Towards accurate classification of HTTPS traffic in Software-Defined Networks

José Suárez-Varela
UPC BarcelonaTech, Spain
Email: jsuarezv@ac.upc.edu

Pere Barlet-Ros
UPC BarcelonaTech, Spain
Talaia Networks, Spain
Email: pbarlet@ac.upc.edu

***Abstract*—In nowadays Internet, there is a strong trend to encrypt the traffic in order to protect users' privacy. This results in a hard challenge for traffic classification, as the payload in the packets cannot be accessed anymore. In this context, some techniques were proposed for traditional networks in order to classify this traffic. However, we could not find previous works addressing encrypted traffic classification considering the particularities of the Software-Defined Networking (SDN) paradigm. In this paper we present an OpenFlow-based classification system which combines techniques leveraging information in SSL/TLS certificates and DNS traffic to perform accurate flow-level classification for encrypted traffic. We make experiments with real-world traffic to evaluate the overall classification accuracy of our system as well as the accuracy detecting specific popular applications (e.g., Netflix). Furthermore, we assess the processing overhead when deploying our system in SDN environments. As a result, we observe that the support provided by OpenFlow enables to achieve a high accuracy with a more reduced processing overhead than in traditional networks, where typically the whole traffic is mirrored to an external collector that classifies the traffic.**

I. Introduction

Traffic encryption is gaining a prominent role in the Internet of today. There is a strong evidence that more and more applications are migrating to web services with encryption protocols in order to protect the privacy of their end-users. Thus, some of the major players of the web, such as Google, Facebook, Twitter or Netflix have gradually moved towards encrypted solutions. The importance of this new communication paradigm is also reflected in [1], where they estimated that 70% of the global Internet traffic would be encrypted by the end of 2016, even exceeding 80% in many networks.

Nowadays, a plethora of applications based on secured web services are continuously emerging with very diverse demands of Quality of Service (QoS) requirements. They range from services where the most relevant factor is the average bandwidth achieved (e.g., cloud storage services), to some others where the end-to-end delay is critical (e.g., VoIP). This new scenario forces the necessity of performing a fine-grained network management to optimally exploit the resources in the networks and, in turn, be able to provide the Quality of Experience (QoE) desired by the end-users. In this context, the advent of the Software-Defined Networking (SDN) paradigm enables to achieve a level of flexibility to manage the networks never seen before. The main success of this novel proposal lies in the use of a logically centralized control plane, which can make decisions with a global picture of the state of the network. However, in order to make the most of this new paradigm, it is essential to perform accurate traffic monitoring and classification. This reveals the great importance to identify the applications generating each flow within the bulk of encrypted traffic, given that it implies a large portion of the whole traffic in the networks. This valuable information enables network managers to efficiently perform some network tasks including traffic engineering, QoS level enforcement, or anomaly detection among others.

All these facts pose a new challenge in the field of traffic classification, since most of the traditional classification techniques based on payload inspection become useless when applied to encrypted traffic, as they do not have access to the packet payload anymore [2]. Note also that, traditionally some techniques based on behavioral classifiers [3] have been proposed for traffic classification using only information at the transport layer. However, they do not perform well in current scenarios, which increasingly rely on platforms where the content is dynamically distributed in shared infrastructures (e.g., content delivery networks). Likewise, some proposals in the literature also leverage some information extracted at the transport layer to classify the traffic using machine learning [4]. Nevertheless, they are not well suited for encrypted traffic, as they cannot accurately discern between specific applications when there are plenty of applications generating traffic over the same protocol (i.e, using the same port).

In the light of the above, some cutting-edge classification techniques have been proposed in order to achieve a comprehensive level of classification for encrypted traffic [5]. These techniques basically harness some information that still remain unencrypted in the traffic in order to unveil the applications generating each encrypted flow. In this paper, we focus on two different approaches that we envision that are good candidates to be deployed in SDN-based networks: (i) those techniques which leverage the information in the SSL/TLS certificates exchanged during the initial handshakes and (ii) those methods extracting information from the DNS traffic to then associate encrypted flows to specific domain names. In particular, we consider the techniques proposed in [2] [6] to extract the Server Name Indication (SNI) fields from the SSL/TLS certificates prior to establishing an encrypted connection, and the frameworks presented in DN-Hunter [7] and SFMap [8], where they extract the server hostnames from the DNS queries that clients execute before opening new flows.

The main goal of this paper is to analyze how feasible is to implement in SDN-based networks the classification techniques for encrypted traffic mentioned above. In particular, we rely on the OpenFlow protocol [9] to implement these classification methods and study the tradeoff between accuracy and computing resources needed to deploy them. More specifically, our contributions are the following:

- We designed a classification system for SDN-based environments which combines techniques specifically targeted to encrypted HTTPS traffic. Particularly, we implemented two classification techniques: (i) based on the hostname information of the SNI fields in the SSL/TLS certificates, and (ii) based on the domain name information extracted from the DNS records in the traffic. To the best of our knowledge, there are not previous works directed to tackle traffic classification for encrypted traffic considering SDN-based scenarios.
- Our implementation makes only use of OpenFlow [9] native features. Thus, we leverage the flow-based processing in OpenFlow to implement our classification system without incurring in a high processing overhead in the controller. Note that our solution is not only compatible with OpenFlow, but also can be deployed using some other novel proposals for the southbound interface of SDN that operate at a flow granularity (e.g., P4 [10]).
- We evaluate the overall accuracy of each of the classification techniques we implemented and additionally provide some results specifically regarding some popular applications. All this evaluation was done with a large dataset containing real-world traffic annotated with labels for encrypted flows.
- We assess the processing overhead that implies to execute our classification system in a SDN controller.

As a result, we show that OpenFlow allows to efficiently deploy accurate classification techniques for encrypted traffic. Thus, this protocol provides support to send to the SDN controller only the desired slice of traffic we want to process. In our case, the controller only processes a reduced amount of traffic related to the DNS queries ($\sim$0.08% of the total bytes in our real-world traffic traces) and the first few packets of the HTTPS flows. In contrast, in traditional networks typically all the traffic is forwarded to an external collector (i.e., port mirroring) where all the packets are inspected.

The remainder of this paper is structured as follows: Firstly, in Section II, we provide a brief overview of OpenFlow focusing on the features involved in our design. Section III defines the design of our OpenFlow-based classification system for encrypted traffic. In Section IV, we evaluate our classification system using real-world traffic. Here, we assess the accuracy of the different classification methods implemented and the overhead contribution when deploying our classification system. Section V provides a summary of the most relevant achievements in the literature regarding the work we present in this paper. Lastly, in Section VI, we expose some conclusions and outline some ideas for future work.

II. OpenFlow background

Software-Defined Networking, in a nutshell, emerged as a proposal to separate the control and data planes of the network in different logical entities for the sake of flexibility in network management. In particular, the control plane is implemented in some entities called SDN controllers, while the data plane still remains in the switches of the underlying infrastructure. These two planes should be completely decoupled and can communicate through an interface known as the southbound API. In this context, the OpenFlow protocol [9] was the first proposal to standardize this interface. Thus, since its inception it has been very well received by the academia and industry to the extent that nowadays it has become a dominant protocol in SDN-based commercial products.

Unlike routing in traditional IP networks, OpenFlow introduces the concept of flow-based traffic management. In this way, the behavior of the network is defined by sets of rules which are installed in the switches by the SDN controllers and aggregate the traffic in different flows that have the same treatment. As a consequence, it enables to perform a very flexible management, since it is possible to apply different actions to flows with a level of granularity at the transport layer (e.g., ports). Basically, the OpenFlow rules installed in the switches include the following elements: (i) Match fields: a set of fields (i.e., specific values in the packet headers) that determines the packets that will be aggregated in the rule, (ii) Instructions: a set of actions that will be applied to packets matching the rule (e.g., forward, drop, modify header fields) and (iii) Timeouts: records to define when the rule will be removed from the switch.

Generally, in OpenFlow-based environments two different modes of operation can be differentiated: (i) proactive and (ii) reactive. The former one consists of installing some default rules in the switches to determine the actions to be applied to future incoming traffic. This is similar to the operation in traditional IP networks, where the routing is typically based on predefined rules considering the destination IPs in the packets. Alternatively, the controller can operate reactively to make decisions in real time for particular flows based on the current state of the network. In more detail, the reactive installation of rules works as follows. When a packet matches a rule installed in the switch with an action *output to controller*, this packet (or a portion of it) is encapsulated and forwarded to the controller via an OFPT_PACKET_IN message. Then the controller can process the packet and may install a flow rule (via an OFPT_FLOW_MOD message) with a set of actions to be applied for the subsequent packets of the flow. Remark that in the reactive mode, packets belonging to the same flow are sent to the controller until a specific rule is installed for them in the switch. As a result, the SDN controller can receive more than one packet belonging the same flow. Likewise, when installing the flow rule, it is possible to define two timeouts (hard and idle) associated to this rule. The hard timeout determines the absolute time that the entry can remain in the switch since it was installed, while the idle timeout defines the maximum time interval between two consecutive packets matching this rule.

As a final remark, note that both operation modes can be combined to define proactively an amount rules for some slices of the traffic and process reactively some flows for which is particularly interesting to perform a more comprehensive management (e.g., QoS-aware traffic engineering).

III. Design of the classification system

In this section we describe the design and implementation of our flow-level[1] classification system for HTTPS traffic. As mentioned earlier, we combined some techniques previously proposed for traditional networks and implemented them in a SDN controller considering the particularities of OpenFlow-based environments. Particularly, we considered the two following techniques: (i) DPI (Deep Packet Inspection) on the first few packets of HTTPS flows in order to extract the server hostnames from the SSL/TLS certificates exchanged in the handshakes, and (ii) DPI on DNS traffic in order to extract the server domain names resolved by clients prior to establish encrypted connections. This provides a deep insight of the encrypted traffic in the network, as the server hostnames allow to unveil the applications generating each flow in the traffic.

First of all, we provide a description of the two classification modules that compose our classification system:

1) HTTPS traffic classifier: This classifier processes the HTTPS traffic that arrives to the controller in order to extract the server hostname included in the SSL/TLS certificates. That way, packets are inspected to parse the SNI (Server Name Indication) field of the certificates exchanged during the handshake of each encrypted connection. As they do in [2], we use the SNI strings to then associate the server hostnames to specific applications. For instance, a hostname containing "mail.google.com" provides enough information to infer that the flow was generated by the Google Gmail application.

In order to implement this module with OpenFlow, the controller proactively adds flow rules in the switches to receive the packets matching port 443. However, this policy might imply a high processing overhead for the controller, as it should inspect every HTTPS packet in the traffic. Note that, the classification technique implemented in this module typically extracts only information from the first few packets of each flow, where the certificates are present. The rest of the flow is completely encrypted and, thereby, it is not possible to extract relevant information from the payload. In view of this, we leverage the reactive operation mode in OpenFlow to receive in the controller only the first few packets of each HTTPS flow. Thus, when a HTTPS packet is processed by this module, the controller reactively installs a specific flow entry with high priority in the switch. This rule allows to apply directly in the switch the convenient forwarding action(s) for the subsequent packets matching the 5-tuple fields of the flow and avoids sending these packets to the controller. As a result, it enables to significantly save the processing power needed in the controller without decreasing the classification accuracy achieved by this module.

In more detail, the controller will receive packets belonging the same flow during the time interval from the arrival of the first packet of a flow, to the time when a specific flow entry is installed in the switch to process subsequent packets of this flow. This time interval mainly depends on the following factors: (i) the time to process an incoming HTTPS packet in the switch and forward it to the controller, (ii) the Round-Trip Time (RTT) between the switch and the controller, (iii) the time to process the packet in the controller and send to the switch an OFPT_FLOW_MOD message to install a flow entry, and (iv) the time in the switch to install the new entry in the flow table. The RTT depends on some features of the control infrastructure that connects the switches with the SDN controllers (e.g., the end-to-end delays, the capacity or the utilization of the links). The first and fourth factors mainly depend on the processing power of the switches. And the second factor depends on the processing power and the workload in the controller.

Furthermore, to make our implementation more efficient, when a packet arrives to this module we first check if its associated flow was already labeled. That way, we only inspect the packet if its flow was not classified yet.

In order to implement this module in our system, we made use of the open source tool Bro IDS [11], which has support to extract the SNI fields from certificates.

2) DNS classifier: The technique implemented in this module is based on the assumption that clients typically send a DNS query to resolve the server IP addresses prior to establish HTTPS connections. Thus, monitoring all the DNS queries enables to then associate the server IPs of encrypted flows to their domain names[2]. As a result, this classifier complements the one described previously by offering domain name labels, which are very valuable to unveil the application generating the flows [2]. Furthermore, this module achieves early classification, given its ability to provide a label even before flows are created. In other words, it is possible to foresee from DNS traffic the flows that will traverse the network [7]. This opens up the opportunity to leverage the flexibility offered in SDN-based networks to perform fine-grained management. Thus, for instance, it is possible to select the paths at the beginning of incoming flows considering the estimated QoS requirements of their associated applications.

In order to implement this classification technique with OpenFlow, it is necessary to forward all the DNS traffic from the switch to the SDN controller. To this end, the controller proactively adds flow entries in the switch to receive all the traffic matching port 53. We then show in Section IV-D that the cost of processing the DNS traffic in the controller is quite reduced.

[1]In the rest of the paper we consider a flow as a set of packets sharing the same IP 5-tuple [src and dst IPs, src and dst ports, IP protocol].

[2]Remark that this technique cannot be applied in the odd case of HTTPS-based applications that use hardcoded IPs to establish a connection with the server, as the client does not perform a previous domain name resolution.

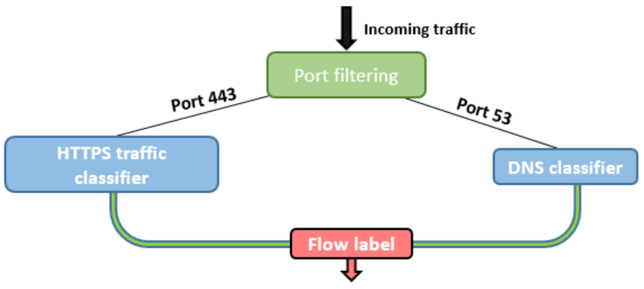

Fig. 1. Scheme of our classification system for encrypted traffic.

Note that in some cases clients do not execute a DNS query prior to create a flow because they had already the domain name resolution stored in their local caches. That way, it is necessary to save in the controller the resolutions of DNS queries at least for the period defined in the TTL (Time-To-Live) records. Thus, our module maintains a local copy with the records that should be stored in client's DNS caches and, thereby, can infer the domain names even if there is not a DNS query just before the start of the flow.

This module was implemented in Python following the guidelines presented in DN-Hunter [7] and SFMap [8] for traditional networks. Particularly, in our solution we maintain a copy of all the DNS records in the traffic, which a priori represents an upper-bound of the accuracy that could be achieved individually by those previous proposals.

In Fig. 1, we show a scheme that describes the operation of our system in the SDN controller. Firstly, as our classification system is only aware of HTTPS and DNS traffic, it filters (by ports) the packets belonging these protocols to process them separately. That way, when either the source or destination ports match port 443, the packet is processed by our "HTTPS traffic classifier". Conversely, in case any of the ports match the port number 53, the packet is processed by the "DNS classifier" in order to store the resolutions from domain names to IP addresses in DNS queries. That way, the "HTTPS traffic classifier" processes the packets from each HTTPS flow until a label is extracted from the certificates or a flow entry is installed in the switches to avoid receiving more packets from this flow. Additionally, we use the information collected by the "DNS classifier" to obtain a label with the domain name associated to each HTTPS flow.

IV. Evaluation of the Classification System

In this section we evaluate our classification system specifically designed for OpenFlow-based environments. To this end, we make experiments with a ground truth we created from a real-world traffic trace. All our experiments were directed to answer the following questions:

- How accurate are the different classification techniques we implemented?

- What is the processing cost in the SDN controller to run the different classification techniques we implemented?

In short, our goal is to analyze how feasible is to deploy in SDN-based networks the classification techniques we considered for HTTPS traffic.

A. Ground truth

In order to evaluate the accuracy of our classification system, we created a ground truth using real-world traffic. In particular, we use traffic from a large university network which includes a collection of 4,676,795 different (5-tuple) flows. More details about this trace are described in Table I.

TABLE I
Summary of the traffic trace used in our experiments.

# of flows	# of packets	Description
731,054 HTTPS flows 643,123 DNS flows 4,676,795 total flows	79,478,303 HTTPS packets 1,411,098 DNS packets 299,975,514 total packets	10 Gbps access link of a large Spanish university, which connects about 25 faculties and 40 departments (geographically distributed in 10 campuses) to the Internet through the Spanish Research and Education network (RedIRIS) Average traffic rate: 3.17 Gbps Date: 17th March 2017

We built our ground truth using the tool Bro IDS [11], which allows to process HTTPS traffic and extracts the hostnames from the SNI fields in the SSL/TLS certificates. Thus, we processed the whole trace (with 731,054 HTTPS flows) and Bro reported a total of 540,505 HTTPS flows with a valid SNI label. We consider that a label is valid when the SNI field fulfills the following conditions: (i) it has at least one character and (ii) it does not correspond to an IP address[3]. As a result, our ground truth is a collection of those HTTPS flows with a valid label. It includes the 5-tuple header fields that identify each flow together with their correspondent classification labels. This selection of labels allows us to properly evaluate and compare the server hostnames provided by the two classifiers presented in Section III.

B. Evaluation of the overall accuracy of our system

In this section, we evaluate the accuracy of our system classifying the HTTPS flows in the traffic.

We first recall that in OpenFlow-based networks, when operating in a reactive mode, packets belonging to the same flow are sent to the controller until a specific entry for them is installed in the switch. As a consequence, our classification system can receive more than one packet of each HTTPS flow. In particular, it receives packets during the time interval from the reception of the first packet of a HTTPS flow in the switch, to the time when a flow entry is installed in the switch to avoid receiving more packets. The factors that determine the duration of this time interval depend on specific features of the network scenario which were previously discussed in Section III.

For our experiments, we consider scenarios with a range from 1 ms to 100 ms for this elapsed time until a specific flow entry is installed in the switch. Thus, we use the real-world traffic trace described in Table I to simulate different values within this range of times. As a result, for each scenario, we only process in our system the first packets of each HTTPS flow that would be sent to controller. Note that, in case of the DNS traffic we forward all the packets to the controller. We

[3]This condition was imposed because we could observe some flows with certificates where the SNI field is an IP address, and it does not provide any valuable information to unveil the application generating the flow.

then show in section IV-D that the total amount of DNS traffic processed is quite reduced.

In our evaluation, we compare the labels that our classifiers provide in the different scenarios to those labels provided in the ground truth presented in Section IV-A. This allows us to compare the results actually obtained in our system with respect to the results that could be achieved if the controller processed every HTTPS packet in the traffic. In other words, the ground truth represents the results that could be obtained in a traditional network where all the packets are inspected in an external collector. Remark that in our scenario the controller only receives the first few packets of each HTTPS flow.

In order to evaluate the accuracy of our system, we only consider the second-level domains of the hostnames in the ground truth. That is, if the hostname is "www.netflix.com", we take "netflix" as the correct label. Thus, we consider that a classifier succeeds if it provides a label with the same second-level domain than the label of the ground truth.

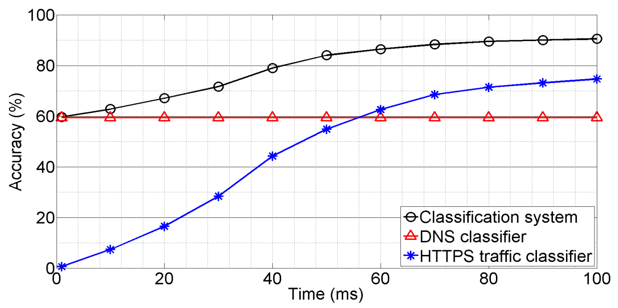

Fig. 2. Overall accuracy achieved by the classification system.

Fig. 2 depicts the results of accuracy obtained in our experiments. Thus, the y-axis represents the percentage of flows well classified with respect to the total number of HTTPS flows in the ground truth (540,505 flows). The x-axis represents the scenarios we simulated with different time intervals. This plot shows the accuracy achieved by our classification system as well as the results individually achieved by each of the two classifiers that compose the system (described in Sec. III). Note that, for the evaluation of the classification system, in case both classifiers provide a hostname label, we select the label provided by the HTTPS traffic classifier to compare it against the ground truth. Otherwise, if only one of the classifiers achieves a valid label, we take it to evaluate the accuracy against the ground truth. From these results, we infer that the HTTPS traffic classifier is quite sensitive to the value of the time interval. This basically reflects that, the more packets it receives from a flow, the more accuracy it achieves. However, the results achieved by the DNS classifier are not affected by the time interval of the scenario, as our system is always fed by all the DNS traffic. We can also observe that the combination of both classifiers in our system significantly improves the accuracy achieved individually by each of them in most of the cases.

As final remark, note that it is possible to extend the time interval fixed by a network scenario by delaying in the controller the execution of the order to install the flow entry in the switch. This would allow to further improve the accuracy achieved by the HTTPS classifier at the expense of a higher processing overhead in the controller, as it would have to process more packets. In Section IV-D we show the amount of traffic processed in the controller with respect to the time interval in different scenarios.

C. Evaluation of the accuracy for specific applications

In order to further analyze our system, we evaluated separately the accuracy targeting specific popular applications. In particular, we consider the domain names associated to the applications shown in Table II. To this end, we use again the trace in Table I and the ground truth described in Section IV-A.

TABLE II
APPLICATIONS EVALUATED IN OUR EXPERIMENTS.

Application name	Domain name
Google Drive	drive.google.com
Google Gmail	mail.google.com
Netflix	netflix.com
Whatsapp web	web.whatsapp.com
YouTube	youtube.com

In fig. 3, we show the accuracy results achieved by our classification system for the different applications. That is the percentage of flows well classified for each application with respect to the total number of flows generated by each of them in our ground truth. It is noteworthy that the results follow different patterns depending on the application. Thus, for example, our system achieves a high accuracy to identify flows from Netflix even in scenarios with short time intervals. However, other applications such as YouTube or Google Drive are quite more sensitive to the time interval.

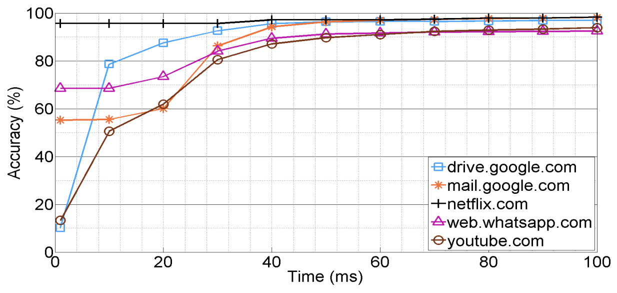

Fig. 3. Application-specific accuracy achieved by the classification system.

D. Evaluation of the overhead contribution of our system

Lastly, we evaluate the processing overhead contribution that would imply to run our classification system in a SDN controller. For this purpose, we use the real-world traffic trace in Table I to quantify the amount of HTTPS traffic that our system should process depending on the time interval of the scenarios we considered previously.

Fig. 4 shows the traffic processed by our classification system in terms of average number of packets per HTTPS flow and percentage of bytes processed with respect to the total amount of HTTPS traffic in the trace. Thus, for instance, we can observe that the percentage of bytes varies from less

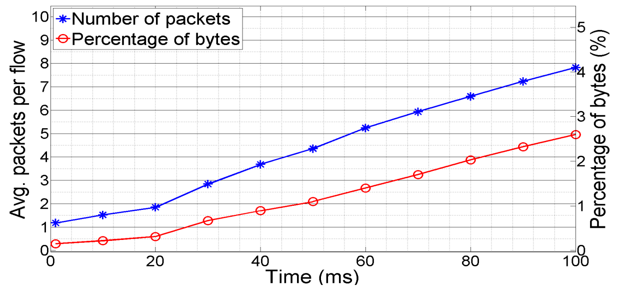

Fig. 4. Evaluation of traffic processed by the classification system.

than 0.4% for time intervals below 20 ms, to approximately 2.6% in the worst case with a time interval of 100 ms.

As for the DNS traffic, we could observe that, despite our system processes all this traffic, it does not represent a significant overhead for the controller. Thus, in the case of our real-world traffic, the DNS traffic involves around 0.08% of the total bytes in the trace, or 0.49% in terms of packets.

In any case, we can infer from these results that our design achieves a much more reduced overhead than in the trivial case in traditional networks where all the HTTPS traffic is forwarded and processed in an external collector. All this thanks to the management flexibility provided by OpenFlow.

V. Related work

We could find in the literature plenty of proposals based on Machine Learning (ML) [4] or Deep Packet Inspection (DPI) that have been traditionally used to classify the traffic. However, in nowadays Internet, where traffic is becoming more and more encrypted, most of these techniques become obsolete. On the one hand, it does not make sense to apply traditional DPI-based techniques, as the content in packet payloads cannot be inspected anymore. On the other hand, ML-based techniques are not sufficiently accurate, as they do not behave well using features at the transport layer to classify different applications generating web-based traffic (i.e., over the same port). Alternatively, other techniques like behavioral classifiers (e.g., BLINC [3]) rely on host profiling to classify the traffic. Nevertheless, they are not appropriate for current network scenarios as well, as nowadays many applications rely on shared infrastructures where the content is dynamically distributed (e.g., CDNs or cloud platforms).

In this context, some solutions were proposed to specifically address traffic classification over encrypted traffic [5] in traditional networks. Thus, authors in [6], propose to extract information from the SSL/TLS certificates in order to infer the applications generating encrypted flows. Other authors, like those in [12] or DN-Hunter [7], use the data in DNS queries to obtain the server domain names associated to encrypted connections. Likewise, more recent works as SFMap [8] or [2] use specific DPI techniques to specifically tackle traffic classification for web-based applications. However, we did not find in the state-of-the-art any contributions addressing how to implement these kind of classification techniques in SDN-based networks.

VI. Conclusions and future work

In this paper we presented a classification system compliant with OpenFlow which provides labels for each HTTPS flow in the traffic. Our system efficiently combines two classification techniques. One leveraging some information in the SSL/TLS certificates, and another using the data in DNS queries. We implemented our system and made experiments with real-world traffic to evaluate how feasible is to deploy these classification techniques in SDN-based networks. Our experiments were directed to measure the accuracy of our system as well as the processing cost to execute it in a SDN controller. Lastly, we conclude that OpenFlow permits to implement these classification techniques achieving a reduced processing overhead if we compare it with solutions in traditional networks, where typically all the traffic is mirrored to an external collector that processes every packet. As future work, we plan to use the labeled flow-level reports provided by our system as inputs to perform automatic fine-grained network management using Deep Learning models in SDN controllers.

Acknowledgement

This work was supported by the European Union's H2020 SME Instrument Phase 2 project "SDN-Polygraph" (grant agreement n° 726763), the Spanish Ministry of Economy and Competitiveness and EU FEDER under grants TEC2014-59583-C2-2-R (SUNSET project) and TEC2017-90034-C2-1-R (ALLIANCE project), and by the Catalan Government (ref. 2014SGR-1427).

References

[1] Sandvine, "Global Internet phenomena spotlight - Encrypted internet traffic," Feb. 2016.

[2] M. Trevisan, I. Drago, M. Mellia, and M. M. Munafò, "Towards Web Service Classification using Addresses and DNS," *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 38–43, 2016.

[3] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark," *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM*, vol. 35, no. 4, p. 229, 2005.

[4] T. T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.

[5] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25.5, pp. 355–374, 2014.

[6] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, "Towards self adaptive network traffic classification," *Computer Communications*, vol. 56, pp. 35–46, 2015.

[7] I. Bermudez and M. Mellia, "Dns to the rescue: Discerning content and services in a tangled web," *Proceedings of the IMC*, pp. 413–426, 2012.

[8] T. Mori, T. Inoue, A. Shimoda, K. Sato, S. Harada, K. Ishibashi, and S. Goto, "Statistical estimation of the names of HTTPS servers with domain name graphs," *Computer Communications*, vol. 94, pp. 104–113, 2016.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[10] 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. 44.3, pp. 87–95, 2014.

[11] "The Bro Network Security Monitor," https://www.bro.org/.

[12] Plonka, "Flexible Traffic and Host Profiling via DNS Rendezvous," *Workshop SATIN*, 2011.