Traffic Classification Based on Zero-Length Packets

Joseph Kampeas, Asaf Cohen, and Omer Gurewitz

***Abstract*—Network traffic classification is fundamental to network management and its performance. However, traditional approaches for traffic classification, which were designed to work on a dedicated hardware at very high line rates, may not function well in a virtual software-based environment. In this paper, we devise a novel fingerprinting technique that can be utilized as a software-based solution which enables machine-learning-based classification of ongoing flows. The suggested scheme is very simple to implement and requires minimal resources, yet attains very high accuracy. Specifically, for TCP flows, we suggest a fingerprint that is based on zero-length packets, hence enables a highly efficient sampling strategy which can be adopted with a single content-addressable memory rule. The suggested fingerprinting scheme is robust to network conditions such as congestion, fragmentation, delay, retransmissions, duplications, and losses and to varying processing capabilities. Hence, its performance is essentially independent of placement and migration issues, and thus yields an attractive solution for virtualized software-based environments. We suggest an analogous fingerprinting scheme for user datagram protocol traffic, which benefits from the same advantages as the TCP one and attains very high accuracy as well. Results show that our scheme correctly classified about 97% of the flows on the dataset tested, even on encrypted data.**

***Index Terms*—Network traffic classification, network monitoring and measurements, machine learning, network function virtualization, software-defined networking.**

I. Introduction

NETWORK traffic classification is the essence of network management and network security functions. The identification process typically relies on dedicated hardware components and complex computational capabilities at the monitoring point. However, such components are typically not available in virtual environments, where various services are running on independent general-purpose hardware, while the resources are shared between various services. Thus, the design of the classification process in a virtual environment is inherently different and challenging, as it needs to overcome the virtualization disadvantages [1].

Manuscript received December 7, 2017; revised March 17, 2018; accepted April 1, 2018. Date of publication April 11, 2018; date of current version September 7, 2018. Part of this study appeared in IEEE International Conference on the Science of Electrical Engineering, 2016. This work was partly supported by the European Union Horizon 2020 Research and Innovation Programme SUPERFLUIDITY, Grant 671566. The associate editor coordinating the review of this paper and approving it for publication was F. De Turck. *(Corresponding author: Joseph Kampeas.)*

The authors are with the Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel (e-mail: kampeas@bgu.ac.il; coasaf@bgu.ac.il; gurewitz@bgu.ac.il).

Digital Object Identifier 10.1109/TNSM.2018.2825881

Identifying applications based on passive observations of individual packets or streams traversing the network, typically relies on two main procedures: *capture* and *classification*.

1) *Capture:* The *capture* procedure samples data packets traversing the network and stores them for further inspection. To obtain a representative sample set for each flow, complex sampling mechanisms that consume *many Content-addressable Memory (CAM) filtering rules* are required [2]–[7]. Even though hardware-based CAM, commonly used in Software-Defined Networking (SDN) switches, is able to search its entire memory space in a single operation, due to its typically narrow table size and lack of support at high rate updates to its ruleset, the resulting capturing capabilities are limited [8]–[11]. Virtual environments, on the other hand, may use software-based CAM (look-up table) that is able to both store a large number of rulesets, and update these rulesets rapidly. Nevertheless, the complexity that is required in order to find a matching rule in a software-based CAM deteriorates the capture performance significantly. Thus, for both cases, minimizing the required filtering ruleset is of great importance [12].

The *traffic classification* process labels traffic based on a predefined set of classes (e.g., applications). A commonly used classification approach is Deep Packet Inspection (DPI), which classifies traffic by inspecting the *payload* of each packet traversing the inspection point. DPI attains remarkably high accuracy for unencrypted traffic. Nonetheless, since DPI relies on the visibility of the payload to the classifier, its effectiveness diminishes with usage of data encryption, which is the conventional wisdom, or when examining the content of the packets traversing the network is forbidden or limited due to privacy or complexity concerns. Consequently, utilizing Machine-Learning (ML) techniques on a relatively limited and easy to extract feature set, which rely on the statistical behavior of each application, overcomes most of the aforementioned hindering factors and is considered as an attractive solution, for both SDN and NFV environments.

2) *Machine Learning-Based Classification:* Typically, an ML algorithm maps flows according to discriminative attributes, then, unknown traffic can be classified, according to the rules that were learnt. The chosen attributes are significant to both classification performance and its complexity. In particular, classifying a large variety of applications may require a large attribute set to attain sufficient accuracy. However, as the size of the attribute set increases, the computational complexity of the classification process increases [13]. To mitigate this issue, a hierarchical approach may be considered, such that at each level, the classifier focuses on a different attribute set [14]–[17].

The classification performance under statistical attribute set (e.g., max, min or average packet size) has been examined with various ML algorithms [18]–[23]. Note that these attributes can be attained by properly sampled packets from each flow. Yet, since such statistics can be determined only after the flows' termination, it is not suitable for policy enforcements or security functions. In the pioneering work of Bernaille *et al.* [24], the authors argued that some of the commonly used features, such as packets inter-arrival time, are very sensitive to the network load, and hence, should not be used to classify traffic flows. Instead, the authors suggested to use the direction and the size of the first packets in a flow to classify flows before termination. This approach attains high accuracy under a variety of classification algorithms, such as unsupervised K-Means clustering, Gaussian Mixture model, Hidden Markov model [25], [26], density-based spatial clustering [27], support vector machines [28], [29], neuro-fuzzy logic [30] and a semi-supervised hybrid approach [31]. Comparisons between traffic classification techniques can be found in [32]–[35]. However, although this approach achieves impressive accuracy while classifying flows online, it requires *maintaining and updating sampling triggers* to capture the first few packets from each flow. Such a task requires high update rate and a large filtering ruleset to maintain an action rule upon every ongoing flow, resulting in expensive CPU and CAM resource consumption. Furthermore, the size of the first few captured packets may be deficient as a discriminator when network conditions vary, since it is sensitive to the network type and parameters (e.g., packets loss, retransmissions, fragmentation, out-of-order arrivals, maximum transmission unit, etc.). Hence, when using the capture and classification unit as a Virtual Network Function (VNF), without knowing a-priori where it will be deployed and under which circumstances, the different actual environments may skew the classification process completely. Thus, when deploying the analysis tool as a VNF, more robust capture and classification is required.

A. Contributions

In this study, we suggest a novel set of features for traffic classification that is applicable to both SDN and NFV environments. In particular, our features rely on an extremely simple sampling strategy, namely, a single filtering rule to capture the required data, and demand only sampling a minimal fraction, yet result in very high classification accuracy. The suggested technique relies on the following key observation: since each application adheres to a proprietary standard message exchange, the data exchanged between two end points of an application should follow a typical pattern which distinctively characterizes the application that generated it. Specifically, when examining a sequence of data sizes, generated by the Application-layer prior to the intervention of the lower layers, exchanged between the entities running the application, one can identify a characteristic pattern which is typical to the associated application and is different from one application to the next. The main observation is that one can utilize these distinct patterns within the network's premises to identify the generating application.

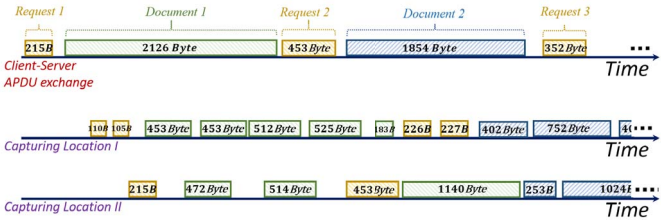


Fig. 1. Illustrated example of data exchanged between two hosts at the application level, compared to this data exchange as observed at network level, at two different locations.

However, since the capture is performed on the network interior, the Application Protocol Data Units (APDUs) described above are highly affected by the network's side effects, such as fragmentation, retransmissions, timing, congestion, packet loss, location, delays, etc. Furthermore, these side effects are inherently different at different networks. Figure 1 illustrates the concept. Specifically, the first line depicts a typical client-server application sequential pattern of request and response APDUs from the Application-layer point of view. In comparison, the next lines depict these APDUs after the network intervention, which may decompose the APDUs in various ways. Thus, examining parameters that are influenced by the network, such as packet size, inter-packet arrival time, etc., that change from one location to another, may not be suitable for a dynamic deployment such as SDN and NFV, and a different set of parameters is required. In NFV environment the traffic classification needs to be as flexible as possible by adjusting the classification complexities and tuning them according to the available resources, in order to achieve the best identification results at run-time. In our context, the flexibility is expressed by the various signature lengths, under which the classifier was designed to operate and maximize the classification accuracy. Thus, our flexible scheme can be virtualized as a network function to achieve the scalability and accuracy required for identifying network flows. Moreover, our scheme is intended to be used as a VNF application, running on a shared commodity server, shifting the classification paradigm from dedicated middlebox hardware to a VNF in shared resources of a server.

The main contributions of this paper are thus as follows. (i) We introduce a simple sampling strategy from which the APDU exchange patterns (signatures) can be restored. Specifically, we extract TCP flow attributes only by sampling *zero-length packets*, i.e., packets that contain control bits, but do not contain any payload (e.g., SYN, ACK, etc.). It is important to note that the zero-length packets are easy to process, and more importantly, they contain critical information on the connection state at the application level. (ii) We suggest an APDU estimation scheme, which enables characterization as well as classification of the generating application, according to the estimated APDU exchange of the participating entities prior to the impact of network side effects. These exchanges are the basis of our unique low-dimension attribute set, which enables us to utilize common ML techniques to classify traffic while the flows are ongoing, with high accuracy for a large variety of applications. Clearly, a specific application can generate several typical patterns, yet each pattern characterizes a

specific application. (iii) To further support our claims, we implemented and tested our scheme in a virtual environment over a large set of traffic traces, comparing our scheme with other common approaches. Even though zero-length packets are frequent in TCP traffic (usually ACK is sent for every two packets), it is important to note that consecutive zero-length packets hold highly correlated information. Thus it is possible to demonstrate robustness by showing that even random sampling of zero-length packets yields high accuracy. This is crucial for both NFV and SDN environments, where resource shortage (e.g., CAM and CPU) is likely to occur.

The rest of the paper is organized as follows. In Section II we provide the preliminaries. In Section III we introduce the sampling and classification scheme. Section IV describes the design, implementation and evaluation of the scheme. The complete classification results are given in the Appendix.

II. Preliminaries

In this section, we provide a brief background, which will be used throughout this paper.

A. Basic Properties of TCP

Even though the Transmission Control Protocol (TCP), which is the prevalent Transport layer protocol for guaranteed in-order data delivery service, is very well known, in this sub-section we provide a brief reminder of some of TCP's features and in particular the Sequence and Acknowledgment numbers (denoted by *seq*# and *ack*#, respectively) that we rely on throughout this paper.

In order to maintain a reliable and ordered connection between two hosts, TCP utilizes positive acknowledgment messages (ACK), timeouts and retransmissions to guarantee error-free delivery. Accordingly, when a data is not received properly (retransmission timer expires before an ACK is received), the data is retransmitted starting at the not yet acknowledged first byte in the flow. In order to maintain this mechanism, TCP allocates *seq*# and *ack*# fields in each TCP segment header. In particular, when data is sent through a TCP connection, these fields indicate which bytes have been sent and correctly received. This is used to ensure that no data is lost on its way to the destination. The receiver utilizes the *seq*# to reorder the segments as needed. The receiver replies with an ACK informing the sender which data segments have been received with no error. Specifically, the *ack*# indicates the next *seq*# the receiver expects to receive from the sender. In this way, both endpoint hosts keep track of the data flow in both directions.

It is important to note that a receiver can *piggyback an ACK on a data packet that it needs to deliver* (i.e., in the forthcoming ACK both the *seq*# and the *ack*# can be increased simultaneously, one informing the receiving host the *seq*# of the sent segment, and the other acknowledging the received data, respectively).

B. Automatic Traffic Classifier

In this paper, we utilize ML and classification algorithms which are already widely used in a variety of applications, including traffic classification. To name a few, [29] uses support vector machines for traffic classification. References [32]–[35] give very good overviews of contemporary works using various ML algorithms for traffic classification as well. These methods mainly rely on two core phases. In the first, the classifier is taught how to map a set of attributes (flow-related in the context of this paper) to a set of (traffic) classes. Then, the classifier applies the rules it has learned to map the data. As the focus of this paper is on building a software-based high-speed traffic classifier, using a novel feature selection process, which creates unique and easy to capture fingerprints from traffic data, and not on the actual classification method, in the sequel we review the J48 decision tree classification method that is used throughout the paper. This method can be efficiently implemented in software. Together with our feature set, this method provides very accurate classification results.

1) J48 Decision Tree [36]: J48 is an open source version of the C4.5 algorithm [37]. In this method, a decision tree is generated from a set of labeled samples. Generally speaking, when building a decision tree, the set of samples is recursively split, one feature after the other. The key concept behind C4.5, is to maximize the *information gain* when deciding which feature will dictate the next branching level in the tree. Specifically, assume each labeled sample consists of a p-dimensional vector $(x_1, x_2, \ldots, x_p)$, where x_j represents the j'th feature (an attribute), and each sample has a label, y_0 or y_1. The algorithm computes, for each feature, the *conditional entropies* given each value of the feature, and subtracts the weighted sum of these entropies from the total entropy of the set. Entropies are calculated on the empirical frequencies of the labels.[1] The result is the information gain for a feature, and the feature which maximizes this gain is chosen for the next level in the tree.

Finally, we note that in order to evaluate the performance, we use the common classification performance metrics, in particular, the *recall and precision* [38, Ch. 5.7].

III. Classification Attributes

In this section, we introduce the sampling and classification scheme which we utilize to classify traffic. The attribute set relies on the observation that data exchange between applications follows a characteristic pattern. This pattern can be utilized to identify the generating application, i.e., different applications generate distinguishable patterns. Let us explain the concept through an illustrated example depicted in Figure 1, which follows the data exchanged (sent and received APDUs) between a client and a server, running an illustrative application. For example, as can be seen in the first time-line of this figure, from the application layer point of view, the APDUs sent from the client to the server (and vice versa) are essentially typical requests and responses that a client may

[1] That is, if under a certain feature value one has n samples with y_0 and m samples with y_1 the relevant entropy is the binary entropy of $(\frac{n}{n+m}, \frac{m}{n+m})$, which is $-\frac{n}{n+m}\log\frac{n}{n+m} - (1-\frac{n}{n+m})\log(1-\frac{n}{n+m})$.

ask and a server may fulfill. The APDUs themselves are generally affected very little by the network characteristics.[2] In particular, since applications anticipate a sufficient amount of data in order to work properly between message exchanges, network delay and packet drops have mostly minor effect on the generated APDUs' size, and hence, this attribute is robust to network effects. Accordingly, throughout this work, we will construct signatures for each application based on the APDU exchange between the two parties, not taking into account the breakups of these APDUs to smaller sizes, i.e., an application signature will be solely based on the APDU exchange sequence, e.g., (215, 2126, 453, 1854, 352) in Figure 1, rather than the sequence (110, 105, 453, 453, 512, ...) in the second timeline or the sequence (215, 472, 514, 453, 1140, ...) in the third. We emphasize that the pattern generated by an application is not necessarily unique (i.e., there can be several signatures to the same application), yet given a pattern (signature), the application that has generated it can be identified with high probability. In other words, the same application can generate more than one typical pattern yet two different applications are not likely to generate similar patterns.

Yet, without any support from the applications, and since the APDUs cannot be constructed directly within the network core (as they are arbitrarily fragmented; see Figure 1), it is challenging to classify on-going flows, in real-time, while keeping the sampling procedure simple and cheap (e.g., only few CAM rules). Moreover, since APDUs are commonly sent over several networks and infrastructures whose capabilities may vary from one hop to another, it is very likely for a traffic flow to look statistically different at two distinct points of observation, due to the underlying network effects (e.g., congestion, packet drop, fragmentation, etc.). For example, a router capturing the IP packets traversing the network, generated according to the example in Figure 1 at capturing location 1, cannot relate between an APDU and the packets that were segmented from it, nor can it identify whether the packets originated from a single or more APDUs. We address this challenge by designing an attribute set which is based on the aggregated chunks of data exchanged between the client and the server, e.g., $\{110+105, 453+453+512+525+183, 226+227, 402+752\}$ from the second time-line in Figure 1, or $\{215, 472+514, 453, 1140, 253+1024\}$ from the third. Our strategy is lightweight and allows both the attribute-set-construction and the online-application-classification within the network core by sampling only a limited amount of traffic.

In the following subsection, we describe our zero-length packet based fingerprint for TCP flows, which enables real time flow classification, based on only a single filtering rule and requiring a limited sampling set.

A. TCP Classification

As previously mentioned, capturing and constructing APDU sequences at the network core is unattainable. Even the task of just capturing APDU sequences without trying to reconstruct them at the network core is quite intricate, especially when considering per flow sampling techniques, which require many CAM entries. As mentioned, the number of required lookups limits the monitoring capabilities significantly, in both NFV and SDN platforms. To address this issue, we introduce APDU estimation scheme that relies on zero-length packets (e.g., SYN, ACK, etc.), which enables to construct approximated APDU fingerprints for each application. Note that zero-length packets can be sampled deterministically by using a single filtering rule (i.e., sample a packet if its payload length equals zero) and are very easy to process. Specifically, from these zero-length packets, we obtain the flows' states, and reconstruct the APDU fingerprint sequence. According to our experiments the resulting dataset comprises only 2-3% of the TCP traffic volume in bytes. Moreover, although these zero-length packets are frequent (approximately, 33% of the TCP packets are zero-length packets), in case of lack of resources, it is possible to capture zero-length packets uniformly at random to reduce the sampling rate, and still attain very high accuracy, as we show in our experiments. This is since the information attained from one zero-length packet is also reflected in other zero-length packets (as long as they are not too far apart).

Let us begin our discussion by examining a TCP connection and the information it holds. Consider a data packet sent by application $side_A$ to application $side_B$ through a TCP connection. As mentioned in Section II, the sequence number carried on this packet indicates the aggregated number of data bytes that $side_A$ sent in this connection so far, whereas the acknowledgment number specifies the number of bytes it correctly received from $side_B$. Note that these bytes can originate from multiple APDUs and there is no mapping between APDU and TCP packet (e.g., a single APDU can be fragmented into multiple TCP packets). In the opposite direction, namely, packets originated by $side_B$, the acknowledgment number indicates the number of bytes $side_B$ received from $side_A$ successfully, while the sequence number determines the aggregated number of data bytes sent by $side_B$. That is, the *seq*# in packets that $side_A$ sends is reflected in *ack*# in packets that $side_B$ sends, and vice versa. Note that the sending station may keep transmitting data that was not confirmed by an ACK. Thus, in order to derive the amount of data that was sent and received by each side, it is sufficient to sample zero-length packets (e.g., ACK) and inspect the progress in the *seq*# and *ack*#.

Since, as previously mentioned, APDU boundaries cannot be determined solely based on TCP packets, we define an accumulated-APDU (a-APDU) of a flow as follows.

Definition 1: The a-APDU of a flow is the number of data bytes traversed to one side (possibly across multiple TCP packets and multiple APDUs), until some data bytes crossed back in the reverse direction.

Typically, after a TCP connection is established, at the protocol negotiation phase, both sides coordinate session parameters by sending iteratively a certain amount of data, until they agree on the session parameters. Since different protocol types require exchanging different sizes of APDUs in the coordination process, it is likely to obtain unique protocol fingerprints by observing these a-APDUs. Accordingly, we define

[2]Some applications may try to identify the hardware/network used (e.g., desktop or mobile, wireless or wired), and change their behavior accordingly. We will treat two such versions as two different fingerprints.

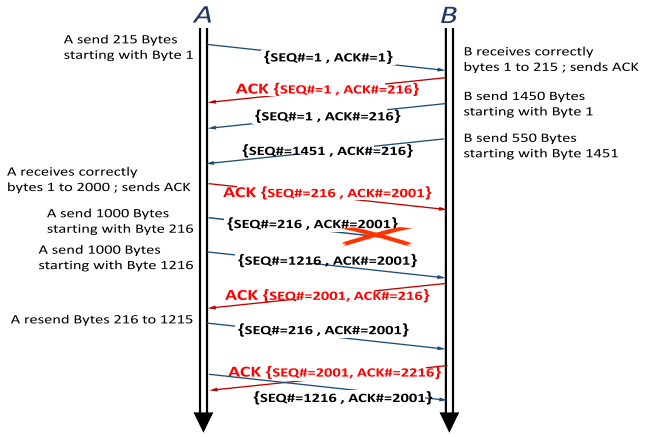


Fig. 2. A Basic TCP protocol illustration, and the fingerprint which can be extracted from the zero-length packets at an intermediate node in the network. In this example, the fingerprint will be {(215, 0); (215, 2000); (2215, 2000) . . .}, where the left number refers to data from A to B and the right number refers to data from B to A.

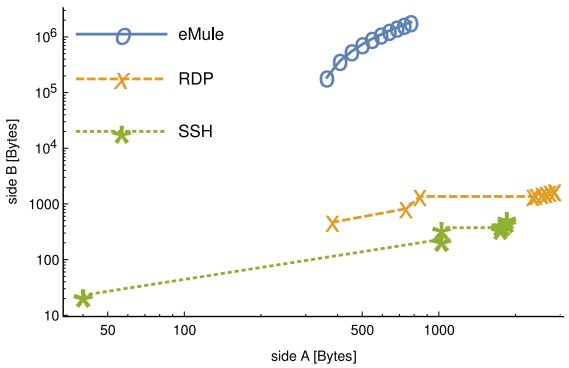


Fig. 3. Ten a-APDU exchanges for three different applications: SSH, RDP and eMule. The axes depict the interlaced number of Bytes sent by A (X-axis) vs. the number of Bytes sent by B (Y-axis) in a flow, on a log-scale.

our application fingerprints as a *series of flow a-APDUs*. This definition disregards the length of the a-APDU series (i.e., the number of a-APDUs in a fingerprint). This is since the number of a-APDUs of a flow is unknown to the classifier a-priori, and further, the classifier is required to label on the fly flows of any a-APDU series length.

Thus, our fingerprint is comprised of a sequence of tuples which describe the number of bytes sent in each direction since the last change in data direction. Note that if the last entry was determined by a change in the number of bytes received in one direction, the next entry will be determined only after a change in the received number of bytes in the reverse direction. For example, in the illustration of the TCP operation in Figure 2, the fingerprint will be {(215, 0); (215, 2000); (2215, 2000), . . .}. Note that since a receiver can piggyback an ACK on a data packet that it sends, and since we capture only zero size packets, both fields in a fingerprint's entry can be different from the fields in the previous entry. It is important to emphasize that we do not assert that the number of bytes in each APDU (all the more so for a-APDU) is fixed. We do claim and will rigorously support it in the evaluation part (Section IV), that the a-APDU pattern can characterize and hence distinguish between different applications.

As previously mentioned, a-APDUs may spread over multiple TCP packets, and the receiver may reply with several ACKs. Furthermore, retransmissions may occur and out of order ACKs may be captured. Nonetheless, the procedure to assemble fingerprints from zero-length messages is quite simple. The intermediate switch (the classifier host) captures all zero-length packets traversing it in all directions. For each traversing flow the classifier maintains an entry in which it reconstructs the a-APDU signature. Reconstruction is simply by accumulating the number of Bytes traversing in one direction before receiving some Bytes traversing in the other direction, for which a new entry in the signature is initiated and the number of bytes in the reverse direction are counted, and so on and so forth. A detailed description including a flowchart is provided in Section IV-A.

In order to illustrate the a-APDU exchange, we provide an example of three representative a-APDU sequences for three ubiquitous applications (SSH, RDP and eMule) as extrapolated by an intermediate node. The representative flows have been arbitrarily extracted from a published dataset, given in [39] (a detailed description of the dataset is provided in Section IV-B) and the a-APDU sequences were extracted according to the description above. Figure 3 depicts ten a-APDUs of these three representative flows, as inferred by the capturing node. The figure depicts the interlaced number of Bytes sent by a sender denoted by A (X-axis) vs. the number of Bytes sent by its associated node denoted by B (Y-axis) on a log-scale, based on captured TCP ACKs. For example, as can be seen in the figure, the difference between the first and second SSH entries indicates that in the first a-APDU A has sent about 1000 Bytes and B has sent about 500 Bytes; the difference between the second and third entries indicates that B has sent about 100 Bytes while A has sent no data (a-APDU (0,100)). Note that as stated earlier, since each fingerprint entry only counts the number of accumulated bytes traversing the network since the last change in direction, i.e., no time-stamps, number of exchanged packets or duplicates, our fingerprints are not sensitive to network noise (e.g., retransmissions, fragmentations, inter-arrival time, losses, etc.).

Even though in the figure we show only a single instance for each application and only for three applications, the figure suggests that each application protocol generates a different a-APDU exchange, thus, can be classified accordingly. In Section IV we show in detail that utilizing the a-APDUs as the attribute set in our classifier design indeed provides very high accuracy, both in terms of Precision and in terms of Recall, for classifying flows traversing the network for many applications.

Obviously, examining the *seq#* and *ack#* of *each and every TCP packet* will result in higher fingerprinting granularity and probably better application characterization. Nonetheless, our method lowers the sampling rate significantly by sampling only messages with no data, which can be statistically sampled to further reduce the sampling rate.

B. Complexity

Let us next examine the complexity and cost of our classification method.

Capturing process - On average, an ACK is sent for every two packets. Thus, about one third of the packets meets the zero length criteria. These packets are sent to a collector that records 8 Bytes (*seq#* and *ack#*) from each zero-length packet. This information is retrieved in O(1) since it is always at the same anchor. After collecting n records (8 Bytes each) of a flow (e.g., seven APDUs spanning 56 Bytes), the flow is sent to a J48 classifier.

Learning and modeling - For a training data of size n with m attributes, when the tree has a standard growth rate of $O(\log n)$, the computation of creating a single tree is $O(mn \log n)$, [38, Ch. 6.1].

Thus, roughly speaking, a collector with 128MB can classify 4M flows when relaying on 4 a-APDUs signatures, by capturing only 33% of the total packets ($\approx$ 3% of the traffic volume) or 2.28M flows when relaying on 7 a-APDUs signatures, using a single filtering rule. Furthermore, as we show in the sequal, missing zero-length packets (due to, e.g., temporary lack of resources) will not have a critical effect on the performance, as the missing information can be recovered from the next zero-length packet of that flow. Thus, this classification method is expected to be more resilient, and more importantly, to scale with the network capacity.

In comparison, gathering the size of the first few packets, as some studies have suggested (e.g., [24]–[26]) can be accomplished in two ways. The first requires a CAM rule-set for each flow, thus, the number of rules scale with the number of flows, which is not affordable for high traffic volumes. Second, requires capturing all packets by, e.g., a single CAM rule, however, this requires from the CPU to handle and process each packet of each flow, which is about 3 times more than the zero-length method. Obviously, missing packets have a significant impact on the latter performance. Methods that rely on flows' statistics also require capturing all the packets (e.g., [40]), thus, may suffer from lack of scalability as well.

C. Extension to UDP

In contrast to TCP, the User Datagram Protocol (UDP) is a simple, unreliable, connectionless transport layer protocol with minimal protocol mechanisms, neither guaranteeing delivery (reliability), ordering, or duplicate protection communication. UDP has no handshaking dialogues. There is no connection setup and data is sent without any feedback from the destination, which implies that in UDP there are no acknowledgments (ACKs), retransmissions, timeouts and datagram reordering. Thus, our a-APDU mechanism cannot attain the protocol fingerprints based on zero length packets, as it can in the TCP case. However, the classifier can still derive a-APDU sequences delivered between endpoints, by only inspecting the UDP header. In particular, we suggest accumulating the length field available in the UDP header, which specifies the length in bytes of the UDP header and payload, until some data is sent in the opposite direction. Similar to the TCP case, we utilize these a-APDU sequences as the attribute set for our classifier. The fact that there are no retransmissions hence no duplicates and there is no reordering, serves to the a-APDU mechanism's benefit, as it guarantees that as long as there are no packet losses, the a-APDU fingerprints will be very accurate.

Obviously in this case, the classifier is required to record the UDP lengths from all UDP packets and not just from a small fraction of the traffic. Yet, it may be doable as one only needs to extract and accumulate the Length field in the UDP header. Furthermore, according to our experiments (Section IV), for UDP traffic, the classifier reaches very high accuracy after two a-APDU exchange. We conjecture that short fingerprints are sufficient due to high fingerprinting granularity and a relatively small number of UDP applications traversing the network.

IV. Implementation and Performance Evaluation

In this section, we evaluate our a-APDU based traffic classifier which we have implemented. After introducing our design and implementation, we thoroughly evaluate the performance on a sterile traffic trace that was generated in our lab, as well as a replayed, large set of published traffic traces, provided in [39].

A. Implementation

Based on the observations and techniques described in the previous sections, we designed, implemented and tested an automated classifier, applicable to real time traffic classification. The system is comprised of a light-weight two-phase procedure, which is suitable for virtualized switches. In particular, in the first phase, the classifier learns the protocols' behavior from a set of a-APDU sequences retrieved from the zero-length messages of the training data. This learning phase is done offline (i.e., *a-priori*). In order to label the flows in real time, the classifier builds and learns a model for each number of a-APDU exchanges, such that at any given time, the classifier holds a label for each active flow. Accordingly, the classifier will have a few fingerprints for each application, based on a single a-APDU entry, two entries, three entries, etc. On the one hand, very short signatures do not hold sufficient information to distinguish between applications. On the other hand, lengthy signatures have large entropy, which encumbers the classification procedure, in some cases even deteriorating accuracy (see discussion below).

In the second (online) phase, the classifier constructs the a-APDU sequences based on the received zero-length packets. Then, after each a-APDU update, the classifier labels the flow by the appropriate model. For example, even after receiving the first a-APDU the classifier will label the flow based on a single a-APDU signature entry (e.g., the classifier will answer the question: "given a single a-APDU entry, which is the most probable application that has generated this a-APDU?"). After receiving each additional a-APDU record (i.e., instead of k entries in the flow signature there are $k+1$ entries), or in case of a captured FIN or Reset message, the classifier updates its flow label. Obviously, the label accuracy is determined by the length of these a-APDU sequences.

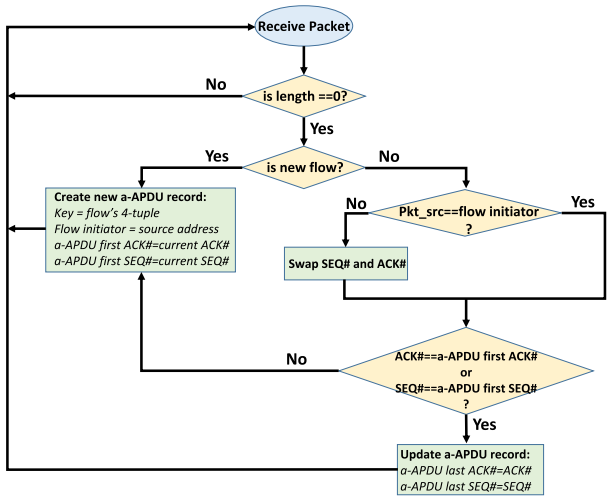


Fig. 4. A procedure for sampling traffic flows and storing their fingerprint at a collecting point. The procedure involves three entities: (i) The zero-length packets sampling point (the top part in the figure). (ii) The a-APDU collecting point, which is responsible for constructing the a-APDU (the lower right part of the figure). (iii) The classifier, which needs continuously (and in real time) to classify each passing flow for each newly received a-APDU.

We implemented and tested the suggested scheme in Java, using Weka's classifiers library. Specifically, after the learning phase, the program receives a stream of captured zero-length packets, which are sent to a flow collector database that gathers packets into flows according to their 4-tuple (i.e., source/desitnation IP address/port). It is important to emphasize that although these fields can be highly useful for the classification process (e.g., port number may reflect on the application), since we wish to quantify the accuracy of the a-APDU fingerprint method isolated from other factors, and specifically without involving attributes from the transport and the network layers, we do not take advantage of the port number or any other such field and the 4-tuple is *only* utilized as a key for the database table, and do not take any part in the classification procedure. Then, each entry in the table is mapped to an a-APDU linked-list. Once the collector adds a new a-APDU record to a flow entry, it sends the classifier the a-APDU sequence it obtained so far, excluding the new a-APDU that is still being measured, for labeling.

In Figure 4, we depict the algorithm flowchart. Specifically, upon a zero-length packet arrival of unseen flow, the collector creates a new a-APDU record and stores it in the database, where the 4-tuple of each flow is used as a key. Each a-APDU record comprises the first and last *ack*# and *seq*# which indicates the a-APDU boundaries, and the direction of the APDU relative to the flow initiator, which is used for *seq*# and *ack*# alignment. When a zero-length packet arrives, and its 4-tuple already exists in the collector's database, the collector first checks that the *ack*# and *seq*# are relevant (e.g., this is not a retransmission, or out of order packet) then it checks if either the current packet *seq*# or *ack*# is greater than the stored *seq*# or *ack*#, respectively. Note that if only one of these fields grows and it is the same field as the last update for this particular flow, we need only to *update* the last *ack*# and *seq*#, as no information passed from the other side during this period. Otherwise, the collector creates a new a-APDU record with first and last *ack*# and *seq*# equal to the value of the packet's *ack*# and *seq*#, respectively. Then, the collector adds the new a-APDU to the corresponding flow entry, and sends the updated a-APDU sequence for classification.

TABLE I
STERILIZED DATASET LABEL COUNT

No.	Label	Count
1	facebook	6512
2	gmaps	15252
3	youtube	32840
4	hangouts	28633
5	skype	17875

B. The Dataset

To assess our classifier, we first generated sterilized traffic by using virtual containers called Docker [41]. Specifically, we installed in each container a single application and a tcpdump process. Then, we used open-source automation tool, named Sikuli [42], to artificially generate *encrypted* traffic (i.e., surfing in Facebook, searching places in Google-maps, watching Youtube videos and performing text/voice and video calls on Skype and Google-Hangouts). Using the intra tcpdump process, we captured $101,112$ TCP flows at total. Even though the generated traffic was fully encrypted, since the traffic was captured inside an isolated container that contained a single application, it could be labeled easily. Table I summarizes the number of flow instances obtained from each class.

To further examine our classifier, we used a published dataset, available in [39], that contains over 35GB of TCP and UDP traffic ($1,262,022$ flows) with full packet payload. For the learning phase, flows were labeled by the nDPI tool [43], which succeeded to label $243,146$ TCP flows with 54 labels. The nDPI tool was recently reported as attaining the highest accuracy among commercial and open-source DPI tools [44]. Table II summarizes the number of flow instances obtained from each class.

Note that $30,780$ flows were labeled by the nDPI tool as "Unknown". We would like to emphasize that as far as the classifier is concerned, an "Unknown" label is a legitimate label, i.e., we expect the classifier to characterize how an "Unknown" a-APDU sequence looks like, and to identify an upcoming "Unknown" flow accordingly, as it does with any other application. Accordingly, we count and report the false negatives and false positives regarding the "Unknown" label as we do for all other applications. Obviously, if one can break, identify and label some of the applications comprising the "Unknown" applications, the classifier will also be able to identify these applications after incorporating these labels in the training phase. Note that this dataset also contains encrypted traffic, and still, the nDPI engine was able to recognize part of the flows due to unencrypted headers and handshakes parameters from the initial connection stages.

C. Overall Results

1) Table I Dataset: As previously mentioned, the classifier is meant to work in real time, classifying traffic on the

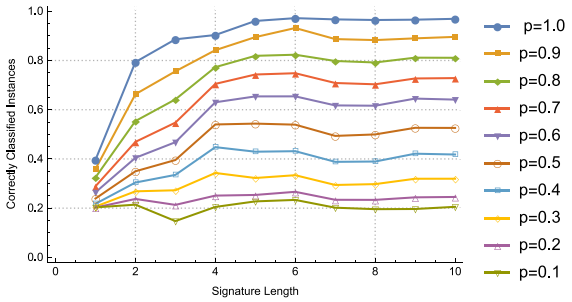


Fig. 5. Correctly classified flows versus the signature lengths, for various values of the sampling probability p.

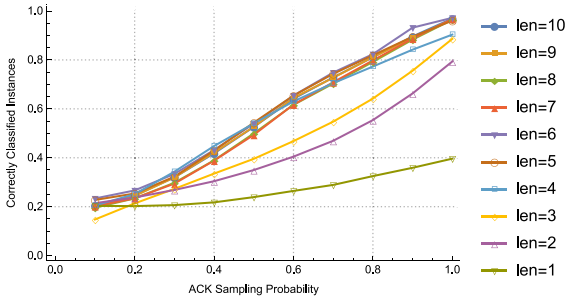
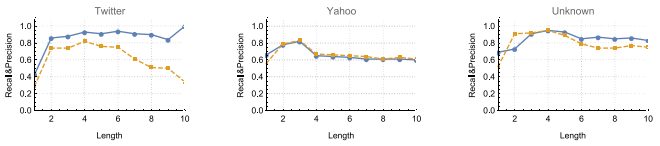


Fig. 6. Correctly classified flows versus the zero-length packet sampling probability, for various signature lengths.

fly. Accordingly, whenever a new flow has been detected, indexed and assigned its first a-APDU, or whenever an additional a-APDU is appended to an already indexed flow, the classifier will try to identify the flow. Obviously, the classifier is expected to provide better classification accuracy the longer the flow signature is, yet since we expect the classifier to identify the flow as soon as possible, we expect it to provide reasonable accuracy even after the first a-APDUs.

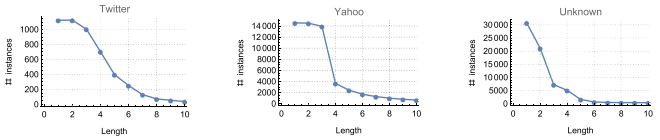
Naturally, in virtualized environment, the capturing capabilities depend on the availability of the shared resources. Thus, to assess the performance deterioration in case of lack of resources, we sampled the zero-length packets of the dataset in Table I uniformly at random. In particular, in the training phase, the classifier built a model based on all zero-length packets, and then, in the online phase, the classifier gathered a-APDUs based on random samples of zero-length packets. In Figure 5, we show the fraction of correctly classified flows versus the signature lengths for several sampling probabilities. Interestingly, the classifier correctly classified over 80% of the flows even when 20% of the zero-length packets were dropped. Moreover, we see that when only 10% of the zero-length packets were sampled, the classifier classified correctly $\approx$ 20% of the flows regardless of the signature length. In Figure 6, we show the fraction of correctly classified flows versus the sampling probabilities for all signature lengths. As we see, the classifier gained much more information from the zero-length packets when the signature length are greater than 3. Further, all signatures with length greater than 3 have the same improvement as the sampling rate grows.

2) Table II Dataset: To provide a rigorous evaluation of our classifier, we compare it with other classification techniques,



(a) Twitter (b) Yahoo (c) Unknown

Fig. 7. The recall (orange dashed) line and precision (blue solid) attained at each APDU length for Twitter, Amazon and unknown traffic.



(a) Twitter (b) Yahoo (c) Unknown

Fig. 8. The total number of instances observed at each APDU length for Twitter, Amazon and unknown traffic.

specifically, according to the size of the first few packets, and according to flow's statistics (min/max/mean/var packet size).

In Figure 7, we depict the performance of three representative applications, namely Twitter, Yahoo and Unknown labels, as a function of a-APDU signature length. Particularly, the solid blue line and the dashed orange line depict the precision and recall values, respectively, for different lengths of a-APDUs exchange. Due to tractability, Figure 7 provides a representative set of applications, which highlight some points.

As can be seen in the figure, the accuracy both in terms of precision and recall is high for all three applications (as well as others) even after receiving few a-APDUs, which is encouraging as it indicates that flows can be identified with high accuracy even after a relatively small number of received zero-length packets. For example the classifier has detected Twitter traffic with high accuracy after 2 a-APDU exchanges, with precision and recall of 78% and 82%, respectively, Yahoo traffic with highest precision and recall after 3 a-APDU exchanges of 82% and 83%, respectively. Surprisingly, also the Unknown traffic attains very high precision and recall of over 90% after 2 a-APDUs and 3 a-APDUs, respectively (Figure 7(c)), which means that the classifier managed to characterize and learn these unknown applications' general behavior and later on to detect them, although this "Unknown" label comprises a variety of applications that were not recognized by the nDPI and are expected to have high variability.

In Figure 7(a) we see that the precision and recall are not necessary coordinated. Specifically, we see that after the 5th a-APDU the Twitter traffic recall deteriorates while the precision increases reaching 1 after appending the 10th a-APDU. Recall that "Recall" counts the number of truly detected flows out of all the flows with the same label, i.e., the "Recall" metric penalizes for each undetected Twitter flow. The "Precision" counts the number of truly detected flows out of all the flows that were labeled with the same label, i.e., it penalizes for each flow that was mistakenly detected as Twitter. Accordingly, high "Precision" implies that there were very few faulty detected Twitter flows.

TABLE II
PUBLISHED DATASET LABEL COUNT

No.	Label	Count	No.	Label	Count	No.	Label	Count
1	Amazon	625	19	IAX	1	37	POPS	4
2	Unknown	30780	20	CiscoVPN	3	38	Whois-DAS	17
3	Yahoo	14575	21	DropBox	1	39	PostgreSQL	4
4	POP3	23	22	RDP	79189	40	MySQL	300
5	IRC	3	23	eDonkey	804	41	BitTorrent	20062
6	IMAPS	7	24	SSL	3228	42	H323	115
7	Telnet	194	25	FTP-CONTROL	40	43	DNS	11
8	HTTP-Proxy	227	26	IPsec	1	44	LastFM	5
9	MSN	434	27	SSH	29380	45	CiscoSkinny	2
10	Facebook	5896	28	GMail	39	46	VNC	405
11	NFS	1	29	SMB	21	47	UPnP	1
12	FTP-DATA	1158	30	Google	9141	48	eBay	67
13	Oscar	8	31	HTTP	34753	49	Apple	532
14	Flash	15	32	Skype	227	50	Twitter	1125
15	WindowsUpdate	664	33	Oracle	3	51	CNN	5
16	AppleiTunes	45	34	OpenVPN	1	52	RTMP	540
17	YouTube	2011	35	Wikipedia	3954	53	Kerberos	4
18	SOCKS5	60	36	GoogleMaps	683	54	IMAP	4

Unexpectedly, for some applications after a certain signature length the performance deteriorates, e.g., the Twitter recall performance declines after 4 a-APDUs signature (Figure 7(a)), and both precision and recall slightly decrease after 3 a-APDUs for the Yahoo traffic (Figure 7(b)). The reason for this performance drop is twofold. First, it is important to note that for some applications after receiving a certain number of a-APDUs the consecutive a-APDUs do not contain any additional information regarding the generated application. Sometimes, the variability of such additional a-APDUs can even mislead the classifier as the additional information is indistinguishable between the "real" generating application and others. The second and more important is due to the significant reduction in the number of instances with large number of a-APDUs in the examined dataset. Note that the applications chosen to be presented in Figure 7 are chosen in order to illustrate behavioral trends of the precision and the recall accuracy, and do not represent the predominant share of traffic. As we will show in the sequel, applications that comprise the dataset predominant share of traffic, such as RDP, SSH or BitTorrent, experienced a very high accuracy rates (higher than the ones depicted in the figure). However, it is important to note that even though the applications depicted in the figure hold only a small fraction of the overall instances in the dataset ($\approx 5\%$), the classifier attained very reasonable accuracy, which means that the suggested fingerprint is indeed a good discriminator (for completeness, the results of all applications are given in the Appendix). In Figure 8 we show the total number of instances observed at each APDU length for the three representative applications, Twitter, Amazon and unknown traffic. As can be seen in the figure, indeed there is high correlation between the decrease in the number of instances as the signature length increases with the Recall and Precision attained as can be seen in Figure 7. For example the number of instances for Twitter starts decreasing for signature length greater than 3 a-APDUs and so does the recall performance (Figures 7(a) and 8(a)). The number of instances for Yahoo decreases for signature length greater than 3 a-APDUs with high correlation with the performance degradation both in terms of recall and in terms of precision. Note however, that even though the number of instances decreased dramatically (from over 14,000 instances to less than 4,000 instances) the performance was only slightly reduced (from 0.8 recall and precision rate to around 0.7). The reason is that despite the high reduction, the number of instances even for signature length of 4, 5 and even 8 a-APDUs is still high (over 1,000 instances). Also the first few a-APDUs are quite dominant for the Yahoo application and the additional a-APDUs with less instances are distinct and unique, such that even for signature length of large number of a-APDUs with relatively few instances attains quite high performance.

In order to fully understand the classifier miss detections, and, in particular, for any miss detection understand to which *other application* was the data classified, we include the classifier *confusion matrices*. Table III and Table IV depict these matrices for TCP and UDP, respectively. In the matrices, each row is indexed with the label (application type) given by the nDPI engine (which we refer to as ground truth), while the column index represents the label *given by our classifier*. The entry itself represents the percentage of the traffic for which our classifier gave the specific label. Thus, diagonal elements depict the percentage of correct classifications. Note that the application recall (precision) is the ratio between the diagonal value to the sum of elements in its row (column). For example, 83.4% of Yahoo's traffic was correctly labeled. 0.2% was labeled as unknown traffic, and 10.9% was labeled as HTTP traffic. For completeness, in the first column right next to the application name we indicate the number of instances. The entries specified in the tables are based on at most four a-APDU signatures, i.e., during classification after reaching 4 a-APDUs the subsequent a-APDUs were ignored.

Figures 5–9 gives some evident that 4 a-APDUs are sufficient to classify accurately the applications in both datasets. We conjecture that the first 4 a-APDUs indeed capture the

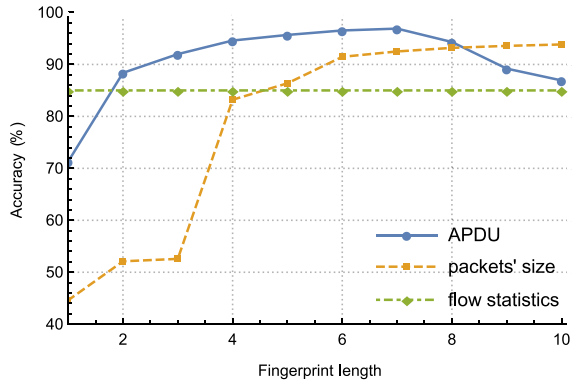
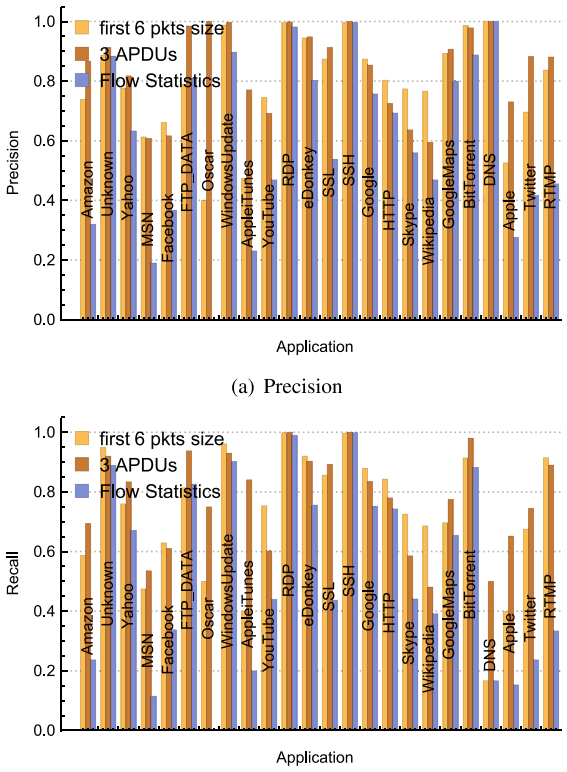


Fig. 9. The classifier accuracy curve as a function of fingerprint length. Note that the APDU classifier suggested can achieve very high accuracy when "stopped" at length 7.



(a) Precision

(b) Recall

Fig. 10. Comparison between precision and recall when classifying according to the first-six packets' size fingerprint, first 3 APDU exchange fingerprint and flow-statistics fingerprint (i.e., min/max/mean/var of packet size). These fingerprints were evaluated under a J48 classifier, which attained the highest accuracy among other WEKA classifiers.

amount of data that is required to be exchanged in the negotiation phase of an application. For example, encrypted applications that wish to exchange keys typically rely on a four-way-hand-shake procedure. The amount of data that is sent afterwards may also indicate on the generating application, yet, it is much less conclusive than the negotiation phase. It is important to note that since our classifier is based on the sequence of a-APDU size, and not on the content, it is immune to encryption, accordingly, as can be seen in the tables even though some applications (e.g., Facebook, Twitter and Google) use an encrypted connection, they were classified correctly with high confidence.

Remark 1 (Real Time Applicability): As shown in Section III-B, the suggested method requires only a small amount of resources and therefore is applicable in very restrictive environments, ones which may not have strict guarantees on the available resources, or real-time cases where complex solutions may limit the ability to work at wire-speeds. Each time a new a-APDU is constructed by the observer, it is added to the already captured sequence of a-APDUs at the classifier. Accordingly, the classifier will try to classify each flow based on the information it currently has: after receiving one (the first) a-APDU, after receiving two a-APDUs, and so on. Obviously the accuracy will improve after receiving several a-APDUs. However, as can be observed from Figure 9, the accuracy is very high even after a few a-APDUs. Thus, we conclude that this classification can be done in real-time, and without waiting for the entire flow before making a decision.

Finally, we compare our classifier to other classification techniques. Specifically, we compare it to a classifier which labels each flow according to the size of the first few captured packets ([24]–[26]), i.e., the k-packet-size classifier captures the first k packets traversing the switch and labels the flow according to the sequence of these packet sizes. In addition we compare our classifier to one which labels each flow according to the flow statistics (min/max/mean/var packet size), i.e., after the flow terminates, the classifier computes the flow statistics and labels the flow accordingly (e.g., [22]). Generally, the information required by the latter is available only after a flow finishes, hence, it may not be suitable for real-time flow classification. In Figure 9 we compare for each fingerprint length the classifier accuracy according to the 3 attribute sets; the size of the first-k packets (dashed-orange line), flow statistics (dot-dashed green line), and the first-k a-APDUs signature size (solid-blue line). As can be seen, taking into account all the applications, the classifier attained its highest accuracy when it used the first-7 APDUs' attribute set, with 96.9% average accuracy in a tenfold cross validation, for 54 different traffic labels. Overall, the a-APDU classifier attained the best results; even after 2 a-APDUs it attained better results than the flow-statistics classifier. Note that there is a slight degradation for signature length greater than 7 (see earlier discussion). The first-k packets classifier attained high accuracy which are comparable with the a-APDU classifier, after capturing the first 10 packets.

Next we analyze the overall results depicted in Figure 9 to the different applications. In particular, in Figure 10 we compare the precision and recall evaluations for representative set of 24 labels when the classifier classifies according to the first-six packets' size, the first 3 a-APDU exchange and flow-statistics. The reason we compared a signature of length 3 of the latter to a signature of length 6 of the former lies in the dimensionality of their feature set. Specifically, recalling that each a-APDU record holds two values (the *ack*# and *seq*#), a signature of length 3 essentially holds six values. Thus, we

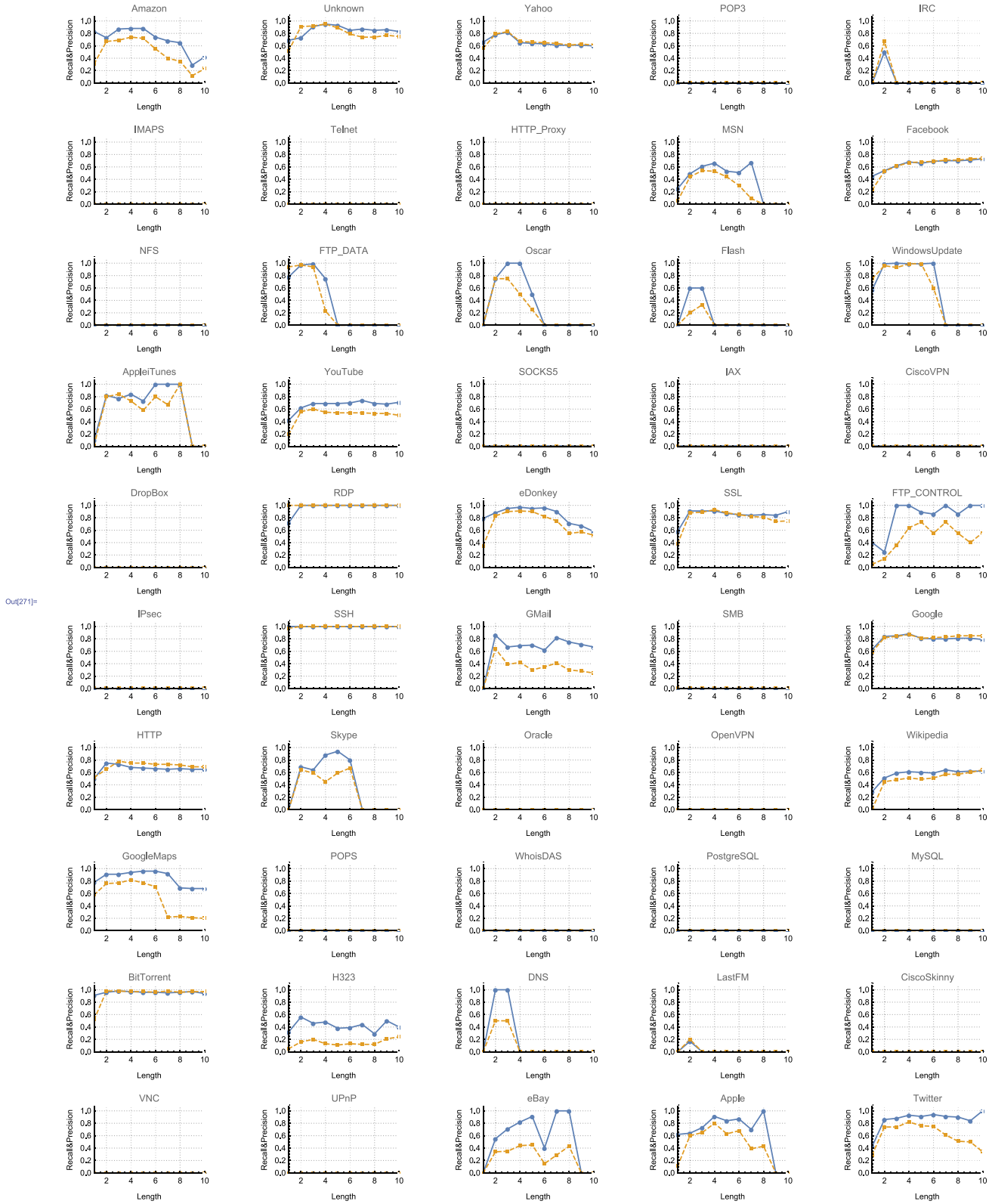


Fig. 11. The recall (orange dashed) line and precision (blue solid) attained at each APDU length for all captured data.

wished to evaluate the accuracy of both schemes when the signatures hold the same amount of information. Moreover, some studies (e.g., [24]) claimed that the size of the first five packets is sufficient for accurate classification. Thus, we used an attribute set with a similar dimensionality. We would like to emphasize that the focus of this paper is on a new feature extraction (a-APDU based one) and any classification technique can be utilized for this matter, e.g., the K-Means

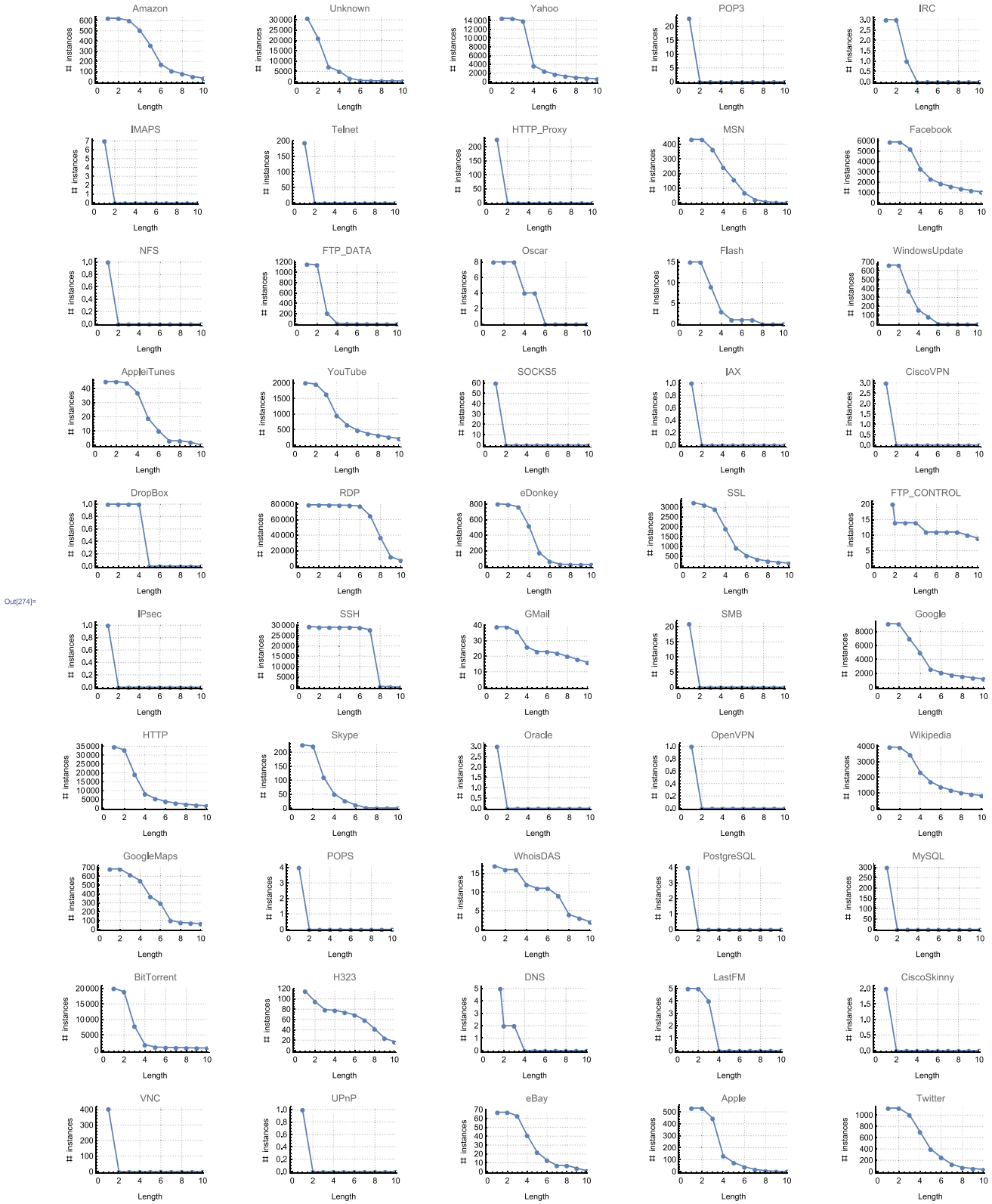


Fig. 12. The total number of instances observed at each APDU length for all captured data.

algorithm for traffic classification which was utilized by the first *k*-packets scheme in [24]. In this paper the two criteria for choosing a classifier were performance and simplicity. We have tested several classifiers which are included in the Weka which is a collection of ML algorithms for data mining tasks, not only the J48 classifier is simple to implement (a tree based classifier), it also gave the best results for both datasets we examined and for all three attribute sets (i.e., a-APDUs, first *k*-packets and flow statistics), in agreement with a more recent study [32].

TABLE III
TCP CLASSIFICATION CONFUSION MATRIX

App/Tag	#inst.	a	b	c	i	j	l	m	n	o	p	q	v	w	x	y	aa	ab	ad	ae	af	ai	aj	ao	ap	aq	av	aw	ax	az
a = Amazon	625	**69.4**	0.7	7.8	0.2	3.2						0.2			1.2				0.8	15		1.5						0.2		
b = Unknown	30780	0.1	**92**	0.7		0.2						0.1	0.2	0.5	0.9				0.2	3.4		0.1		1.2						0.4
c = Yahoo	14575	0.1	0.2	**83.4**	0.1	2.2						0.4			0.2				0.8	10.9		1.5		0.1				0.1	0.1	
e = IRC	3		100																											
i = MSN	434		0.3	5.2	**53.6**	4.4						4.1			0.5				0.3	26.4		4.1					0.5	0.5		
j = Facebook	5896	0.2	0.2	6.9	0.4	**61**						1.2			0.3				2.5	23.8		3						0.3	0.2	
l = FTP-DATA	1158		0.5	0.5			**93.8**					0.5								3.3		1.4								
m = Oscar	8							**75**							25															
n = Flash	15		22.2	11.1					**33.3**										22.2	11.1										
o = Win.Update	664		0.3							**93**					0.3						6.4									
p = AppleiTunes	45	2.3									**84.1**				13.6															
q = YouTube	2011		0.4	7	0.4	6						**60.2**		0.1	0.4				4.3	17.9		3	0.1					0.2		
u = DropBox	1					100																								
v = RDP	79189												**100**																	
w = eDonkey	804		9.2											**90.3**	0.3					0.1				0.1					
x = SSL	3228		5.3	0.3	0.1	0.7					0.4				**89.2**				1.1	1.7			0.1	0.1			0.1	0.6	0.2
y = FTP-CTRL	40		64.3													**35.7**														
aa = SSH	29380																**100**													
ab = GMail	39			2.8											2.8			**38.9**	50	5.6									
ad = Google	9141		0.2	2.4		2.2						0.6			0.3			0.1	**83.5**	9.1		0.9	0.4					0.1		
ae = HTTP	34753	0.1	1	7.5	0.3	5.5						0.9	0.1		0.2				2.4	**78**		3.1		0.3				0.2	0.2	
af = Skype	227		22.5	0.9	0.9					0.9										16.2	**58.6**									
ai = Wikipedia	3954	0.1	0.1	8.3	0.3	6.9	0.1					1.8			0.1				1.6	32.4		**48.1**						0.2	0.1	
aj = GoogleMaps	683			1		1.1						0.6			0.2				11.5	7.5		0.6	**77.5**							
al = WhoisDAS	17			12.5									68.8						6.3	6.3		6.3								
ao = BitTorrent	20062		0.9										0.1		0.1					0.9				**98**						
ap = H323	115		7.6										39.2							3.8				1.3	**20.3**					27.8
aq = DNS	11		50																							**50**				
ar = LastFM	5		75																	25										
av = eBay	67			9.5	1.6	7.9						1.6							1.6	39.7		1.6		1.6			**34.9**			
aw = Apple	532	0.4	0.2	7.4	0.2	4.9						0.2			3.4				0.2	13.5		3.1						**65.2**	1.1	
ax = Twitter	1125		0.3	2.2	0.1	2.3						0.7			3				3.4	11.3		1.1		0.2				1	**74.5**	
ay = CNN	5											50								50										
az = RTMP	540		4.2	0.2											1.6					1.6					3.4					**89**

TABLE IV
UDP CLASSIFICATION CONFUSION MATRIX

App / Tagged As	BitTorrent	DNS	eDonkey	NTP	RTP	Skype	Viber
BitTorrent	**99.3%**	0.6%					0.1%
DNS		**100%**					
eDonkey		0.2%	**99.8%**				
NTP				**100%**			
RTP	0.3%	3.8%	0.2%		**91.8%**	4%	
Skype	0.4%	1.1%	0.6%		24.1%	**73.7%**	
Viber	12.3%	4.1%		2.7%			**80.8%**

V. CONCLUSION

In this paper, we suggested a classification strategy which samples only zero-length packets in a TCP flow, and is implementable with only a single filtering rule. We introduced a novel fingerprinting mechanism, which expresses the protocols' behavior in a compact and efficient way, regardless of the network parameters or the measurement time and location. Experiments using real traffic showed very encouraging results, classifying a large variety of applications with a relatively small error ratio.

APPENDIX

In Figure 11 we depict the performance for 50 labels. The blue solid line and the orange dashed line depict the precision and recall values, respectively, for different lengths of a-APDU exchange, as a function of the number of captured a-APDUs. It can be seen that the number of a-APDUs required for maximal accuracy ranges between the applications. Nonetheless, as explained in Section IV-C, for most applications a sequence of 4 a-APDUs is sufficient to provide a very accurate classification. Figure 12 also provides the typical number of a-APDUs generated by each application, and specifically the distribution of the number of a-APDUs generated by each application (how many flows generated n a-APDUs before termination). As expected, applications that lasted at most two a-APDUs, such as POP3 and VNC, attained very poor performance. Further note that few applications such as IPsec and UPnP have very few instances to attain rigorous conclusions. Note that there are 19 labels that our classifier could not detect at all. Nonetheless, most of these labels had only a single a-APDU exchange. This essentially reveals the limitations of our a-APDU based classification. Small number of a-APDU exchange indicates on a half-duplex communication pattern (such as keep-alive messages or one way notifications). This can be seen for example in dropbox, POP3, IMAPS, etc. Such a half-duplex pattern is not long or detailed enough for our classifier. Moreover, we assume that applications which significantly change their behavior based on the platform on which they run may also mislead the classifier. Fortunately, in this case, there is a relatively easy work-around, and one can simply create a fingerprint per-application per-platform.

REFERENCES

[1] B. Pfaff *et al.*, "The design and implementation of open vSwitch," in *Proc. NSDI*, 2015, pp. 117–130.

[2] M. Rifai *et al.*, "Too many SDN rules? Compress them with MINNIE," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, 2015, pp. 1–7.

[3] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better NetFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 245–256, 2004.

[4] N. Hohn and D. Veitch, "Inverting sampled traffic," in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas.*, 2003, pp. 222–233.

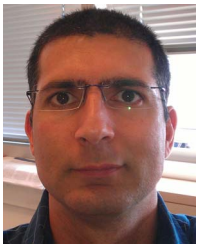
[5] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: Control of volume and variance in network measurement," *IEEE Trans. Inf. Theory*, vol. 51, no. 5, pp. 1756–1775, May 2005.

[6] S. Fernandes, C. Kamienski, J. Kelner, D. Mariz, and D. Sadok, "A stratified traffic sampling methodology for seeing the big picture," *Comput. Netw.*, vol. 52, no. 14, pp. 2677–2689, 2008.

[7] S. Zander, T. Nguyen, and G. Armitage, "Sub-flow packet sampling for scalable ML classification of interactive traffic," in *Proc. IEEE 37th Conf. Local Comput. Netw. (LCN)*, 2012, pp. 68–75.

[8] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks," in *Proc. Symp. SDN Res.*, Santa Clara, CA, USA, Mar. 2016, p. 6.
[9] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in OpenFlow networks," in *Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst.*, 2014, pp. 177–188.
[10] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *Proc. Int. Conf. Passive Active Netw. Meas.*, 2015, pp. 347–359.
[11] Q. Maqbool, S. Ayub, J. Zulfiqar, and A. Shafi, "Virtual TCAM for data center switches," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2015, pp. 61–66.
[12] R. McGeer and P. Yalagandula, "Minimizing rulesets for TCAM implementation," in *Proc. IEEE INFOCOM*, 2009, pp. 1314–1322.
[13] H. Zhu, S. Chen, L. Zhu, H. Li, and X. Chen, "RangeTree: A feature selection algorithm for C4.5 decision tree," in *Proc. IEEE 5th Int. Conf. Intell. Netw. Collaborative Syst. (INCoS)*, 2013, pp. 17–22.
[14] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 229–240, 2005.
[15] M. Tavallaee, W. Lu, and A. A. Ghorbani, "Online classification of network flows," in *Proc. IEEE 7th Annu. Commun. Netw. Services Res. Conf. (CNSR)*, 2009, pp. 78–85.
[16] L. Grimaudo, M. Mellia, and E. Baralis, "Hierarchical learning for fine grained Internet traffic classification," in *Proc. 8th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2012, pp. 463–468.
[17] Y. Wang, Y. Xiang, and S. Yu, "Internet traffic classification using machine learning: A token-based approach," in *Proc. IEEE 14th Int. Conf. Comput. Sci. Eng. (CSE)*, 2011, pp. 285–289.
[18] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Proc. IEEE Conf. Local Comput. Netw. 30th Anniversary*, 2005, pp. 250–257.
[19] A. Dainotti, W. De Donato, A. Pescape, and P. S. Rossi, "Classification of network traffic via packet-level hidden Markov models," in *Proc. IEEE Glob. Telecommun. Conf. (IEEE GLOBECOM)*, 2008, pp. 1–5.
[20] Y.-D. Lin, C.-N. Lu, Y.-C. Lai, W.-H. Peng, and P.-C. Lin, "Application classification using packet size distribution and port association," *J. Netw. Comput. Appl.*, vol. 32, no. 5, pp. 1023–1030, 2009.
[21] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification: Identifying SSH and Skype," in *Proc. CISDA*, vol. 9, 2009, pp. 289–296.
[22] S. Zhao *et al.*, "A novel online traffic classification method based on few packets," in *Proc. IEEE 8th Int. Conf. Wireless Commun. Netw. Mobile Comput. (WiCOM)*, 2012, pp. 1–4.
[23] D. Tong and V. Prasanna, "Dynamically configurable online statistical flow feature extractor on FPGA," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, 2013, pp. 1–6.
[24] L. Bernaille, A. Soule, M. I. Jeannin, and K. Salamatian, "Blind application recognition through behavioral classification," CNRS, Paris, France, Rep. LIP6, 2005.
[25] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, 2006.
[26] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. ACM CoNEXT Conf.*, 2006, p. 6.
[27] J. Zhang, Z. Qian, G. Shou, and Y. Hu, "Online automatic traffic classification architecture in access network," in *Proc. 9th Int. Conf. Electron. Meas. Instrum. (ICEMI)*, 2009, pp. 24–29.
[28] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for TCP traffic classification," *Comput. Netw.*, vol. 53, no. 14, pp. 2476–2490, 2009.
[29] T. Groléat, M. Arzel, and S. Vaton, "Stretching the edges of SVM traffic classification with FPGA acceleration," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 3, pp. 278–291, Sep. 2014.
[30] A. Rizzi, S. Colabrese, and A. Baiocchi, "Low complexity, high performance neuro-fuzzy system for Internet traffic flows early classification," in *Proc. IEEE 9th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2013, pp. 77–82.
[31] B. Yang, G. Hou, L. Ruan, Y. Xue, and J. Li, "SMILER: Towards practical online traffic classification," in *Proc. ACM/IEEE 7th Symp. Archit. Netw. Commun. Syst.*, 2011, pp. 178–188.
[32] M. Shafiq *et al.*, "Network traffic classification techniques and comparative analysis using machine learning algorithms," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, 2016, pp. 2451–2455.
[33] P. Perera, Y.-C. Tian, C. Fidge, and W. Kelly, "A comparison of supervised machine learning algorithms for classification of communications network traffic," in *Proc. Int. Conf. Neural Inf. Process.*, 2017, pp. 445–454.
[34] R. Archanaa, V. Athulya, T. Rajasundari, and M. V. K. Kiran, "A comparative performance analysis on network traffic classification using supervised learning algorithms," in *Proc. IEEE 4th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, 2017, pp. 1–5.
[35] P. Mehta and R. Shah, "A survey of network based traffic classification methods," *Database Syst. J.*, vol. 7, no. 4, pp. 24–31, 2017.
[36] T. R. Patil and S. Sherekar, "Performance analysis of naive Bayes and J48 classification algorithm for data classification," *Int. J. Comput. Sci. Appl.*, vol. 6, no. 2, pp. 256–261, 2013.
[37] J. R. Quinlan, *C4.5: Programs for Machine Learning*, vol. 1. San Mateo, CA, USA: Morgan Kaufmann, 1993.
[38] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Amsterdam, The Netherlands: Morgan Kaufmann, 2005.
[39] V. Carela-Español, T. Bujlow, and P. Barlet-Ros, "Is our ground-truth for traffic classification reliable?" in *International Conference on Passive and Active Network Measurement*. Cham, Switzerland: Springer, 2014, pp. 98–108.
[40] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, 2005.
[41] *Docker*. Accessed: Apr. 2018. [Online]. Available: https://www.docker.com/
[42] *Sikuli Script*. Accessed: Apr. 2018. [Online]. Available: http://www.sikuli.org/
[43] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," in *Proc. IEEE Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2014, pp. 617–622.
[44] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Extended independent comparison of popular deep packet inspection (DPI) tools for traffic classification," Dept. Comput. Archit., Universitat Politècnica de Catalunya, Barcelona, Spain, Rep. UPC-DAC-RR-CBA-2014-1, Jan. 2014. [Online]. Available: https://www.ac.upc.edu/app/research-reports/html/RR/2014/1.pdf

Joseph Kampeas received the B.Sc. and M.Sc. degrees from the Department of Communication System Engineering, Ben-Gurion University of the Negev, Beer Sheva, Israel, in 2010 and 2012, respectively, where he currently pursuing the Ph.D. degree. He currently holds a research position with General Motors–Advanced Technical Center, Israel. His main research interests are in the area of wireless communication, distributed systems, multiuser scheduling, physical-layer security, machine learning, and game theory. He was a recipient of the Negev Scholarship for Outstanding Ph.D. students in 2014.



Asaf Cohen received the B.Sc., M.Sc., (High Hons.) and Ph.D. degrees from the Department of Electrical Engineering, Technion, Israel Institute of Technology, in 2001, 2003, and 2007, respectively. He is a Senior Lecturer with the Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Israel. He was a Post-Doctoral Scholar with the California Institute of Technology (Caltech). From 1998 to 2000, he was with the IBM Research Laboratory, Haifa, where he researched on distributed computing. His areas of interest are information theory, learning, and coding. He is interested in sequential decision making, with applications to detection and estimation; network security and anomaly detection; network information theory and network coding; statistical signal processing; and coding theory and performance analysis of codes. He served on the program committees of several conferences in the above areas. He was a recipient of several honors and awards, including the Viterbi Post-Doctoral Scholarship, the Student Paper Award at IEEE Israel 2006, and the Dr. Philip Marlin Prize for Computer Engineering in 2000.

Omer Gurewitz received the B.Sc. degree in physics from the Ben Gurion University of the Negev, Beer Sheva, Israel, in 1991 and the M.Sc. and Ph.D. degrees in electrical engineering from the Technion—Israel Institute of Technology, Haifa, Israel, in 2000 and 2005, respectively. He is currently a Tenured Faculty Member with the Department of Communication Systems Engineering, Ben Gurion University of the Negev. From 2005 to 2007, he was a Post-Doctoral Researcher with the Electrical and Computer Engineering Department, Rice University, Houston, TX, USA. He has published highly cited papers in leading journals and conferences. His research interests are in the areas of wired and wireless communication networks with a focus on protocol design, analysis, and performance evaluation.