FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications

Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello†, Fernando M. V. Ramos, André Madeira
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
{diogo.barradas, nuno.m.santos, ler, fvramos, andre.madeira}@tecnico.ulisboa.pt
†LASIGE, Faculdade de Ciências, Universidade de Lisboa
ssignorello@ciencias.ulisboa.pt

***Abstract*—An emerging trend in network security consists in the adoption of programmable switches for performing various security tasks in large-scale, high-speed networks. However, since existing solutions are tailored to specific tasks, they cannot accommodate a growing variety of ML-based security applications, i.e., security-focused tasks that perform targeted flow classification based on packet size or inter-packet frequency distributions with the help of supervised machine learning algorithms. We present FlowLens, a system that leverages programmable switches to efficiently support multi-purpose ML-based security applications. FlowLens collects features of packet distributions at line speed and classifies flows directly on the switches, enabling network operators to re-purpose this measurement primitive at runtime to serve a different flow classification task. To cope with the resource constraints of programmable switches, FlowLens computes for each flow a memory-efficient representation of relevant features, named "flow marker". Despite its small size, a flow marker contains enough information to perform accurate flow classification. Since flow markers are highly customizable and application-dependent, FlowLens can automatically parameterize the flow marker generation guided by a multi-objective optimization process that can balance their size and accuracy. We evaluated our system in three usage scenarios: covert channel detection, website fingerprinting, and botnet chatter detection. We find that very small markers enable FlowLens to achieve a 150 fold increase in monitoring capacity for covert channel detection with an accuracy drop of only 3% when compared to collecting full packet distributions.**

I. Introduction

Recently, several systems have been proposed for tackling security concerns in modern high-speed networks [90, 33, 49, 82]. By leveraging the capabilities offered by programmable switches, these systems can process packets at line speed directly on the switch hardware, bringing relevant benefits for network security, such as decreased reaction times to attacks, avoidance of network bottlenecks, and decreased costs associated to equivalent centralized server-based infrastructures. So far, the proposed systems target very specific security-driven tasks. These tasks include the ability to mitigate DDoS attacks [90], enforce context-aware security policies [33], obfuscate network topologies [49], filter spoofed traffic [37], or detect data exfiltration through timing covert channels [82].

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24067
www.ndss-symposium.org

However, besides the specific tasks tackled by the previous work, there is currently a lack of support for a new range of security applications that resort to machine learning (ML) to classify flows in real time [92, 27]. This brand of applications has become more relevant as a result of a global trend towards encrypting all Internet traffic [20, 58], which has rendered deep-packet inspection (DPI) increasingly ineffective. As an alternative to DPI, the use of ML-based techniques has proved useful to classify flows with high accuracy for a wide range of scenarios, such as multimedia covert channel detection [7], website fingerprinting [40], botnet traffic identification [53], malware tracking [2], IoT device behavioral analysis [59, 79], or detection of DRM-protected streaming [26, 68, 66].

Most of these ML-based applications rely on supervised machine learning algorithms [7, 40, 68, 53] that need to collect flow features such as packet length and/or inter-packet time frequency distributions. However, both the set of features and ML algorithms used are highly application-dependent. As such, a general service to enable the implementation of ML-based security applications must be versatile enough to accommodate application-specific requirements without impairing its ability to produce accurate classification results. In addition, it must efficiently use the limited switch resources to maximize the number of flows that can be probed, scale to large networks comprising numerous switches, introduce minimal switch downtime caused by upgrades of switch programs, and require low maintenance effort.

We present FlowLens, a system that enables efficient flow classification for multi-purpose ML-based security applications. At the heart of our system lies a set of software components that run on the network switches' data plane and control plane. These components are responsible for collecting compact, but meaningful, features of the flows going past the data plane, and for running the ML-based algorithms responsible for classifying the flows on the control plane in real time. By performing both these tasks on the switches in a fully decentralized fashion, FlowLens does not depend on a centralized service that could introduce bottlenecks for operations in the critical path. To deliver the best performance, these software components must be fine-tuned for each specific ML-based security application. FlowLens includes the mechanisms to generate (and upload to the switches) application-dependent configurations that strike a good balance between classification accuracy and switch resource utilization efficiency. Because these configurations can be automatically generated, the maintenance effort of our system is greatly reduced.

A key challenge in fully offloading ML-based flow analysis onto the switches is tied to the hardware and programming restrictions of modern programmable switches. Ideally, we would like to collect the full packet length and inter-packet arrival time frequency distributions for every flow traveling through the switches. This approach would allow us to collect full per-flow information (on the data plane) which different applications could then process in order to extract relevant features and running specific ML classifiers (on the control plane). However, given that the amount of stateful switch memory is very limited, an information-lossless scheme for collecting flow data would considerably reduce the coverage of our system, i.e., the number of simultaneous flows that could be probed. In alternative, one could employ a lossfull scheme where the amount of dedicated memory allocated per flow is reduced thereby increasing flow coverage. Such a scheme, however, must be such that (1) the collected information does not deteriorate the accuracy of flow classification, (2) it can be implemented with a small set of basic hardware instructions and within few compute cycles as imposed by the switch, and (3) it precludes the need to frequently reprogram the switch as it would cause switch downtime in the order of seconds.

To address this challenge, we make two core technical contributions. First, we devised a new compact representation of packet length and inter-time packet arrival distributions which is small yet provides enough information to perform accurate application-specific traffic classification. We name such representations *flow markers*. We then developed a primitive named Flow Marker Accumulator (FMA) which generates flow markers while depending on simple and efficient operations that can be implemented on modern programmable switches. The FMA consists of a parameterizable data structure deployed on the data plane pipeline such that, for each incoming packet, it performs two simple operations, namely *quantization* and *truncation*, which adjust the granularity of the flow's frequency distribution intervals in bins (quantization), and select the bins considered to be the most relevant features for flow classification (truncation). As shown in Figure 1, the set of resulting bins for each flow constitutes the respective flow marker, which will then be processed by the classification algorithm.

Second, we developed an automatic profiler to find adequate quantization and truncation parameters of the FMA for a given application. Because there is a large space of configurations that present different trade-offs between switch memory savings and flow classification accuracy, manually setting up these parameters for each application would both be cumbersome and render sub-optimal results. Our profiler relies on well-known Bayesian optimization techniques [24] for finding suitable configurations by iteratively testing only a small subset of the possible FMA configurations. It can be tuned to find parameterizations according to different criteria, including (a) the maximization of a user-defined trade-off between space-efficiency and accuracy, (b) the smallest marker able to achieve a classification accuracy above a given threshold, or (c) the marker that maximizes the accuracy given some space constraint.

We have implemented FlowLens on a Barefoot/Intel Tofino programmable switch and evaluated our system in three use case applications: covert channel detection, website fingerprinting, and botnet detection. When comparing the classification scores achieved by FlowLens against those computed over raw packet length distributions, FlowLens offers similar accuracy scores while using significantly less memory, e.g., covert channels can be detected with at most 3% loss in accuracy using only a 20-byte memory footprint per flow. When compared with related methods for capturing compressed packet frequency distributions [22, 55], FlowLens consistently outperforms them in terms of the classification accuracy under similar memory restrictions. FlowLens also achieves considerable bandwidth savings when compared to network telemetry approaches [74] that rely on a server infrastructure responsible for flow analysis.

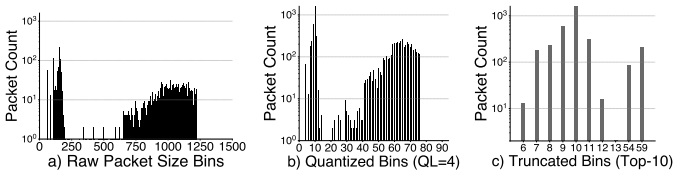

Figure 1. Histograms of packet size distribution for a single flow. A flow consists of a stream of packets identified by the same 5-tuple of TCP/IP header fields, at increasing degrees of compactness: a) the raw packet size distribution; b) the quantized representation, where packet sizes are aggregated into bins of size 2^4 bytes; c) the flow marker generated through truncation of the quantized representation which comprises the most relevant 10 bins for the detection of covert channels mounted through multimedia protocol tunneling.

II. Motivation and Design Goals

This section motivates our work by characterizing a set of emerging ML-based security applications and discussing the technical constraints of modern programmable switches. It then provides an overview of FlowLens's design goals.

A. ML-based Network Security Applications

In recent years, generalized interest has grown in detecting atypical network flows using ML classification algorithms [58]. To deliver accurate flow classification results, these algorithms depend on a range of features that require the collection of the packet length/inter-packet timing frequency distributions. Below, we present three examples of applications in the realm of network security that rely on the analysis of such distributions for performing traffic classification. These examples are chosen to showcase the versatility of FlowLens in accommodating different classification algorithms. We will further use them to validate the classifiers' accuracy when deployed on FlowLens.

Covert channel detection: Capturing packet distributions makes it possible to detect covert channels, thereby providing a valuable asset for cyberforensic investigations. To achieve stealthy data transmissions, advanced covert channel tools tend to obfuscate covert flows such that their high-level features (e.g. packet lengths) resemble those of regular flows [81, 6, 38, 47]. However, recent work [7] has shown that these tools can be defeated or severely weakened due to subtle differences in packet distributions which can be detected by ML techniques.

Website fingerprinting: Privacy-enhancing technologies like OpenSSL or OpenVPN allow users to hide the destination address behind a proxy and the content of website visits from external observers through the use of encryption. However, it may still be possible to identify which sites they access by collecting the flows' packet length distributions [40, 29] and feeding them to ML classification algorithms for website

fingerprinting purposes. This technique may help authorities respond against individuals engaged in illegal activities.

Botnet chatter detection: Botnets [35] can jeopardize the security of multiple organizations, emerging as a highly profitable activity for malicious actors [63]. Unfortunately, due to their decentralized P2P architectures and stealthy communication patterns, botnets have become incredibly resistant to takedown attempts. Nevertheless, state-of-the-art approaches to analyzing botnet traffic are able to identify the presence of bots through the combined analysis of packet lengths and inter-packet timing distributions of network hosts [48, 53]. Being able to employ these techniques can help network administrators prevent and mitigate botnet threats to an organization's network.

B. Design Goals

In this work, our goal is to use programmable switches to collect packet frequency distributions and provide a multi-purpose flow classification platform for implementing a variety of ML-based security applications. By using our solution, a network operator will be able to scan local traffic in near real-time and look for specific flows that match a set of application-specific traffic patterns, such as those presented in Section II-A. In summary, we are driven by the following design goals:

Scalability: We aim at monitoring flows in very large and fast networks (at the Tbps scale), comprised of many switches, while reducing the costs of the network telemetry infrastructure. To this end, we aim to avoid relying either on edge-based solutions, which capture the traffic through middleboxes [61], or solutions that collect packet features on the switches but offload them for further processing and classification on dedicated servers [73, 74]. Reducing the bandwidth consumed with the offloading of telemetry data is also a crucial point as the amount of collected data grows with the increasingly high link speeds and becomes substantial for large scale networks [87].

Accuracy: We aim at collecting a compact representation of packet distributions while retaining enough information about flows to enable high accuracy on classification tasks. Achieving such a representation requires the design of a flow compression scheme that is simple enough to be efficiently implemented by the primitives available in current P4-programmable switches, but which is able to retain meaningful features for classification purposes. Additionally, computing compact representations of packet distributions should not consume the majority of resources in the switch, enabling the system to co-exist with other typical applications, e.g. forwarding, or to be used in tandem with complementary network telemetry solutions.

Availability: Re-purposing our system to different traffic analysis tasks should not involve the deployment of a new P4 program. This is because deploying a new program involves a scheduled downtime during which the switch will be unable to perform its basic functions, causing service disruptions.

C. Constraints of Modern Programmable Switches

To efficiently collect and process packet distributions, we explore the programming capabilities of modern switches, such as Barefoot Tofino [4] and Broadcom Tomahawk II [14]. These switches include two types of processors which operate in two different planes of the network architecture. On the data plane, forwarding ASICs are able to quickly forward and perform simple computations on packets at line-rate, thus enabling the analysis of billions of packets at the Tbps scale. On the control plane, CPUs can be used for general-computing tasks such as controlling the packet forwarding pipeline, or for exchanging data with the ASIC through DMA.

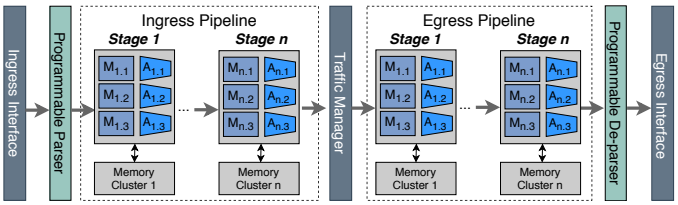

Figure 2. Protocol Independent Switch Architecture (PISA).

Switching ASICs can be programmed in a hardware-independent language, such as P4 [12]. Figure 2 illustrates the architecture of our targeted switching ASIC: the Protocol Independent Switch Architecture (PISA) [18]. Packets arrive at the switch ingress interfaces and, after parsing, are processed by two logical pipelines of *match+action units* (MAUs) arranged in stages. Packet headers along with packet metadata may then *match* (M) a given table, triggering further processing by the *action* (A) unit associated with the matching table's entry. These actions may modify packet header fields and change persistent state (e.g., increment a counter in stateful memory). Tables and other objects defined in a P4 program are instantiated inside MAUs and populated by the control plane at run-time.

Memory constraints: Several constraints in the memory architecture of switching ASICs may restrict the layout of the data structures that can be used by P4 programs. These ASICs are equipped with two high-speed types of memory: (i) TCAM, which is a content addressable memory suited for fast table lookups, such as for longest-prefix matching in routing tables [89], and (ii) SRAM which enables P4 programs to persist state across packets (e.g., using register arrays), and to hold exact-match tables. Unfortunately, switching ASICs contain a small amount of stateful memory (in the order of 100MB SRAM [51]), and only a fraction of the total available SRAM can be used to allocate register arrays. Moreover, accessing all available registers can be a complex task since the registers in one stage cannot be accessed at different stages [19]; this is because the SRAM is uniformly distributed amongst the different stages of the processing pipeline (see Figure 2).

Processing constraints: The P4 programs installed on the switch must also use very simple instructions to process packets. To guarantee line-rate processing, packets must spend a fixed amount of time in each pipeline stage (a few ns [70, 69]) which restricts the number and type of operations allowed within each stage. Multiplications, divisions or floating-point operations, and variable-length loops are not supported. Moreover, each table's action can only perform a restricted set of simpler operations, like additions, bit shifts, and memory accesses that can quickly be performed while the packet is passing through an MAU without stalling the whole pipeline [71].

III. System Overview

This section describes FlowLens, a system for efficient flow classification that achieves the aforementioned goals. Figure 3 shows the architecture of FlowLens, illustrating how it can

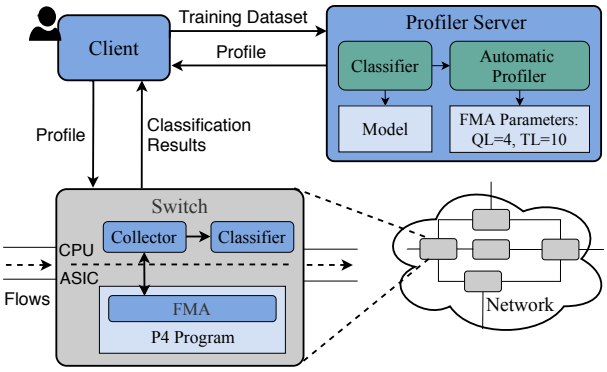

Figure 3. FlowLens architecture and components.

be used to monitor traffic on a single switch of a high-speed network. In general, it can be deployed across multiple other switches in the network at the system operator's discretion. FlowLens consists of the following components: a *P4 program* and two software components (*collector* and *classifier*) running on the switch, a standalone *profiler* server, and a software *client* that provides an interface to the system operator.

Taken together, the components running on the switch are responsible for analyzing traffic and classifying flows as per ML-based security application. The P4 program runs on the data plane, and implements a tailor-made data structure named *Flow Marker Accumulator* (FMA). The FMA is used to collect concise encodings of the packet length or inter-packet timing distributions of flows named *flow markers*. Essentially, the FMA implements the necessary knobs for fine-tuning the memory savings and the classification accuracy of the system. It can accommodate the restrictions of the switch and generate flow markers at different compression levels by carefully adjusting a set of configuration parameters that control (i) the size of the memory footprint allocated per-flow marker, and (ii) the loss of information in the flow markers due to compression.

Running on the local CPU of the switch, two additional FlowLens software components implement several control plane functions. The *collector* is responsible for loading the P4 program, configuring the FMA data structures in the forwarding pipeline, initiating the flow collection process, and collecting the resulting flow markers. The *classifier* runs the ML algorithms responsible for classifying flows based on the collected markers. FlowLens can generally employ any ML algorithm that can reason over flow markers, and whose memory and compute requirements can be accommodated on the switch control plane. After the classification step, results can be downloaded by the client and displayed to the system operator.

Since the classifier and the FMA configuration parameters depend on the application domain, FlowLens uses a standalone *profiler* to pre-configure the classifier models and FMA parameters (i.e., the application *profile*) onto the switch. The system workflow involves two phases: profiling and flow classification.

1. Profiling: FlowLens needs to be pre-configured by the system operator for specific applications. This operation involves the profiler server, which can automatically create profiles by using an application-specific classifier and a training set containing labeled flow samples provided by the system operator. To this end, the profiler runs an optimization process that explores the classification performance of different FMA quantization and truncation values, generating a configuration according to a given user-defined criterion (see Section V).

2. Flow classification: As soon as the P4 program has been loaded into the switch (which happens only once when bootstrapping FlowLens), the collector takes the profile computed in phase 1 for a specific application, e.g., website fingerprinting, and configures the FMA accordingly. Afterward, the switch can start to process packets and compute flow markers. The collector then fetches the resulting flow markers from the data plane and the classifier processes them based on the loaded model. The classifier results can be retrieved by the system operator, who can then take targeted actions about particular flows such as dropping flagged flows or scheduling further logging operations. The system operator can later reconfigure the system for other ML-based application profiles without the need to re-deploy the P4 program.

In the following sections, we present the relevant design details of FlowLens, namely the FMA data structure (Section IV) and the automatic profiling scheme aimed at choosing markers with good accuracy/memory saving trade-offs (Section V).

IV. Flow Marker Accumulator

The Flow Marker Accumulator (FMA) is the data structure responsible for computing flow markers on the switch data plane. Next, we present its internal operations by describing the FMA design for capturing markers of packet length distributions, and presenting the main changes when computing inter-packet arrival timing distributions. Then, we describe how the FMA is used by the control plane, and discuss alternative FMA setups.

A. Collecting Packet Length Distributions

To generate flow markers, the FMA provides two basic operators that can be implemented efficiently and be used to obtain a space-efficient encoding of a packet length distribution: *quantization* and *truncation*. Quantization consists of counting packet lengths in coarse bins that represent ranges of contiguous packet lengths. Truncation further trims the number of bins that need to be reserved for a certain classification task. These operators allow us to selectively collect the bin values which, in many cases, correspond to the most relevant features employed by the ML engine to yield accurate classifications [84, 7].

To perform these operations, the FMA is composed of several data structures shown in Figure 4. They consist of two match+action tables and one register array: the *flow table*, the *truncation table*, and the *register grid*, respectively. The register grid is a matrix of memory registers. Each line is used to store a flow marker. The index of each line (flow offset) is used to address the flow marker. Internally, a flow marker consists of a number of cells in a register (the grid's columns). These cells play the role of bins for storing samples of the flow's packet length frequency distribution. The flow table maps the monitored flows against the respective flow markers in the register grid. The truncation table identifies the bin that must be incremented for every incoming packet.

Next, we describe in detail the procedure for updating the flow marker for a given incoming packet. Consider the example shown in Figure 4, where an input packet arrives in the switch

from source IP 162.2.13.42, source port 41065, with length 1024 bytes. The FMA performs the following four operations:

1. Lookup: First, the FMA's flow table matches the incoming packet with the corresponding flow ID, which is a 5-tuple of header fields $\langle IP_{src}, Port_{src}, IP_{dst}, Port_{dst}, Proto \rangle$ that is used as lookup key to return its associated flow offset. To be efficiently performed, we leverage the match+action units of the switch to accommodate specific rules for flow table indexing. Each rule in the flow table assigns a unique flow offset to each flow ID. For instance, in the running example, the input packet is matched against the rule $\langle 162.2.13.42, 41065, 146.3.18.71, 80, 6 \rangle \rightarrow 0$. (Section IV-C describes how the flow table is populated.)

2. Quantization: The flow offset determined above locates the packet's flow marker in the register grid. Next, the FMA must increment the correct bin in the flow marker which is a function of the packet length. The first step to determine the right bin involves quantization, which aggregates, and so counts, a range of contiguous packet lengths into the same bin. To avoid complex instructions unsupported by the switch hardware (e.g., multiplications), the bin indexed by a certain packet length PL is computed by $bin(QL, PL) = length(PL) >> QL$, where QL denotes the *quantization level* and $0 \leq QL < log_2(PL_{max})$. For efficient lookup of a packet length's bin, FMA uses power-of-two bin sizes; this allows for computing the packet bin by right-shifting the packet length value by QL number of bits. In the shown example, applying $QL = 4$ to the packet length (1024 bytes) yields quantization bin #64.

3. Truncation: Based on the obtained quantization bin, truncation leverages an auxiliary data structure – the truncation table – which contains match+action rules exclusively for the bins that should be accounted for in the flow marker. Each rule is keyed by the quantized bin length, i.e., $bin(QL, PL)$, and indexes the flow marker's bin (bin offset) where the packet length frequency must be recorded. If no such rule exists for a given quantized bin, the packet is not counted. In this example, the current packet is considered because a rule exists for the packet's quantized bin (#64). In contrast, packets whose quantized bin values fall, e.g., within the bin range 52-63, will not be accounted for. This strategy allows for selectively filtering the most meaningful bins for flow classification.

4. Increment: Lastly, by combining the flow offset and the bin offset, the register grid can be indexed and the correct bin incremented. In the running example, this entails incrementing bin 2 of the flow marker pointed to by the flow offset 0. These steps are repeated for every incoming packet.

B. Collecting Packet Timing Distributions

Gathering inter-packet timing distributions requires only minor modifications to the FMA design. This task does not affect the main FMA data structures and operators, but it requires additional resources in the switch processing pipeline.

To compute the arrival time difference between two consecutive packets of the same flow, the FMA stores the timestamp of the last packet seen per each flow. That information is available to the P4 program through device intrinsic metadata. With exception to the first packet of a flow, FMA computes the difference between the current packet timestamp and the last timestamp observed for that particular flow. That value can then be processed by the same quantization and truncation operators described for packet lengths, which produce the corresponding bin to be updated in the register grid.

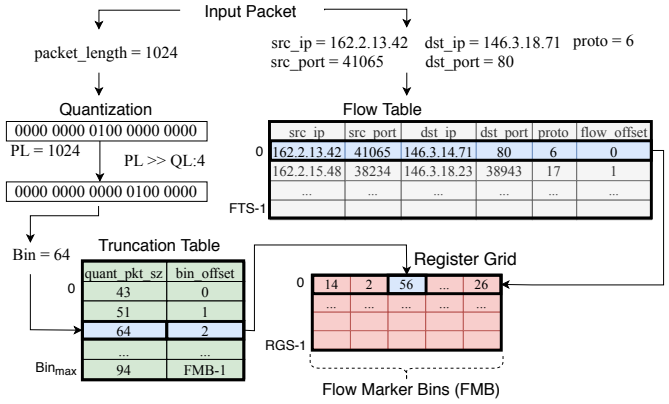

Figure 4. FMA internals: Flow Table, Truncation Table, and Register Grid.

C. Usage of the FMA by the Control Plane

To monitor flows on a switch, the FMA must be coordinated by the control plane's collector software. The collector predefines the FMA's quantization and truncation parameters, and determines: i) which amongst all flows traversing the switch at a given time will be monitored by the FMA, and ii) for how long the packets of the monitored flows will be considered in the respective flow markers. Next, we present the default FMA operational settings and then describe alternative customization policies that can be enabled by the FlowLens operator.

Default FMA measurement operations: By default, the control plane sets up the FMA to i) compute flow markers for all the flows on a first-come, first-served (FCFS) basis until they exhaust the register grid capacity, and ii) measure the flows' respective packets for a predefined time interval that we refer to as *collection window*. The control plane starts by clearing all registers of the FMA's register grid, and setting a timer for the duration of the collection window. Then, for every incoming packet for which a rule does not exist in the FMA's flow table (i.e., the packet belongs to a new flow), the control plane automatically installs one of two possible rules for that flow. If there is free space in the register grid, then it will install a new rule that points to a free flow marker, allowing the flow to be monitored by the FMA. Otherwise, if the register grid is full, the control plane will install a single wildcard *ignore rule* (featuring a reserved offset value) instructing the FMA to ignore that flow and all subsequent flows until the end of the collection window. When the timer expires, the control plane reads in batch all the computed flow markers. To prevent concurrent updates while reading, the control plane deletes the flow rules prior to reading the registers. Once the registers have been read, a new collection window round can then ensue. A side effect of this design is that FlowLens may skip the packets of a new flow until the flow's respective rule is installed in the switch. However, while this is a limitation of our system, such loss does not impair FlowLens's ability to trace most typical flows as they tend to last longer than a few milliseconds.

Discretionary flow monitoring: Prior to the enforcement of the FCFS flow marker allocation strategy, it is possible to filter which traffic should be monitored and therefore limit the amount

of flows to inspect. For instance, ML-based security applications are frequently focused on traffic that can be identified by common target ports (e.g., HTTP). In other cases, the FlowLens operator may be interested in monitoring flows based on IP or network address ranges. Such a discretionary flow filtering stage can be implemented on the control plane by installing ignore rules for all uninteresting flows based on allow / deny policies provided by the FlowLens operator. Ignore rules can be defined to target a single flow or to perform wildcard matching based on IP and port ranges, specific protocol fields, etc.

Fine-tuning of the collection window: Flow markers should not linger for an arbitrarily large amount of time inside FMA's data structures, as this would prevent the flow marker's memory from being used for other flows. To increase the number of flows that can be monitored at any given time, the collection window can be configured, based on three setups: i) the definition of a fixed window duration (explained above); ii) the use of a specific flag set in FMA data structures that enables the control plane to check for flow termination through a polling procedure; iii) a hybrid of both approaches. While option i) is arguably the simplest alternative, it may lead to memory waste since short-lived flows can occupy the FMA's data structures longer than necessary. In contrast, a polling approach allows FlowLens to pinpoint flows that will receive no new packets (e.g., after detecting FIN packets in the data plane pipeline which signals the termination of a TCP connection), but it may indefinitely keep flows which termination is not explicit (e.g., UDP-based multimedia flows). Thus, a hybrid collection approach allows us to proactively read and reset the FMA data structures in use by terminated flows while preventing long-lived flows to be monitored indefinitely. In other words, this approach fully refreshes the register grid every time the collection window is over and partially updates it whenever a particular row is in condition to be evicted.

Flow marker eviction: A high rate of new flows may saturate the capacity of the FMA data structures and prevent storing flow markers for all flows crossing the switch. In this case, as explained above, the FMA's default strategy is to not track new flows as long as existing flows are still being tracked. As an alternative behavior, it is possible to evict flow markers from the FMA according to an LRU policy. In this case, the control plane keeps track of the oldest flow markers stored by the FMA, and replaces them as new incoming flows cross the switch. The most suitable policy will greatly depend on the expected workload and topology of the network.

D. Distributed and Orchestrated FMA Operation

So far, we explained several design decisions of FlowLens when considering its operation to be contained within a single switch. In this section, we describe how FlowLens can benefit from a deployment in multiple vantage points.

Scaling the number of measured flows: Although the number of flows whose state can be kept by a single switch is limited, it is possible to take advantage of multiple vantage points in the network for monitoring a larger amount of flows. This is akin to the operation of other measurement frameworks [43, 31] and may be accomplished, for instance, by splitting packets coming from different IP address spaces between existing switches in the organization's network infrastructure.

Increasing collection coverage: In the case that our system operates with a maximum collection window (see Section IV-C), reading and resetting FMA's data structures requires a non-negligible amount of time (a few seconds) [36]. This may prevent FlowLens from collecting flow information while these operations take place. To ensure visibility over the network traffic crossing an organization, FlowLens can be deployed in a cascade fashion across an additional switch to intertwine the collection windows of the different switches.

Increasing application coverage: The design of FlowLens is tailored for enabling a single profile to be loaded into a given FMA at any given time. However, a coordinated operation of FlowLens across several switches can provide support to multiple ML-based security applications. For instance, when deployed across multiple switches, one FMA instance may be dedicated to the detection of covert channels, while other is dedicated to the identification of botnet behavior.

V. Automatic Profiling

The flow markers generated by FMA depend on the parameters, i.e., the quantization level and the truncation table, dictated by an application-specific profile which determines how efficiently the switch SRAM will be used and how accurate the flow classification will be. In general, finding the parameters that offer an optimal trade-off would require an exhaustive search of the parameter space. Unfortunately, this is a cumbersome task that requires non-trivial computational resources and time, e.g., automatically exploring the full space of configurations for the botnet detection task (Section VII-F) took one day.

To search on the parameter space for a configuration that offers a good trade-off between flow marker size and classification accuracy, the profiler implements optimization techniques that, albeit may fail to yield the optimal result, usually find near-optimal solutions quickly. Next, we describe the optimization criteria and algorithm employed in FlowLens. Note that FlowLens is not tightly coupled to a specific implementation of the profiler and nothing prevents the use of alternate optimization techniques [72, 9]. Investigating further optimization approaches is outside the scope of this paper.

A. Optimization Criteria

We expect that a FlowLens's operator will want to find a suitable FMA parameterization for any given ML-based security application. Because there is a space/accuracy trade-off in the FMA configuration, we are faced with a multi-criteria optimization problem that does not have a single optimal solution but, instead, has a number of Pareto optimal solutions [57]. The current version of the FlowLens's profiler can approximate three different pre-set points in the Pareto frontier, that can be selected by the system operator:

1. Smaller marker for target accuracy: In this mode, the system operator specifies a target accuracy value to be attained, and the profiler automatically chooses the quantization and truncation parameters that yield the smallest marker that is able to offer the target accuracy. Note that the profiler will not return a configuration if the accuracy set by the user cannot be achieved for the particular dataset under analysis.

2. Best accuracy given a size constraint: Here, the system operator specifies the maximum size for the flow marker and the system automatically picks the quantization and truncation parameters that maximize the classification accuracy, among the configurations explored, without exceeding the target marker size. This constraint also allows us to reduce the search space, since the marker size generated by a set of quantization and truncation parameters is known beforehand.

3. Size vs. accuracy trade-off: Lastly, the profiler can work in a fully automated fashion. In this case, the profiler attempts to maximize an accuracy vs marker size trade-off that is expressed by the following reward function: $\textit{reward} = \alpha \cdot \textit{accuracy} + (1-\alpha)\frac{1}{\textit{marker size}}$. A smaller α attributes less importance to the accuracy in favor of compactness, and vice-versa. Our prototype uses $\alpha = 0.5$, but the system operator can define the value of α as well as a different reward policy of its choosing altogether.

B. Optimization Algorithm

The search space is the product of the different quantization and truncation configurations; on its own, the number of configurations that result from truncation is combinatorial with the number of available bins. To guide the profiler's search, we use an optimization algorithm that consists of two phases. In the first phase, called *search space reduction*, we use domain knowledge to narrow the search, by excluding configurations that are unlikely to offer acceptable results. In the second phase, we resort to *Bayesian optimization* to find a suitable flow marker. We detail these two steps in the next paragraphs.

1. Search space reduction: To reduce the search space, we discretize the domain of quantization parameters, e.g., aggregate bins in powers of two. Then, we leverage a pre-training step for narrowing the truncation space: we first generate a coarser representation of the packet distributions of each sample in the training data according to a given quantization parameter, then we use a classifier to build a model based on these representations. We leverage the fact that most classifiers can output information regarding the top-N most relevant bins for accurate classification. Thus, for each quantization, we constrain the exploration to points that include an increasing number of features from the top-50 (i.e., the configuration that includes only the top-10 bins, the top-20 bins, etc). When the classifier is unable to output the top-N features, we fall back to a simpler strategy to narrow the search space: we sample the input space, exclude bins that have not been observed in the sampled points, and feed the remaining ones to the Bayesian optimizer.

2. Bayesian optimization: To reduce the manual labor required to explore a large space of configurations, we rely on Bayesian optimization, which is a well-known method for optimizing black-box functions and for finding near-optimal solutions with few function evaluations [24]. We optimize the combination of quantization and truncation parameters using the Python Hyperopt [10] software package. In each iteration, the profiler selects a parameterization, trains a classifier using flow markers accordingly generated, and records the classifier accuracy alongside the size of produced flow markers. The next parameterization to sample is selected by the optimizer which we run for a fixed number of iterations. The whole process took us a few hours to complete.

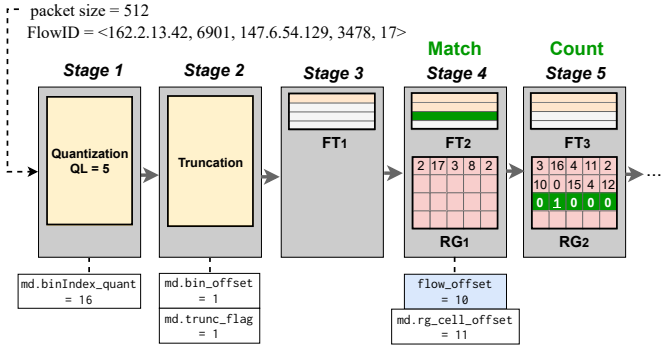

Figure 5. FMA implementation on Tofino switch. Each partition of the flow table (FT_m) records `flow ids` (marked in orange) and their corresponding flow markers are recorded in the register grid partition of the upcoming stage (RG_m). This packet is matched in FT_2 and the respective flow marker updated in RG_2. The white boxes in the bottom indicate the metadata values computed in each stage; the value in the blue box is loaded by the control plane.

VI. Implementation

We built a prototype of the FlowLens system. Excepting the FMA, which was written in P4, we implemented all other components in Python. The classification engine of the profiler server uses Python's `scikit-learn` [62] and the Weka [28] libraries. We implemented FlowLens's FMA [8] for a Barefoot Tofino ASIC [4] using about 500 lines of $P4_{16}$ code, which was compiled with the P4 Studio Software Development Environment (SDE) [5]. While the FMA's design presented in Section IV-A is generically compatible with PISA architecture, its implementation required careful reasoning due to the specific intricacies of currently available switching hardware.

To implement the FMA code for a Tofino switch, we need to fit the FMA's data structures and operations into the specific pipeline and compute capabilities of the switch. To implement flow marker updates, it would be desirable to compute the flow offset and the bin offset of the target flow marker (see Figure 4) in a single pipeline stage to be able to use all the memory in upcoming stages to store flow markers. However, this cannot be achieved on our target hardware due to three major data dependencies: i) matching (i.e., indexing) the truncation table depends on the quantized packet length, but quantization and truncation are too complex to be realized together in a single stage; ii) indexing a flow marker's bin requires the result of truncation, but the truncation table and the flow table cannot be matched in the same physical stage; iii) matching the flow table and updating the respective flow marker are also too complex to perform in a single stage. Moreover, it is not possible to access all the switch memory from a single stage.

Laying out the FMA in hardware: To accommodate for the above requirements, we split the functioning of FMA across different stages, as depicted in Figure 5. To resolve dependencies i) and ii), we reserve the first and second stages of the pipeline to perform quantization and truncation. Then, we partition the flow table and register grid along the remaining stages to use up all the per-stage stateful registers across the processing pipeline. To overcome the inability to calculate the bin offset and increment the corresponding register cell in the same stage – dependency iii) – the flow table partitions and register grid partitions are placed in contiguous stages. Each flow table partition is responsible for managing flow markers in its corresponding register grid partition.

Figure 5 depicts in detail the operation of FMA when a packet for a new flow arrives. Notation FT_m and RG_m denote partition m of the flow table and register grid, respectively. Assume that the collector has installed a rule for `flow id` ⟨162.2.13.42, 6901, 147.6.54.129, 3478, 17⟩ in FT_2, and that the first incoming packet for this flow has a size of 512B. In stage 1, action `quantization_act` is triggered, quantizing the packet length using QL=5 and setting the resulting quantized packet length (`md.binIndex_quant`) to 16. The object `md` stores the metadata carried over across the pipeline stages.

In the second stage, the truncation table matches against the quantized packet length (refer to Figure 4) and triggers the `truncation_act` action, which returns the bin offset `md.bin_offset` within the flow marker and sets a truncation flag `md.trunc_flag` in order to inform the downstream stages that this packet's corresponding flow marker should be incremented. In case no match exists in the truncation table, the truncation flag is not set, and the packet is not accounted for.

Next, since the flow matching rule is not installed in FT_1, the packet is not matched until it reaches stage 4, where FT_2 is located. Upon matching the `flow id` and verifying that `md.trunc_flag` is set, the `set_flow_data_act2` action is triggered. This action computes `md.rg_cell_offset` by adding `md.bin_offset` and `flow_offset` loaded by the control plane into FT_2. The resulting value is used to index a cell in RG_2 which is then incremented. To index the correct partition of the register grid, we use control flow logic to test which flow table partition was matched, triggering the respective `reg_grid_act2` action that updates the flow marker on the corresponding register grid partition in the next pipeline stage.

Optimizing per-packet computations: To reduce the complexity of the P4 program, we leverage the capabilities offered by the control plane to offload the computation of complex operations from the data plane. This results in a program sampled in Listing 1 which implements four simple actions that can be performed within single stages of the pipeline. Specifically, we offload two operations into the control plane:

a) Computing a flow offset within the register grid: The index of the register grid where a flow marker is located can be easily calculated in the control plane. Since the FMA parameterization is known prior to the loading of the P4 program on the switch, the control plane can compute the number of bins used by a flow marker in a given configuration. Thus, when a new flow is matched, the collector installs a rule where the flow offset is given by the number of flow rules installed in a particular flow table partition times the number of bins composing a marker. Upon matching, the flow offset is passed as an argument to action `set_flow_data_act2` (line 14).

b) Computing a bin offset within a flow marker: To index a bin within a flow marker two values must be added: the flow offset, and the bin offset. While the former can be computed as described in the previous paragraph, the latter is computed by the truncation operator. The quantized packet length passed as an argument to the action responsible for performing truncation (`truncation_act`, line 8) is computed by action `quantization_act` (line 3) using a simple bit shift. Then, the translation between a quantized packet length and the corresponding bin offset can also be computed offline once a specific FMA parameterization is known, and later loaded by the control plane into the truncation table (refer to Section IV-A). Pre-computing these values in the control plane saves stateful memory and pipeline's stages for either monitoring more flows or executing other forwarding behaviors.

```
1  // triggered by the quantization table
2  // bin_width_shift depends on the Quantization Level (QL)
3  action quantization_act(bit<32> bin_width_shift){
4         md.binIndex_quant =
5         (bit<32>) (md.pkt_length >> bin_width_shift);
6  }
7  // triggered by the truncation table
8  action truncation_act(bit<32> new_index, bool flag) {
9         md.bin_offset = new_index;
10        md.trunc_flag = flag;
11 }
12 //triggered by the flow table (FT_2)
13 // compute the offset of the bin in the RG partition
14 action set_flow_data_act2(bit<32> flow_offset) {
15        md.rg_cell_offset = flow_offset + md.bin_offset;
16
17 //triggered after matching the flow table (FT_2)
18 action reg_grid_act2() {
19        bit<16> value;
20        reg_grid2.read(value, md.rg_cell_offset);
21        reg_grid2.write(md.rg_cell_offset, value+1);
22 }
```

Listing 1: P4 code fragment that implements the actions performed per-packet by the FMA. The complexity of the truncation operator and of the computation of flow marker offsets is offloaded to the control plane and the resulting values are loaded through MAUs.

VII. Evaluation

Here, we present our experimental evaluation of FlowLens aimed at analyzing the accuracy of ML-based flow classification tasks and the efficiency of the switch resources usage.

A. Metrics and Methodology

Our experiments aim at identifying a particular class of flows denoted as the *target* class. For instance, when using FlowLens for covert channel detection, the target class can be covert traffic. We assess the quality of FlowLens using the following set of metrics: *accuracy*, i.e., the percentage of flows that were correctly classified in their class, *false positive rate* (FPR), i.e., flows that do not belong to the target class but were erroneously classified as part of the target, and *false negative rate* (FNR), i.e., flows of the target class that were flagged as not belonging to the class. We also resort to related metrics such as *precision* – ratio of the number of relevant flows retrieved to the total number of relevant and irrelevant flows retrieved – and *recall* – ratio of the number of relevant flows retrieved to the total number of relevant flows.

We train our system to be able to identify specific target class flows within the context of three usage scenarios:

Covert channel detection: We train our system to identify Skype flows carrying covert channels encoded by two censorship resistance tools: Facet [38] and DeltaShaper [6]. We train two independent FlowLens applications, for Facet and for DeltaShaper traffic, using a balanced dataset including covert / legitimate samples of recorded flows. The traffic is classified using the XGBoost [7] classifier, based on the packet length distribution of the sampled flows.

Website fingerprinting: We train a second FlowLens application to identify webpages browsed through encrypted tunnels.

Table I. SCALABILITY OF FLOWLENS.

Use Case	FMA Configuration	Marker	Raw Dist.	Scaling
Covert Channels	$\langle QL_{PL}$=4, Top-N=10$\rangle$	20B	3000B	150×
Website Fgpt.	$\langle QL_{PL}$=5$\rangle$	94B	3000B	32×
Botnet Detection	$\langle QL_{PL}$=4, QL_{IPT}=6$\rangle$	302B	10200B	34×

Table II. HARDWARE RESOURCE CONSUMPTION.

Resources	Computational			Memory	
	eMatch xBar	Gateway	VLIW	TCAM	SRAM
Usage	8.46%	5.21%	3.39%	0.00%	38.54%

We leverage the dataset made available by Herrman et al. [29]. This dataset has been widely used for the evaluation of novel website fingerprinting techniques [93, 60], and it contains traces of webpage accesses over OpenSSH. Websites are fingerprinted resorting to the Multinomial Naïve-Bayes classifier [29], which leverages the packet length distribution of the incoming and outgoing data in a connection as features. This classifier also allows us to illustrate how FlowLens can accommodate alternative truncation schemes whenever a given classifier does not return a ranking of feature importance (Section VII-E).

Botnet detection: Our last FlowLens application aims at detecting the presence of botnet chatter among legitimate P2P traffic. We use the dataset produced by Rahbarinia et al. [64], which comprises traffic flows produced by four benign P2P applications (uTorrent, Vuze, Frostwire, and eMule), and two P2P botnets (Waledac and Storm). Malicious flows can be identified by analyzing packet length and inter-packet timing distributions resorting to a Random Forest classifier [64].

We simulate the classification of flows of a given target class in software based on a set of application-specific flow samples. We also configured all the classifiers (Multinomial Naïve-Bayes, XGBoost, and Random Forest) to use the same hyperparameters suggested by the papers we drew our use-cases from. Throughout the evaluation, we assess the performance of different FlowLens configurations while exposing the system to a workload that, to the best of our abilities, mimics those described in the literature. However, we highlight the adoption of a single holdout test instead of the cross-validation approach employed in other representative works [7, 53]. The reason is that, when applying truncation (Section-IV), FlowLens employs a pre-training step to obtain a feature ranking from the classifier. Then, it uses the top-N most important ones to fill the Truncation Table (Figure-4). Since cross-validation returns an average of the results obtained by multiple holdout models trained with different splits of the dataset, the resulting top-N features would not directly translate to be the top-N ones found in a particular model instance, namely in the model to be deployed on the switch for classification. Further, we chose a 50/50 holdout to increase the amount of unseen (test) data and better assess the generalization ability of the classifier.

B. Overall Performance

To give a general insight into the performance of FlowLens, Table I presents the scalability gains of our system when it is used to classify flows for covert channel detection, website fingerprinting, and botnet traffic detection while displaying an accuracy loss of at most 3% when compared with the use of complete packet frequency distributions. For these experiments, we generated the possible combinations of flow markers for the three considered use case scenarios, and assessed whether they allow for accurate flow classification despite their compact size. Packet lengths (PL) vary from 1 to 1500 bytes (MTU), and each cell of a flow marker has a size of 2 bytes.

These results show that, when the quantization and truncation parameters are properly fine-tuned (i.e., QL and truncation table), FlowLens can monitor at least 32 times more flows when compared to the baseline setup without compression, i.e., QL=0 and truncation disabled. Our system can also reach a 150 fold increase in its monitoring capacity when detecting covert channels. This is achieved for QL=4 and by selecting the top-10 most relevant bins for truncation. In this case, with a flow marker as small as 20 bytes, FlowLens manages to achieve a classification accuracy of 93%, only 3% shorter than the result obtained using raw packet length distributions. For website fingerprinting, the flow marker is larger (94 bytes) because we face a multi-class classification problem – different websites are better classified resorting to different bins. Thus, the truncation table is configured to map all quantized packet lengths. Lastly, for botnet chatter detection, we combine the quantization of packet inter-arrival time distribution (IPT) with the PL distribution. In this case, we achieve a marker size of 302 bytes which enables the bookkeeping of $>30\times$ flows.

In general, the absolute number of flows that FlowLens can handle ultimately depends on the switches' available SRAM. The NDA we have signed with Tofino prevents us from disclosing the amount of switch memory but other sources [51] reveal that current switches feature hundreds of MBs of SRAM.

C. Hardware Resource Efficiency

To evaluate the efficiency of FlowLens's hardware resource usage on the switch, we focus independently on the data plane and on the control plane. As for the data plane, Table II shows the average hardware resource consumption of FlowLens across all stages of the switch. The table shows that besides the SRAM required for the tables and register, the consumption of other resources is negligible. Since our flow matching logic entirely relies on exact matching, the FMA's flow table does not consume any of the TCAM resources on the switch. In tandem with the deployment of flow tables in SRAM, FlowLens leaves over 60% of SRAM available. Overall, these results suggest that FlowLens makes enough room for the concurrent execution of many other common forwarding behaviors, like access control, rate limiting or encapsulation, that do not necessarily require an extensive use of the stateful memory in the switch pipeline.

On the control plane, the switch has sufficient resources to fit all models used by FlowLens and to readily classify flows. In particular, the botnet chatter detection is our largest model, occupying only 140MB of memory, and 5.6MB of storage when compressed. In contrast, the model for covert channel detection uses only 64KB of memory and 24KB of storage. All these models comfortably fit within the control plane hardware resources, which has 32GB available RAM. Additionally, the flow classification step is very fast in all cases. For covert channel detection, once the flow markers have been collected from the data plane, the median of the time it takes for the classifier to output a label for a sample flow ranges approximately from 100 to 200 microseconds on the switch's

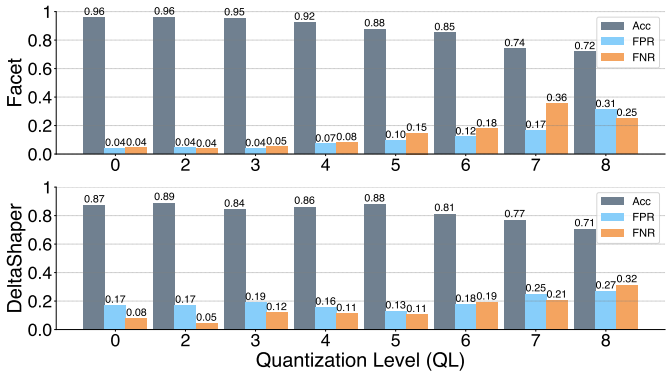

Figure 6. Accuracy, FPR, and FNR for multimedia protocol tunneling detection when using quantized packet length distributions.

Table III. FLOW MARKER SIZE FOR DIFFERENT QUANTIZATION LEVELS.

Bins and Memory	**Quantization Level (QL)**							
`len(1 bin) = 2 Bytes`	0	2	3	4	5	6	7	8
Number of Bins	1500	375	188	94	47	24	12	6
Memory per Flow(B)	3000	750	376	188	94	48	24	12

Intel Broadwell 8-core general-purpose CPU operating at 2 GHz. These results indicate that flow classification can be efficiently conducted on the switch control plane.

Next, we present a set of micro-benchmarks which allow us to assess the benefits of our flow marker generation scheme. We will see that flow marker size (hence memory efficiency) tends to be more sensitive than classification accuracy to small variations in the quantization. The trend is the inverse for truncation, where accuracy is more sensitive to small variations than flow marker size. Our optimizer helps to find sweet-spot setups on the Pareto curve (as explained in Section VII-H).

D. Effects of Quantization

To study the effects of FlowLens's compression schemes, we first focus on the generation of flow markers for packet length distributions and start by analyzing the trade-offs of quantization. We present our main findings:

1. Multimedia covert channels can be detected with up to 92% accuracy using 188-byte flow markers. We leverage XGBoost to classify covert channels [7]. Figure 6 shows how the absolute values obtained for the accuracy, FPR, and FNR of the classifier vary when identifying Facet and DeltaShaper covert channels for different quantization levels (QL). For instance, for quantization level QL=4 FlowLens can correctly identify Facet and DeltaShaper flows with less than 5% and 1% decrease in accuracy, respectively. Table III shows that, for QL=4, a flow marker can be represented in 94 bins instead of a full distribution composed of 1500 bins, amounting to an order of magnitude memory savings. While a single flow marker is then represented using 188B instead of 3000B, DeltaShaper classification scores are maintained with respect to those obtained when using full information (see Figure 6).

2. Accuracy of website fingerprinting is maintained when compressing flow markers by two orders of magnitude. For assessing the quality of FlowLens on website fingerprinting, we use the Multinomial Naïve-Bayes classifier [29]. We reproduced the multiclass closed-world website fingerprinting task for

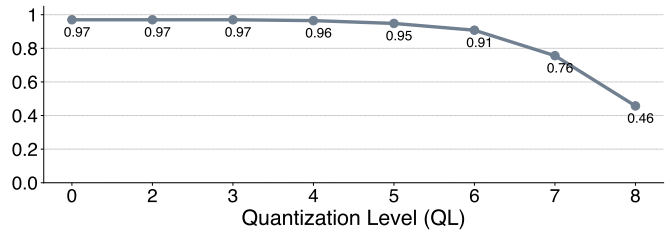

Figure 7. Accuracy results for website fingerprinting when using quantized packet length distributions.

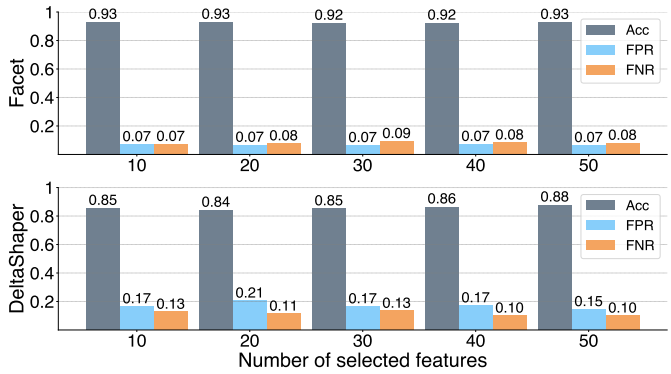

Figure 8. Accuracy, FPR, and FNR for covert channel detection with an increasing number of features for quantization level QL=4.

different quantization levels. Figure 7 shows that FlowLens is able to maintain the same classification accuracy up to a quantization level QL=3. Providing that classification accuracy can be relaxed in favor of memory savings, quantization can be further increased to QL=6, while still achieving over 90% accuracy and reducing a flow marker's memory footprint by two orders of magnitude.

3. Very coarse-grained flow markers are unsuitable for performing traffic differentiation. Figure 7 shows that flow markers can only be compressed to a given factor before causing a steep decrease in the quality of the models' predictions. For instance, in Figure 6, it is possible to observe that for QL=7 the accuracy of the classifier is already over 20% and 10% away from the result obtained with full information for Facet and DeltaShaper, respectively. Thus, it is imperative to find the correct balance between memory savings and accuracy. FlowLens balances this trade-off, for different use cases, through a parameterization during the profiling phase.

E. Effects of Truncation

The second mechanism to generate compact flow markers is that of truncating the flow marker to a subset of bins which make up for the most relevant features leveraged by the classifier. This is illustrated next, as we highlight our main findings after applying tailored truncation in different use cases.

1. Accurate detection of covert channels can be achieved using a flow marker of just 20 bytes. We elaborated a tailored truncation approach based on the importance of features computed by XGBoost. Figure 8 depicts the results obtained when performing quantization with QL=4 and truncating to the top-N most important features. The accuracy, FPR, and FNR rate of the classifier are practically identical when using the

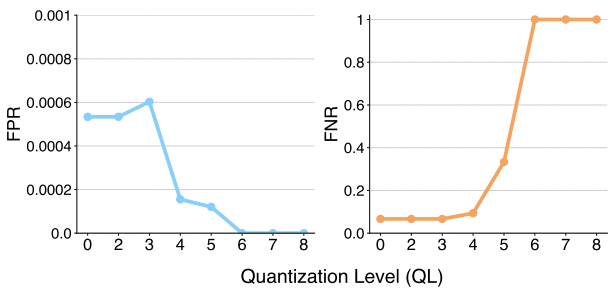

Figure 9. FPR and FNR for `www.amazon.com` at different quantization levels (QL). FNR shows the probability of identifying `www.amazon.com` as some other website. FPR shows the probability of some other website being mistakenly classified as `www.amazon.com`. Truncation is applied at each QL.

Table IV. NUMBER OF BINS USED IN `www.amazon.com` TRUNCATION.

Bins and Memory	**Quantization Level (QL)**							
`len(1 bin) = 2 Bytes`	0	2	3	4	5	6	7	8
Total Number of Bins	1500	375	188	94	47	24	12	6
Bins After Truncation	159	159	156	87	46	23	12	6

top-10 and top-50 features to classify flows (e.g., a difference of only 1% in FNR for Facet flows), and very similar to the results obtained when using full information (refer to QL=4 in Figure 6). Thus, truncation can not only maintain high accuracy, but further reduce the flow marker footprint from 188B (QL=4) to just 20B (QL=4, top-N=10).

2. 20-byte flow markers enable tracking 150× more flows. Covert flow markers can be reduced to just 20B using truncation. This corresponds to a 150× space-saving when representing a flow (from 3000B to 20B). The space freed by compressing a single flow represents an increase in FlowLens's measurement capacity by two orders of magnitude.

3. Fingerprinting accesses to a website yields good results even when feature ranking is unavailable. The truncation method employed for covert channel detection is only applicable when considering classifiers able to output feature importance. To overcome the fact that Herrmann et al.'s [29] classifier is unable to output a rank of feature importance, we perform manual bin selection aimed at identifying a single website, e.g., `www.amazon.com`. Essentially, we first take a collection of access traces performed over a period of time to that particular website. Then, we simply discard the bins that correspond to packet lengths which have had zero counts of the sampled flows. Based on this selection, we then train our classifier accordingly. We can see in Figure 9 that the results obtained using this approach remain competitive. For instance, with quantization level QL=4, flows can be correctly identified with a 0.016% FPR and 9.333% FNR. As shown in Table IV, this flow marker footprint is not as small as with covert channel detection. Yet, it is practical to fingerprint website accesses with QL=4, yielding flow markers with a compression ratio of 1500:87, i.e., 17.2×.

F. Measuring Inter-Packet Timing

In this section, we concentrate on the ability of FlowLens to perform tasks that require both the inspection of packets' inter-arrival (IPT) and length (PL) distributions. To this end, we evaluate FlowLens in detecting P2P botnet chatter. Since the network traffic produced by bots tends to be stealthy and spread

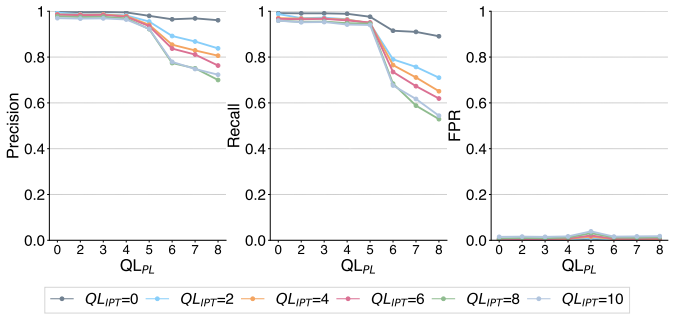

Figure 10. Precision, recall, and FPR for malicious P2P traffic.

across time, packets sent in bot conversations are expected to have a higher IPT than those of legitimate P2P conversations. A conversation consists of the set of flows between any two hosts within a given time window, called *flowgap*. We resort to the Random Forest classifier originally employed by Narang et al. in PeerShark [53], and follow their recommendation to set flowgap to 3600s. Since the largest flowgap is set to 3600s, we vary the quantization of inter-arrival time down to a minimum of 4 bins ($QL_{IPT} = 10$).

Figure 10 depicts the precision, recall, and FPR obtained by the classifier when identifying botnet chatter for different QL applied to both PL and IPT distributions. While $QL_{PL} \leq 4$ slightly degrades precision and recall, we observe a sharp drop in both metrics when $QL_{PL} > 4$. The FPR, however, is not significantly affected by increasing quantization levels. Our experiments also reveal that precision and recall in the identification of legitimate P2P traffic are largely unaffected by the effect of quantization, whereas FPR takes a sharp increase for $QL_{PL} \geq 6$ (20% at $QL_{PL} = 6$ up to 50% at $QL_{PL} = 8$).

This figure also shows that it is possible to accurately identify botnet traffic with compact flow markers. For instance, $\langle QL_{PL} = 4, QL_{IPT} = 6 \rangle$ achieves a recall of 0.96, only 3% worse when compared to the result obtained when using full information (0.99). This accounts for a memory saving of 16× when storing a flow's packet length distribution, as well as occupying just 57 buckets×2B = 114B to keep an inter-packet timing distribution. These results suggest that FlowLens can offer different space-saving/accuracy trade-offs.

G. Performance of Automatic Profiling

To evaluate FlowLens's automatic profiling mechanism, we explore the parameter search space for each use case. For Facet and DeltaShaper, the search space includes the quantization and truncation parameters studied above (48 configurations). For website fingerprinting, the search space corresponds to 8 quantization configurations. For botnet detection, we consider 40 possible configurations based on packet length and IPT quantization, as we refrain from considering those whose $QL_{IPT} = 0$. We configure the optimizer to explore i=10 configurations for covert channel and botnet detection, and i=4 for website fingerprinting. For simplicity, we use no initial sampling for bootstrapping the optimizer, but techniques like Latin Hypercube sampling [75] may be also plugged in.

Fully automatic mode: Table V depicts the results obtained by our automatic profiler to choose an FMA configuration. In all cases, the profiler chooses a configuration that, albeit not the best accuracy wise, still provides a competitive accuracy

Table V. RESULTS OF THE PROFILING PROCEDURE, INCLUDING THE CONFIGURATION OUTPUT BY THE OPTIMIZER AND THE TOP-3 EXPLORED CONFIGURATIONS (LISTED BY DECREASING ACCURACY, EXCEPT FOR THE CASE OF BOTNETS WHICH CORRESPONDS TO MALICIOUS TRAFFIC RECALL).

Config. Rank	Facet (i=10)	DeltaShaper (i=10)	Website Fingerprinting (i=4)	Botnets (i=10)
#1	⟨QL=2, Top-N=all⟩ = 0.960	⟨QL=5, Top-N=all⟩ = 0.880	⟨QL=0⟩ = 0.970	⟨QL_{PL}=2, QL_{IPT}=2⟩ = 0.970
#2	⟨QL=3, Top-N=50⟩ = 0.951	⟨QL=0, Top-N=all⟩ = 0.873	⟨QL=4⟩ = 0.965	⟨QL_{PL}=0, QL_{IPT}=6⟩ = 0.969
#3	⟨QL=0, Top-N=30⟩ = 0.947	⟨QL=0, Top-N=20⟩ = 0.870	⟨QL=5⟩ = 0.948	⟨QL_{PL}=4, QL_{IPT}=6⟩ = 0.960
Output	⟨QL=3, Top-N=10⟩ = 0.944	⟨QL=5, Top-N=10⟩ = 0.840	⟨QL=4⟩ = 0.965	⟨QL_{PL}=3, QL_{IPT}=1024⟩ = 0.953

while generating compact flow markers. For instance, for Facet, the top-3 configurations exhibit a marker size of 375, 50, and 30 bins, respectively. Our profiler chooses a configuration that provides a marker size of 10 bins while achieving an accuracy only 1.6% worse than the configuration with the best-found accuracy (and with a 37× smaller marker). Our reward policy leads the optimizer to perform good decisions over the explored configurations. In website fingerprinting, the profiler outputs the top-2 configuration rather than top-1 since the latter's marker size is too big in comparison (1500 vs 94 bins). The profiler also refrains from choosing top-3, a configuration whose marker is 2× smaller but less accurate. This trend can be observed for the remaining use cases.

Smaller marker for target accuracy: FlowLens can find a configuration that exceeds a minimum accuracy threshold, and that provides the smallest marker. For instance, we set a target accuracy of 0.85 for a DeltaShaper configuration. Among the 10 experimented configurations, the optimizer has found 3 candidate configurations with an accuracy larger than the set threshold. The system output ⟨QL=4, Top-N=30⟩ = 0.850, albeit finding ⟨QL=5, Top-N=40⟩ = 0.876 or ⟨QL=0, Top-N=40⟩ = 0.880, two other configurations which produced larger accuracy at the expense of a larger marker.

Best accuracy given a size constraint: The system is also able to find configurations with a larger accuracy value, given a maximum marker size. Additionally, and since the size of a marker can be computed offline without first trying a configuration, we achieve a reduction in the search space. In the case of DeltaShaper, setting a maximum marker size equal to 30 enables the reduction of the search space from 48 to 21 possible configurations. In this case, the optimizer outputs ⟨QL=2, Top-N=30⟩ = 0.890, albeit finding other smaller but less accurate alternatives such as ⟨QL=3, Top-N=20⟩ = 0.850.

H. Comparison with Related Approaches

In this section, we compare FlowLens against two related approaches: i) techniques which are able to produce compressed representations of packet distributions, and ii) techniques for collection of traffic features resorting to programmable switches.

Alternative feature compression approaches: Online Sketching (OSK) [22] and Compressive Traffic Analysis (CTA) [55] generate compressed packet length/inter-packet timing distributions using linear transformations. However, both approaches depend on matrix multiplications and/or floating-point operations unsupported by current switching hardware. Yet, we compare the classification accuracy of FlowLens against the accuracy obtained by OSK and CTA when using each technique to compress flow representations.

For evaluating the quality of the solutions yielded by the different compression techniques, we leverage the concept of

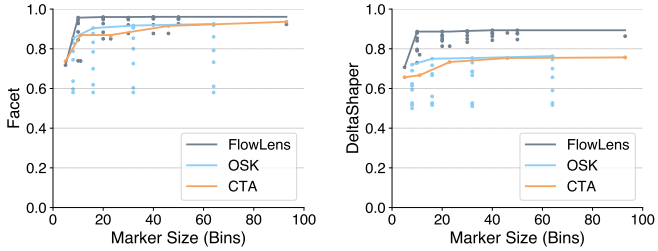

Figure 11. Pareto frontier for covert channel detection when using FlowLens, OSK, and CTA. Dots show individual configurations.

Pareto optimality [57] which allows us to compare possible solutions to multi-criteria optimization problems (flow marker size vs. accuracy, in our case). A solution is said to be Pareto optimal if it cannot be improved in one of the objectives without adversely affecting the other. By generating the set of all of the potentially optimal solutions (Pareto frontier) for each approach, we can observe which approach delivers the best trade-offs between classification accuracy and marker size.

Figure 11 depicts the accuracy obtained in the classification of covert channels when using flow markers (with size up to 100 bins) generated by FlowLens, OSK, and CTA, while using different compression ratios. FlowLens configurations are achieved by combining the different quantization and truncation parameters. Solid lines represent the Pareto frontiers [44] that capture the best configurations for the three approaches. Overall, FlowLens produces flow markers that exhibit a better accuracy/memory trade-off and obtain the most accurate compressed representations of flows. For instance, in DeltaShaper, most FlowLens configurations achieve over 0.80 accuracy (and a maximum of 0.89 using a flow marker with a size of only 10 bins). In comparison, the most accurate OSK marker takes 16 bins and achieves an accuracy of only 0.76. A similar trend occurs in the case of Facet detection.

Alternative feature collection approaches: Systems such as *Flow [74] are able to collect fine-grained packet features at line rate from the switch and offload them to dedicated servers, where the packet distributions can be computed and analyzed by other dedicated systems for specific applications. FlowLens provides a complementary decentralized design where both the collection of packet distribution features and the application-specific analysis (i.e., flow classification) take place on the switches, thus achieving considerable savings in communication, compute, and storage hardware resources.

To estimate the potential gains of our design, we analyze the communication costs of both *Flow and FlowLens. Assuming the existence of 250k concurrent flows where each flow sends 15k packets during a collection window of 30 seconds, *Flow offloads data structures named *grouped packet vectors* (GPVs), each containing a flow key and a list of packet lengths, from a sequence of packets in a flow, on an average of 640ms [74]

which totals 47 evictions. Since each GPV has a fixed header of 24 bytes, assuming 2 bytes to encode a packet length, *Flow must transfer (24B×47 + 2B×15k)×250k = 7.78GB per collection window. This data would then need to be processed on a dedicated server. In contrast, FlowLens only transfers the classification score of each flow at the end of the collection window which involves sending a fixed-size header per flow (13B for flow ID plus a 4B score value) times 250k flows, i.e., ≈4.25MB. Thus, FlowLens exhibits a communication footprint three orders of magnitude smaller than *Flow.

VIII. Security Analysis

We now analyze the security properties of FlowLens when functioning under an adversarial model. The overarching goal of the adversary is to be able to generate flows of a target application class without being detected by FlowLens. We consider three categories of increasingly sophisticated adversaries considering their knowledge about FlowLens and the models employed in ML-based security applications.

1. No knowledge about FlowLens nor the ML model: In the weakest threat model, the attacker knows nothing about the presence of FlowLens in the network infrastructure, nor the details of the models being used by the ML-based security applications leveraging the capabilities of our system. In such a case, as shown in the sections above, ML-based security applications making use of the vanilla FlowLens setup can identify different target classes of traffic with high accuracy.

2. FlowLens-aware adversary: In the second case, we consider an adversary that is aware of the deployment of FlowLens in the network infrastructure, but who is unaware of the particular machine learning models being used to filter the network for particular classes of traffic. In this case, the adversary may attempt to launch two particular types of attacks:

Flow aggregation attacks: An adversary may attempt to evade FlowLens's classifier by misusing the truncation and quantization steps to make the aggregation of flows of a given class of traffic indistinguishable from another class. In this sense, this type of attack is similar to our covert channel scenario (Section VII-D) where the adversary's goal is to mimic the distribution of legitimate traffic and evade a classifier. Figure 6 and Figure 8 show that a finer-grained aggregation of packet distributions does make it harder to evade the classifier. This suggests that increasing flow marker granularity makes FlowLens more robust against flow aggregation attacks.

Evading collection windows: When analyzing long-lived network flows, FlowLens collects flow markers during a maximum pre-defined collection window. Once this window elapses, the FMA located on a given switch stops monitoring flows while the flow markers are read and FMA data structures are reset. An adversary may attempt to exploit this window of opportunity to transmit a class of traffic targeted by FlowLens during this period. However, as mentioned in Section IV-D, FlowLens can tolerate such attacks provided that multiple switches are used in an interleaved fashion to ensure that at least one switch can collect traffic pertaining to flows traversing the network.

DoS attacks: A FlowLens-aware adversary may also attempt to compromise the availability of our system. For instance, it may try to mount a DoS attack based on the transmission of packets with random IP addresses, forcing FlowLens to keep track of multiple dummy flows and waste the switch memory. To mitigate such a threat, FlowLens can temporarily prevent the installation of new rules in the FMA flow table when it detects unusual bursts of traffic, or reconfigure FMA parameters on-the-fly to store smaller (yet less accurate) flow signatures so as to increase the number of measured flows.

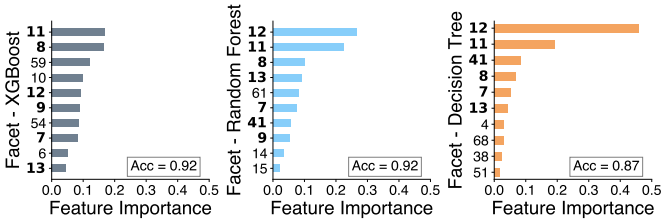

Figure 12. Features (PL bins) collected by the FMA for the ⟨QL=4, top-N=10⟩ configuration, when considering different classifiers to identify Facet covert channels. Features in bold are shared among at least two classifiers.

3. FlowLens and ML model-aware adversary: The third adversary we consider is cognizant of the operation of FlowLens and knowledgeable about the ML model used by a given ML-based application. Apart from an adversary's attempts to evade or compromise the availability of our system, such an adversary aims to leverage adversarial ML techniques [3] to subvert the correct behavior of the model. These attacks can be grouped in two main categories [3]: i) *training-time*, where an attacker aims at manipulating the training set used by the ML model through the insertion of specific samples that alter the decision boundaries of the classifier; ii) *test-time*, where an attacker aims to evade classifiers by crafting traffic samples in such a way that these fool the classifier during its operational phase.

In general, providing defenses to such attacks is orthogonal to FlowLens's ability to collect flow markers and it concerns the particular models used by the different ML-based security applications. Nevertheless, FlowLens is compatible with various techniques aimed at increasing the robustness of the models used for traffic analysis. For mitigating *training-time* attacks, FlowLens's profiling phase can incorporate mechanisms aimed at filtering out contaminated instances upon training [45, 83, 25, 67]. Alternatively, FlowLens operators can leverage recent models whose training is explicitly hardened against the introduction of adversarial samples [16, 15, 78, 34]. For tackling *execution-time* attacks, FlowLens is compatible with the use of several techniques that increase the difficulty of an adversary to successfully evade network traffic classifiers. For instance, FlowLens can leverage classifier ensembles [1, 11, 42] or randomize the classifiers deployed at test-time [50].

Hardening FlowLens against adversarial ML attacks: We performed a simple experiment to understand whether FlowLens can leverage the above techniques to improve its robustness to adversarial attacks, while still collecting flow markers of small size. To this end, we profiled three different classifiers – XGBoost, Random Forest, and Decision Tree [7] – to identify Facet traffic in a ⟨QL=4, top-N=10⟩ FMA configuration.

Figure 12 depicts the importance of the top 10 features selected by the different classifiers after FlowLens's profiling step. Recall that, for a QL=4, there is a total of 94 features (bins), from which only the top-10 is considered. We draw two main observations from this figure. First, since all three classifiers share several features (marked in bold), crafting the

traffic to subvert a given feature (e.g., feature 12), requires extra effort to collectively assess how it affects the classification accuracy not just of a single, but of all three classifiers. Second, each classifier selects a subset of features that are exclusive to it. Thus, while an adversary may shape a given flow to respect the features analyzed by a particular classifier, there may be another classifier that considers a different set of features. For instance, XGBoost leverages bins 59, 10, 54, and 6 to better inform a prediction, while the Random Forest classifier ignores these features and includes 61, 14, 15 in its top-10 instead.

To run multiple classifiers in execution-time, FlowLens must collect a superset of all meaningful features required by each model. Thus, it is expected that flow markers will increase their size. Figure 12 suggests that randomization, i.e., the random selection of one possible classifier, can provide a good compromise between robustness to adversarial ML and flow marker size. While a flow marker consists of 10 features for a given classifier (amounting to 20B), a flow marker that enables FlowLens to choose from three different models to classify flows uses a total of 18 distinct features, producing a flow marker amounting to just 36B. In a similar fashion, FlowLens could use all three classifiers to produce an ensemble which will ultimately classify a flow by majority voting [11].

Performance impact of the defense mechanisms: Although we have not empirically assessed the performance overheads caused by the proposed defense mechanisms, we argue that these mechanisms should not significantly impair the performance of FlowLens. Nevertheless, we reckon that they may require additional resources. The impact on resource allocation could be estimated, e.g., by measuring the memory consumed using ensembles, or by studying how many switches would suffice to plummet the risks of window evasion attacks.

IX. Related Work

There is a considerable body of work proposing approaches for building efficient network telemetry systems for large scale networks [87]. Programmable switches can leverage TCAM-based flow tables for keeping flow data [80, 76, 52] and wildcard rules [13] to record a few statistics about a given flow [46]. The major drawback of this technique is tied to the limited size of TCAM which prevents the bookkeeping of more than a few thousand flows [88]. While multiple flows can be combined in the same table entry [91, 52], this aggregation jeopardizes the accurate representation of a large number of flows [88].

Traffic sampling techniques enable the collection of statistics for a large set of flows by recording a small number of packets of each flow [17, 56, 23]. Examples of general monitoring systems implementing sampling are OpenSample [77] or Planck [65]. Canini et al. [17] introduced a per-flow measurement technique that holds on the partial sampling of flows. However, the accuracy of sampling techniques is usually reduced when one aims at obtaining a faithful representation of a flow's distribution [39]. Moreover, increasing the sample rate is at odds with a larger memory footprint which can impact the overall performance of the network [30].

Probabilistic data structures known as sketches enable the error-bounded representation of flows' statistics within restrictive memory limits [88]. While multiple sketches allow for the extraction of flow's coarse-grained features [21, 86, 39, 43, 31, 85, 32], their strive for generality prevents recording fine-grained information such as approximations of flows' packet lengths and timing distributions. NetWarden [82] uses sketches to record approximate distributions of inter-packet timing distributions for a specific security task. In contrast, FlowLens can be broadly applicable to a range of ML-based applications. Coskun et al. [22] and Nasr et al. [55] explore additional ways to compress packet distributions based on the use of linear projections. Unfortunately, such techniques cannot be implemented efficiently in current switching hardware.

Recent systems relying on network query refinement [54], such as Turboflow [73] and *Flow [74], allow the data plane to offload simple packet features to servers for aggregation and processing. However, differently from FlowLens, such a strategy may increase the risks of network congestion and introduce scalability bottlenecks in large networks [41, 87].

X. Conclusions

This work proposed FlowLens, the first traffic analysis system for ML-based security applications that collects and analyses compact representations of flows' packet distributions – flow markers – within programmable switches. We evaluated our system for three use cases comprising the detection of network covert channels, website fingerprinting, and botnet chatter detection. FlowLens can accurately predict these classes of traffic flows with the help of compact flow markers, allowing for a reduction between one to two orders of magnitude of the memory footprint to represent packet distributions.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) via the SFRH/BD/136967/2018 grant, and the PTDC/EEI-COM/29271/2017, UIDB/50021/2020, and PTDC/CCI-INF/30340/2017 (uPVN) projects.

References

[1] J. Aiken and S. Scott-Hayward, "Investigating adversarial attacks against network intrusion detection systems in sdns," in *Proceedings of the Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Dallas, TX, USA, November 2019, pp. 1–7.

[2] B. Anderson and D. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, Vienna, Austria, October 2016, pp. 35–46.

[3] G. Apruzzese, M. Colajanni, L. Ferretti, and M. Marchetti, "Addressing adversarial attacks against security systems based on machine learning," in *2019 11th International Conference on Cyber Conflict*, vol. 900, Tallinn, Estonia, May 2019, pp. 1–18.

[4] Barefoot Networks, Tofino Switch https://www.barefootnetworks.com/products/brief-tofino/, accessed: 2021-01-05.

[5] ——, P4 Studio, https://www.barefootnetworks.com/products/brief-p4-studio/, accessed: 2021-01-05.

[6] D. Barradas, N. Santos, and L. Rodrigues, "Deltashaper: Enabling unobservable censorship-resistant TCP tunneling over videoconferencing streams," in *Proceedings on Privacy Enhancing Technologies*, Minneapolis, MN, USA, July 2017, pp. 5–22.

[7] ——, "Effective detection of multimedia protocol tunneling using machine learning," in *Proceedings of the 27th USENIX Security Symposium*, Baltimore, MD, USA, August 2018, pp. 169–185.

[8] D. Barradas and S. Signorello, "FlowLens code repository," https://github.com/dmbb/FlowLens, 2020, accessed: 2021-01-05.

[9] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

[10] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, GA, USA, June 2013, pp. 115–123.

[11] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.

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

[13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[14] Broadcom, Tomahawk II 6.4Tbps Ethernet Switch, https://www.broadcom.com/news/product-releases/broadcom-first-to-deliver-64-ports-of-100ge-with-tomahawk-ii-ethernet-switch, accessed: 2021-01-05.

[15] S. Calzavara, C. Lucchese, and G. Tolomei, "Adversarial training of gradient-boosted decision trees," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, Beijing, China, November 2019, pp. 2429–2432.

[16] S. Calzavara, C. Lucchese, G. Tolomei, S. Abebe, and S. Orlando, "Treant: training evasion-aware decision trees," *Data Mining and Knowledge Discovery*, pp. 1–31, 2020.

[17] M. Canini, D. Fay, D. Miller, A. Moore, and R. Bolla, "Per flow packet sampling for high-speed network monitoring," in *Proceedings of the First IEEE International Communication Systems and Networks and Workshops*, Chennai, India, December 2009, pp. 1–10.

[18] C. Cascaval and D. Daly, P4 Architectures, https://p4.org/assets/p4-ws-2017-p4-architectures.pdf, accessed: 2021-01-05.

[19] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, and T. Edsall, "drmt: Disaggregated programmable switching," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Los Angeles, CA, USA, August 2017, pp. 1–14.

[20] Cisco, Cisco Encrypted Traffic Analytics Whitepaper, https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf, accessed: 2021-01-05.

[21] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," *IEEE/ACM Transactions on Networking*, vol. 13, no. 6, pp. 1219–1232, 2005.

[22] B. Coskun and N. Memon, "Online sketching of network flows for real-time stepping-stone detection," in *Proceedings of the IEEE Annual Computer Security Applications Conference*, Honolulu, HI, USA, December 2009, pp. 473–483.

[23] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," in *Proceedings of the SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 325–336.

[24] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[25] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.

[26] J. Gu, J. Wang, Z. Yu, and K. Shen, "Walls have ears: Traffic-based side-channel attack in video streaming," in *Proceedings of the IEEE Conference on Computer Communications*, Honolulu, HI, USA, April 2018, pp. 1538–1546.

[27] R. Habeeb, F. Nasaruddin, A. Gani, I. Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A survey," *International Journal of Information Management*, vol. 45, pp. 289–307, 2019.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[29] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the ACM Workshop on Cloud Computing Security*, Chicago, IL, USA, November 2009, pp. 31–42.

[30] Q. Huang, X. Jin, P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, August 2017, pp. 113–126.

[31] Q. Huang, P. Lee, and Y. Bao, "Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Budapest, Hungary, August 2018, pp. 576–590.

[32] N. Ivkin, Z. Yu, V. Braverman, and X. Jin, "Qpipe: Quantiles sketch fully in the data plane," in *Proceedings*

of the 15th International Conference on Emerging Networking Experiments And Technologies, Orlando, FL, USA, December 2019, pp. 285–291.

[33] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable in-network security for context-aware BYOD policies," in *Proceedings of the 29th USENIX Security Symposium*, Virtual Event, USA, August 2020, pp. 595–612.

[34] A. Kantchelian, J. Tygar, and A. Joseph, "Evasion and hardening of tree ensemble classifiers," in *Proceedings of the 33rd International Conference on Machine Learning,*, vol. 48, New York, NY, USA, June 2016, pp. 2387–2396.

[35] S. Khattak, N. Ramay, K. Khan, A. Syed, and S. Khayam, "A taxonomy of botnet behavior, detection, and defense," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 898–924, 2013.

[36] J. Kučera, D. A. Popescu, H. Wang, A. Moore, J. Kořenek, and G. Antichi, "Enabling event-triggered data plane monitoring," in *Proceedings of the Symposium on SDN Research*, San Jose, CA, USA, March 2020, pp. 14—26.

[37] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan, "Nethcf: Enabling line-rate and adaptive spoofed ip traffic filtering," in *Proceedings of the 27th IEEE international conference on network protocols*, Chicago, IL, USA, October 2019, pp. 1–12.

[38] S. Li, M. Schliep, and N. Hopper, "Facet: Streaming over videoconferencing for censorship circumvention," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, Scottsdale, AZ, USA, November 2014, pp. 163–172.

[39] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation*, Santa Clara, CA, USA, March 2016, pp. 311–324.

[40] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM conference on Computer and Communications Security*, Alexandria, VA, USA, October 2006, pp. 255–263.

[41] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "Netview: Towards on-demand network-wide telemetry in the data center," *Computer Networks*, vol. 180, 2020.

[42] L. Liu, W. Wei, K.-H. Chow, M. Loper, E. Gursoy, S. Truex, and Y. Wu, "Deep neural network ensembles against deception: Ensemble diversity, accuracy and robustness," in *Proceedings of the 16th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Monterey, CA, USA, November 2019, pp. 274–282.

[43] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communications Conference*, Florianópolis, Brazil, August 2016, pp. 101–114.

[44] D. Luc, *Pareto Optimality*. Springer New York, 2008, pp. 481–515.

[45] S. Ma and Y. Liu, "Nic: Detecting adversarial samples with neural network invariant checking," in *Proceedings of the 26th Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2019.

[46] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)," in *Proceedings of the IEEE Conference on Computer Communications*, Toronto, Canada, April 2014, pp. 934–942.

[47] R. McPherson, A. Houmansadr, and V. Shmatikov, "CovertCast: Using live streaming to evade internet censorship," *Proceedings on Privacy Enhancing Technologies*, vol. 2016(3), pp. 212–225, 2016.

[48] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[49] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "Nethide: Secure and practical network topology obfuscation," in *Proceedings of the 27th USENIX Security Symposium*, Baltimore, MD, USA, August 2018, pp. 693–709.

[50] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the ACM SIGSAC conference on computer and communications security*, Dallas, TX, USA, October 2017, pp. 135–147.

[51] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Los Angeles, CA, USA, August 2017, pp. 15–28.

[52] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 419–430, 2015.

[53] P. Narang, S. Ray, C. Hota, and V. Venkatakrishnan, "Peershark: detecting peer-to-peer botnets by tracking conversations," in *Proceedings of the IEEE Security and Privacy Workshops*, San Jose, CA, USA, May 2014, pp. 108–115.

[54] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Los Angeles, CA, USA, August 2017, pp. 85–98.

[55] M. Nasr, A. Houmansadr, and A. Mazumdar, "Compressive traffic analysis: A new paradigm for scalable traffic analysis," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Dallas, TX, USA, October 2017, pp. 2053–2069.

[56] Netflow, https://www.ietf.org/rfc/rfc3954.txt, accessed: 2021-01-05.

[57] P. Ngatchou, A. Zarei, and A. El-Sharkawi, "Pareto multi objective optimization," in *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*, Arlington, VA, USA, January 2005, pp. 84–91.

[58] T. Nguyen and G. Armitage, "A survey of techniques for

internet traffic classification using machine learning." *IEEE Communications Surveys and Tutorials*, vol. 10, no. 1-4, pp. 56–76, 2008.

[59] T. O'Connor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "Homesnitch: Behavior transparency and control for smart home iot devices," in *Proceedings of the 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, Miami, FL, USA, May 2019, pp. 128—-138.

[60] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, Chicago, IL, USA, October 2011, pp. 103–114.

[61] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "Netbricks: Taking the V out of {NFV}," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah, GA, USA, November 2016, pp. 203–216.

[62] 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 E. Duchesnay, "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] C. Putman, Abhishta, and L. Nieuwenhuis, "Business model of a botnet," in *Proceedings of the 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Cambridge, UK, March 2018, pp. 441–445.

[64] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "Peerrush: Mining for unwanted p2p traffic," *Journal of Information Security and Applications*, vol. 19, no. 3, pp. 194–208, 2014.

[65] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 407–418, 2014.

[66] A. Reed and M. Kranch, "Identifying https-protected netflix videos in real-time," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Scottsdale, AZ, USA, March 2017, pp. 361–368.

[67] K. Roth, Y. Kilcher, and T. Hofmann, "The odds are odd: A statistical test for detecting adversarial examples," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, Long Beach, CA, USA, June 2019.

[68] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *Proceedings of the 26th USENIX Security Symposium*, Vancouver, BC, Canada, August 2017, pp. 1357–1374.

[69] N. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, March 2017, pp. 67–82.

[70] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, "Packet transactions: High-level programming for line-rate switches," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Florianópolis, Brazil, August 2016, pp. 15–28.

[71] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the ACM Symposium on SDN Research*, Santa Clara, CA, USA, April 2017, pp. 164–176.

[72] J. Snoek, H. Larochelle, and R. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, vol. 2, Lake Tahoe, NV, USA, December 2012, pp. 2951–2959.

[73] J. Sonchack, A. Aviv, E. Keller, and J. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proceedings of the Thirteenth EuroSys Conference*, Porto, Portugal, April 2018, pp. 1–16.

[74] J. Sonchack, O. Michel, A. Aviv, E. Keller, and J. Smith, "Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow," in *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, USA, July 2018, pp. 823–835.

[75] M. Stein, "Large sample properties of simulations using latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.

[76] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Flowcover: Low-cost flow monitoring scheme in software defined networks," in *Proceedings of the IEEE Global Communications Conference*, Austin, TX, USA, December 2014, pp. 1956–1961.

[77] J. Suh, T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems*, Madrid, Spain, June 2014.

[78] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *Proceedings of the Sixth International Conference on Learning Representations*, Vancouver, Canada, April 2018.

[79] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Packet-level signatures for smart home device events," in *Proceedings of the 27th Network and Distributed Security Symposium*, San Diego, CA, USA, February 2020.

[80] N. van Adrichem, C. Doerr, and F. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *Proceedings of the IEEE Network Operations and Management Symposium*, Krakow, Poland, May 2014.

[81] C. Wright, S. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Proceedings of the 16th Annual Network & Distributed Security Symposium*, San Diego, CA, USA, February 2009.

[82] J. Xing, Q. Kang, and A. Chen, "Netwarden: Mitigating

network covert channels while preserving performance," in *Proceedings of the 29th USENIX Security Symposium*, Virtual Event, USA, August 2020, pp. 2039–2056.

[83] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Proceedings of the 25th Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2018.

[84] J. Yan and J. Kaur, "Feature selection for website fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 200–219, 2018.

[85] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Budapest, Hungary, August 2018, pp. 561–575.

[86] T. Yang, L. Wang, Y. Shen, M. Shahzad, Q. Huang, X. Jiang, K. Tan, and X. Li, "Empowering sketches with machine learning for network measurements," in *Proceedings of the 2018 ACM SIGCOMM Workshop on Network Meets AI & ML*, Budapest, Hungary, August 2018, pp. 15–20.

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

[88] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "Countmax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2774–2786, December 2018.

[89] F. Zane, G. Narlikar, and A. Basu, "Coolcams: Power-efficient tcams for forwarding engines," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1, San Francisco, CA, USA, March 2003, pp. 42–52.

[90] 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 *Proceedings of the 27th Network and Distributed Security Symposium*, San Diego, CA, USA, February 2020.

[91] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Deploying default paths by joint optimization of flow table and group table in sdns," in *Proceedings of the 25th IEEE International Conference on Network Protocols*, Toronto, ON, Canada, Ocotober 2017, pp. 1–10.

[92] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," in *Proceedings of 11th International Conference on the Design of Reliable Communication Networks*, Kansas City, MO, USA, March 2015, pp. 267–270.

[93] Z. Zhuo, Y. Zhang, Z.-l. Zhang, X. Zhang, and J. Zhang, "Website fingerprinting attack on anonymity networks based on profile hidden markov model," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1081–1095, 2017.