions on Network and Service Management*, 2022.

[112] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association*, 2023.

[113] R. Kamath and K. M. Sivalingam, "Machine learning based flow classification in dcns using p4 switches," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–10.

[114] T.-N. Dao and H. J. Lee, "Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection," *IEEE Internet of Things Journal*, 2021.

[115] T.-N. Dao, V.-P. Hoang, C. H. Ta *et al.*, "Development of lightweight and accurate intrusion detection on programmable data plane," in *2021 International Conference on Advanced Technologies for Communications (ATC)*. IEEE, 2021, pp. 99–103.

[116] A. T.-J. Akem, M. Gucciardo, M. Fiore *et al.*, "Flowrest: Practical flow-level inference in programmable switches with random forests," in *IEEE International Conference on Computer Communications*, 2023.

[117] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1938–1947.

[118] Q. Qin, K. Poularakis, and L. Tassiulas, "A learning approach with programmable data plane towards iot security," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 410–420.

[119] J.-H. Lee and K. Singh, "Switchtree: in-network computing and traffic analyses with random forests," *Neural Computing and Applications*, pp. 1–12, 2020.

[120] M. Zang, C. Zheng, R. Stoyanov, L. Dittmann, and N. Zilberman, "P4pir: in-network analysis for smart iot gateways," in *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, 2022, pp. 46–48.

[121] A. T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Henna: hierarchical machine learning inference in programmable switches," in *Proceedings of the 1st International Workshop on Native Network Intelligence*, 2022, pp. 1–7.

[122] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the ai accelerator?" in *2018 Morning Workshop on In-Network Computing*, 2018, pp. 20–25.

[123] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, 2021, pp. 12–14.

[124] X.-H. Nguyen, X.-D. Nguyen, H.-H. Huynh, and K.-H. Le, "Realguard: A lightweight network intrusion detection system for iot gateways," *Sensors*, vol. 22, no. 2, p. 432, 2022.

[125] A. Ahmim, L. Maglaras, M. A. Ferrag, M. Derdour, and H. Janicke, "A novel hierarchical intrusion detection system based on decision tree and rules-based models," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2019, pp. 228–233.

[126] L. Reuter, O. Jung, and J. Magin, "Neural network based anomaly detection for scada systems," in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2020, pp. 194–201.

Michael Seufert (Senior Member, IEEE) is a Full Professor at the University of Augsburg, Germany, heading the Chair of Networked Embedded Systems and Communication Systems. He received the Bachelor's degree (2018) in economathematics and the Diploma (2011), PhD (2017), and Habilitation (2023) degrees in computer science from the University of Würzburg, Germany, and holds the First State Examination degree (2011) in mathematics, computer science, and education for teaching in secondary schools. His research focuses on user-centric communication networks, including QoE of Internet applications, AI/ML for QoE-aware network management, as well as group-based communications.

Katharina Dietz is a Research Assistant at the Chair of Communication Networks at the University of Würzburg, where she is pursuing her PhD. She studied computer science at the University of Würzburg, Germany and received her Master's degree in 2020. Her research mainly focuses on managing and securing communication networks with machine learning-based approaches, ranging from performance prediction to anomaly detection.

Nikolas Wehner studied computer science at the University of Würzburg, Germany, where he received a Master's degree. In 2018, he started to work as a Research Engineer at the Center for Technology Experience at the AIT Austrian Institute of Technology in Vienna, Austria. Since October 2019, he is a doctoral researcher at the Chair of Communication Networks of the University of Würzburg. His interests are user-centric communication networks, focusing on the QoE of Internet applications, and machine learning for networks in general.

Stefan Geißler Stefan Geißler received the PhD degree for his thesis "Performance Evaluation of Next-Generation Data Plane Architectures and their Components" from the University of Würzburg, Würzburg, Germany, in 2022. He is a Research Associate with the Chair of Communication Networks, University of Würzburg, where he leads the Cloud Applications and Networks Research Group. His current research is focused on the analytical and simulative performance evaluation and optimization of next-generation communication technologies, including TSN, 5G, and Internet of Things.

Joshua Schüler received the Master's degree in computer science from the University of Würzburg, Würzburg, Germany, where he also worked as a student research assistant at the Chair of Communication Networks. He is currently a Digital Systems Engineer at Tesat-Spacecom GmbH & Co. KG, Backnang, Germany, where he works on network equipment for satellites. His research interests include artificial intelligence applications in computer networks and fault tolerant embedded systems.

Manuel Wolz received the Bachelor's degree in computer science from University of Würzburg, Würzburg, Germany in 2020, and is currently studying towards the Master's degree. Since 2018 he is working as student research assistant at the Chair of Communication Networks of the University of Würzburg with a focus on network simulation, benchmarking, and P4.

Andreas Hotho holds the Chair of Data Science at the Julius Maximilian University of Würzburg and is the spokesperson of the Centre for Artificial Intelligence and Data Science (CAIDAS) at the University of Würzburg. In recent years, he has conducted research in various areas of data science. Core research areas include NLP with the integration of knowledge into language models, the study of historical novels in cooperation with the digital humanities, the analysis of corporate data for recommendation systems or anomaly detection, e.g. the analysis of network flow data, as well as studies based on sensor data on air pollution, climate modelling and bee behaviour.

Pedro Casas (Member, IEEE) received the Electrical Engineering degree from the Universidad de la República, Uruguay, in 2005, and the Ph.D. degree in computer science from Télécom Bretagne, in 2010. He is currently a Senior Scientist in AI/ML for Networking, with the AIT Austrian Institute of Technology, Vienna. He was a Postdoctoral Researcher with the LAAS-CNRS, Toulouse, from 2010 to 2011, and a Senior Researcher with the Telecommunications Research Center Vienna, from 2011 to 2015. His work focuses on machine learning-based approaches for networking, big data analytics and platforms, Internet network measurements, network security, and anomaly detection, as well as Internet QoE monitoring. He has published more than 200 networking research papers in major international conferences and journals, and received 18 awards for his work, including eight Best Paper Awards. He is the General Chair for different actions in network measurement and analysis, including the IEEE ComSoc ITC Special Interest Group on Network Measurements and Analytics.

Tobias Hoßfeld is professor at the Chair of Communication Networks at the University of Würzburg, Germany, since 2018. He finished his PhD in 2009 and his professorial thesis (habilitation) "Modeling and Analysis of Internet Applications and Services" in 2013 at the University of Würzburg, where he was also heading the "Future Internet Applications & Overlays" research group. From 2014 to 2018, he was head of the Chair "Modeling of Adaptive Systems" at the University of Duisburg-Essen, Germany. He has published more than 100 research papers in major conferences and journals, receiving several best conference paper awards, 3 awards for his PhD thesis, and the Fred W. Ellersick Prize 2013 (IEEE Communications Society) for one of his articles on QoE. He is member of the editorial board of IEEE Communications Surveys & Tutorials, Springer Quality and User Experience, ACM SIGMM Records and elected chairperson of the ITG/VDE expert group "Communication Networks and Systems" within the German society of Information Technology (ITG).

Anja Feldmann got her Ph.D. from Carnegie Mellon University in 1995. The next four years she did research work at AT&T Labs Research, before taking professor positions at Saarland University, the TU Munich, and TU Berlin. From 2012 to 2018 she served on Supervisory Board of SAP SE. Since the beginning of 2018, Anja is a director at the Max Planck Institute for Informatics in Saarbruecken, Germany. She is a member of multiple academies and on the steering committees of CoNEXT and IMC. She was TPC-chair of Sigcomm, IMC, CoNEXT, as well as HotNets.

APPENDIX

A. Appendix – Taxonomy of ML-driven Solutions

System	Monitoring Task					Performance		Implementation			
	TC	QoE	ID	IoT	UCA	Parallel Flows	Line Rate	Platform	CA	Task of PDP in ML Pipeline	ML Algorithm
Marina (this work)	✔	✔	✔	✔	✔	524,288	100 Gbps	HW	✔	Time slot-based moments	Various
FastFE [107]	✘	✘	✔	✘	✘	(✘)	40 Gbps	HW	✘	Time slot-based statistics	Neural Networks (NN)
Musumeci et al. [39], [40]	✘	✘	✔	✘	✘	(✘)	10 Gbps	SW	✘	Time slot-based statistics	RF, KNN, SVM, **no** NN
Sivanathan et al. [43]	✘	✘	✘	✔	✘	(✘)	(✘)	(✘)	✘	Time slot-based statistics	Random Forest
Gray et al. [76]	✘	✘	✔	✘	✘	65,536	100 Gbps	HW	✔	Flow-based statistics	Random Forest
FlowMoni [108]	✘	✘	✔	✘	✘	∼352k - 2M[a]	(✘)	SW	✔	(Sub)flow-based statistics	RF, DNN
PoirIoT [81]	✘	✘	✘	✔	✘	(✘)	40 Gbps	HW	✔	Rule-based metadata extraction	DBSCAN
NetMicroscope [109]	✘	✔	✘	✘	✘	(✘)	(✘)	(✘)	✘	ML-based session quality inference	Regression (various)
Xavier et al. [110], [111]	✘	✘	✔	✔	✘	(✘)	10 Gbps	HW	✔	ML-based flow/packet classification	Decision Trees
IIsy [25]	✘	✘	✔	✘	✔	(✘)	100 Gbps	HW	(✔)[e]	ML-based flow/packet classification	Various, **no** NN
NetBeacon [112]	✔	✘	✔	✘	✔	65,536	100 Gbps	HW	✔	ML-based flow/packet classification	Decision Trees
SMASH [113]	✔	✘	✘	✘	✘	(✘)	(✘)	SW	✔	ML-based flow classification	Decision Trees
Dao et al. [114], [115]	✘	✘	✔	✘	✘	(✘)	30 Mbps	(✘)	✘	ML-based flow classification	Neural Networks
FlowLens [35]	✔	✘	✔	✘	✔	>250k[b]	100 Gbps	HW	✔	ML-based flow classification	Various, **no** NN
pForest [24]	✘	✘	✔	✘	✘	∼250k - 1M[c]	(✘)	HW[d]	✘	ML-based subflow classification	Random Forest
Flowrest [116]	✔	✘	✔	✔	✔	(✘)	100 Gbps	HW	✔	ML-based subflow classification	Random Forests
Mousika [117]	✔	✘	✔	✘	✔	(✘)	100 Gbps	HW	✔	ML-based packet classification	Decision Trees
IIsy (prototype) [26]	✘	✘	✘	✔	✘	(✘)	10 Gbps	HW	✔	ML-based packet classification	Various, **no** NN
Qin et al. [118]	✘	✘	✔	✘	✘	(✘)	(✘)	SW	✔	ML-based packet classification	Neural Networks
SwitchTree [119]	✘	✘	✔	✘	✘	(✘)	(✘)	SW	✔	ML-based packet classification	Random Forest
P4pir [120]	✘	✘	✔	✘	✘	(✘)	(✘)	SW	✘	ML-based packet classification	DT, RF, **no** NN
Henna [121]	✘	✘	✘	✔	✘	(✘)	100 Gbps	HW	✔	ML-based packet classification	Decision Trees
BaNaNa [122]	✘	✘	✘	✘	✘	(✘)	(✘)	HW	✔	ML-based classification, **no** FE	Neural Networks
N3IC [23]	✘	✘	✔	✘	✘	(✘)	40 Gbps	HW	✘	ML-based classification, **no** FE	Neural Networks
Planter [123], [22]	✘	✔	✔	✘	✔	(✘)	100 Gbps	HW	✔	ML-based classification, **no** FE	Various

[a] theoretically approximated for Intel Tofino 3 chip architectures, [b] minimum, no upper limit discussed, [c] per 10 MB memory, derived from SW, [d] only limited functionality of software implementation, [e] only the same code as for the prototype available

Table II: Related ML-driven monitoring systems. (TC – traffic classification, QoE – application health/Quality of Experience, ID – intrusion detection, IoT – Internet of Things, UCA – use case agnostic, HW/SW – hardware/software platform, CA – code available, FE – feature extraction, ✔ – fulfilled, ✘ – not fulfilled, (✔) – partially fulfilled, (✘) – unknown)

B. Appendix – ML Benchmark Results

			Facebook			Hangouts				Skype					
Model	Features	Mail	Audio	Chat	Video	Audio	Chat	Video	SFTP	Audio	Chat	File	Video	Spotify	YouTube
SOTA	[61]	0.820	0.950			0.980			1.000	0.990				0.980	0.990
	[63]	0.990	>0.999			0.943			>0.999	0.995				0.997	0.992
DT	*Marina* (ideal)	>0.999	0.993	0.965	0.993	0.998	0.967	>0.999	0.998	>0.999	0.998	>0.999	0.998	>0.999	0.998
	Marina (impl.)	0.976	0.987	>0.999	0.993	0.998	>0.999	>0.999	0.974	>0.999	0.980	0.998	>0.999	>0.999	0.998
RF	*Marina* (ideal)	>0.999	>0.999	>0.999	0.996	0.998	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.996	0.998
	Marina (impl.)	0.974	0.993	>0.999	0.993	0.998	>0.999	>0.999	0.974	>0.999	>0.999	>0.999	>0.999	>0.999	0.998
ERT	*Marina* (ideal)	>0.999	0.996	>0.999	0.993	0.998	0.998	>0.999	>0.999	>0.999	0.996	0.996	>0.999	>0.999	0.998
	Marina (impl.)	0.998	>0.999	>0.999	0.996	0.998	>0.999	0.998	0.996	>0.999	>0.999	0.996	>0.999	0.998	>0.999
GB	*Marina* (ideal)	>0.999	>0.999	>0.999	0.996	0.998	>0.999	>0.999	>0.999	>0.999	>0.999	0.998	>0.999	0.998	0.998
	Marina (impl.)	0.976	0.998	>0.999	0.996	0.998	>0.999	>0.999	0.95 0	>0.999	>0.999	0.972	>0.999	>0.999	0.996
XGB	*Marina* (ideal)	0.976	0.998	>0.999	0.993	0.996	>0.999	>0.999	0.976	>0.999	>0.999	>0.999	>0.999	0.993	0.998
	Marina (impl.)	>0.999	0.998	0.998	0.998	0.996	>0.999	0.998	0.998	>0.999	>0.999	0.998	>0.999	>0.999	>0.999
KNN	*Marina* (ideal)	>0.999	0.999	>0.999	>0.999	0.998	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.998	>0.999	0.998
	Marina (impl.)	0.972	>0.999	>0.999	0.996	0.998	>0.999	0.998	0.967	0.998	>0.999	0.987	>0.999	0.998	>0.999
DNN	*Marina* (ideal)	>0.999	0.998	>0.999	0.991	0.998	>0.999	>0.999	>0.999	>0.999	>0.999	0.993	>0.999	0.987	0.984
	Marina (impl.)	0.980	>0.999	>0.999	0.996	0.998	>0.999	0.998	0.967	0.9998	>0.999	0.987	>0.999	0.998	>0.999

Table III: ML performance of *Marina* for encrypted traffic classification. Hit rates for traffic classes.

Model	Features	Stalling			Resolution			Bit Rate	
		F1 - No Stalling	F1 - Stalling	F1 - Macro	F1 - SD	F1 - HD	F1 - Macro	RMSE [kbps]	MAE [kbps]
SOTA	[12]*	0.993	0.875	0.934	0.851	0.819	0.837	331	231
DT	*Marina* (ideal)	0.965	0.612	0.789	0.827	0.797	0.812	607	363
	Marina (impl.)	0.966	0.609	0.787	0.828	0.807	0.817	667	424
RF	*Marina* (ideal)	0.993	0.870	0.931	0.825	0.791	0.808	474	286
	Marina (impl.)	0.976	0.691	0.834	0.836	0.826	0.831	476	293
ERT	*Marina* (ideal)	0.992	0.863	0.927	0.815	0.784	0.800	537	358
	Marina (impl.)	0.982	0.685	0.834	0.832	0.831	0.831	564	364
GB	*Marina* (ideal)	0.974	0.679	0.826	0.837	0.807	0.822	471	293
	Marina (impl.)	0.967	0.612	0.789	0.844	0.831	0.837	551	347
XGB	*Marina* (ideal)	0.983	0.762	0.872	0.832	0.796	0.814	456	271
	Marina (impl.)	0.959	0.550	0.755	0.862	0.844	0.853	495	309
KNN	*Marina* (ideal)	0.988	0.809	0.899	0.807	0.763	0.785	663	428
	Marina (impl.)	0.970	0.567	0.768	0.813	0.774	0.793	705	454
DNN	*Marina* (ideal)	0.987	0.788	0.887	0.848	0.817	0.832	337	241
	Marina (impl.)	0.983	0.748	0.865	0.849	0.841	0.846	341	250

Table IV: ML performance of *Marina* for video streaming application health/Quality of Experience monitoring from encrypted traffic. F1 scores for binary stalling and resolution classification, RMSE and MAE for bit rate regression. *SOTA results are not exactly as given in [12] as authors provided an extended data set and retrained their model to obtain comparable results.

Model	Features	Bot	FTP-P.	SSH-P.	DDoS	GE	Hulk	SH	SL	Heartb.	Infil.	Port	XSS	BF	SQL
SOTA	[80]	0.664	0.997	0.993	>0.999	>0.999	>0.999	0.996	0.997	1.000	0.917	>0.999	0.928	0.995	0.810
	[124]	0.985	>0.999	>0.999	1.000	>0.999	0.985	0.995	>0.999	N/A	>0.999	>0.999	N/A	N/A	N/A
	[125]	0.465	0.996	>0.999	0.999	0.676	0.968	0.938	0.978	1.000	1.000	0.999	0.306	0.733	0.500
	[114]	0.000	0.936	0.505	0.922	0.882	0.733	0.771	0.577	0.200	0.000	0.994	0.000	0.048	0.000
	[111]	N/A	N/A	N/A	N/A	0.870	1.000	0.940	0.950	N/A	N/A	0.940	N/A	0.950	N/A
	[126]	0.314	0.542	0.608	0.991	0.870	0.992	0.769	0.635	N/A	N/A	0.971	0.235	0.185	N/A
DT	*Marina* (ideal)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	0.957	0.262	>0.999	>0.999
	Marina (impl.)	>0.999	0.990	>0.999	>0.999	>0.999	0.988	>0.999	>0.999	>0.999	0.997	>0.999	0.234	>0.999	>0.999
RF	*Marina* (ideal)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.241	>0.999	>0.999
	Marina (impl.)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.207	>0.999	>0.999
ERT	*Marina* (ideal)	0.959	0.918	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.950	>0.999	0.600	>0.999	>0.999
	Marina (impl.)	0.921	0.918	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.950	>0.999	0.448	>0.999	>0.999
GB	*Marina* (ideal)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.214	>0.999	>0.999
	Marina (impl.)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.207	>0.999	>0.999
XGB	*Marina* (ideal)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.214	>0.999	>0.999
	Marina (impl.)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.207	>0.999	>0.999
KNN	*Marina* (ideal)	0.996	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.572	>0.999	>0.999
	Marina (impl.)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	0.989	0.662	>0.999	>0.999
DNN	*Marina* (ideal)	>0.999	0.985	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	>0.999	0.997	>0.999	0.703	>0.999	>0.999
	Marina (impl.)	>0.999	0.990	>0.999	>0.999	>0.999	>0.999	0.984	>0.999	>0.999	0.997	0.995	0.531	>0.999	>0.999

Table V: ML performance of *Marina* for intrusion detection. Hit rates for attack detection (Bot – Botnet, FTP-P. – FTP-Patator, SSH-P. – SSH-Patator, DDoS – Distributed Denial of Service, GE – DoS GoldenEye, Hulk – DoS Hulk, SH – DoS SlowHTTP, SL - DoS Slowloris, Heartb. – Heartbleed, Infil. – Infiltration, Port – Port Scan, XSS - Cross-site Scripting, BF - Brute Force, SQL – SQL Injection).

Model	Features	Mean	Min	5%	25%	50%	75%	95%	Max
SOTA	[85]	0.682	0.000	0.007	0.200	0.947	0.998	1.000	1.000
DT	*Marina* (ideal)	0.860	0.719	0.743	0.850	0.866	0.895	0.917	0.939
	Marina (impl.)	0.898	0.804	0.856	0.889	0.901	0.914	0.920	0.931
RF	*Marina* (ideal)	0.930	0.817	0.853	0.925	0.934	0.946	0.964	0.971
	Marina (impl.)	0.952	0.917	0.924	0.947	0.954	0.961	0.967	0.974
ERT	*Marina* (ideal)	0.933	0.833	0.847	0.925	0.941	0.954	0.98	0.972
	Marina (impl.)	0.945	0.914	0.920	0.938	0.947	0.956	0.960	0.970
GB	*Marina* (ideal)	0.666	0.329	0.450	0.558	0.631	0.803	0.892	0.960
	Marina (impl.)	0.709	0.407	0.510	0.600	0.695	0.849	0.931	0.942
XGB	*Marina* (ideal)	0.884	0.789	0.814	0.852	0.888	0.917	0.939	0.958
	Marina (impl.)	0.926	0.879	0.898	0.916	0.928	0.940	0.950	0.955
KNN	*Marina* (ideal)	0.727	0.550	0.600	0.701	0.737	0.763	0.816	0.847
	Marina (impl.)	0.922	0.876	0.895	0.908	0.924	0.936	0.949	0.953
DNN	*Marina* (ideal)	0.454	0.146	0.168	0.283	0.398	0.658	0.911	0.960
	Marina (impl.)	0.772	0.476	0.554	0.692	0.766	0.880	0.978	0.985

Table VI: ML performance of *Marina* for IoT device classification. Mean and percentiles of the hit rate distribution for the detection of 37 IoT devices (classes).