Towards traffic classification offloading to stateful SDN data planes

Davide Sanvito, Daniele Moro*, Antonio Capone
Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
Email: {name.surname}@polimi.it, *daniele1.moro@mail.polimi.it

***Abstract*—Traffic classification allows network operators to gain important insights to better characterize packet flows, enabling fundamental applications such as traffic engineering, network analytics and Quality of Service (QoS) enforcing. A common approach adopted for flow classification is based on Deep Packet Inspection (DPI): all the traffic is processed by a middlebox whose task is the association of a network flow to the application-level information by inspecting the entire content of the packets. The increased volume of encrypted traffic limits the type of analysis performed by network middleboxes. However, an important amount of information can still be extracted from packets belonging to the very initial phase of a connection which are transmitted in clear (e.g. DNS and TLS handshake). Furthermore, recent research work has shown that it is possible to reduce the burden on the DPI without a significant loss in classification accuracy, by limiting the amount of data processed per flow. In this paper, we propose to exploit the programmability of new stateful SDN data planes to offload down to the network the process of filtering traffic to the DPI. We show that it is jointly possible to reduce the required computing power of the DPI, as well as the network bandwidth between the switches and the DPI. By taking advantage of the flexibility of stateful data planes we also manage to delegate to switches the computation of useful network analytics metrics (such as number of packets, number of bytes and duration) which would otherwise require the DPI to inspect the entire traffic flow.**

I. Introduction

The adoption of encrypted connections over HTTPS has undergone a remarkable boost in the last few years: in 2014 more than half of the overall traffic volume was encrypted [2], [3] and this trend is still increasing in last years [4]. While encryption used to be confined to handling sensitive transactions (involving the exchange of a small volume of traffic), nowadays HTTPS is normally used also for carrying multimedia content, including videos (for instance Facebook and YouTube have enabled HTTPS by default in 2013 and 2014, respectively). Even if this is fully justified by the growing concern about security, it challenges the network operators' need to get information on traffic statistics for several reasons ranging from traffic engineering to Quality of Service (QoS), network analytics, and security. Traffic classification, i.e. the process of associating each network flow to the application-level information, is commonly performed by specialized middleboxes implementing a Deep Packet Inspection (DPI) functionality, which analyzes the packets' payload to extract additional metrics to characterize the ongoing flow. However, the need of inspecting every single packet of each single flow poses a significant computational effort on the middleboxes.

This work has been partly funded by the EU in the context of the "BEBA" project [1] (Grant Agreement: 644122).

In the context of traffic analytics (e.g. understanding the volume and duration of different services), even if a big volume of traffic is encrypted, the establishment of a HTTPS connection requires the exchange of unencrypted data (DNS request and the SSL/TLS negotiation), fundamental for the operation of the protocol itself, which at the same time discloses significant information for the classification.

Based on these simple observations, and as also confirmed by recent research work (see Sec. V), it is reasonable to try to reduce the traffic volume analyzed by the DPI extracting only packets useful for classification and relevant flow statistics. Unfortunately, this may not solve the computational complexity problem if the process of packet filtering and flow statistics collection is performed by just another middlebox or the network controller.

In this work, we propose a DPI offloading mechanism for traffic classification based on a stateful SDN data plane in network switches. Our numerical analysis shows that it is possible to reduce the amount of traffic volume inspected by the DPI up to 98.6% without reducing classification accuracy.

This paper is organized as follows: Sec. II presents the motivations of our work. Sec. III describes the SDN architecture enabling in-switch stateful processing and the offloading application. Sec. IV shows the experimental results and Sec. V related works. Finally, in Sec. VI we draw our conclusions.

II. Motivations

The research community has so far put a lot of effort in optimizing traffic classification. Several solutions have improved the computational complexity of the most critical module of traffic classificator, but without lowering the amount of network bandwidth required to forward all the traffic to the DPI. Our proposal is that of exploiting the programmability offered by the SDN paradigm to offload down to the network the process of filtering traffic to the DPI. The goal is not to delegate the classification process, but rather to offload simple operations (such as counting and filtering) which can be efficiently performed at wirespeed by network switches. This allows classifying the same amount of traffic with less computational resources. Obviously, implementing the packet filtering logic as an application on the controller would not obtain the same result and would simply move the bottleneck on the controller itself, while our approach balances in-network processing with centralized traffic analysis.

Standard OpenFlow (OF) [5] devices are not flexible enough to allow flow rule updates and packet forwarding based on statistics without relying on the external controller. Our scheme is built on top of OPP [6], which introduces an architecture to program stateful applications within the datapath. The idea is to perform counting and filtering (fast, on a per-packet basis) in the network devices, while relying

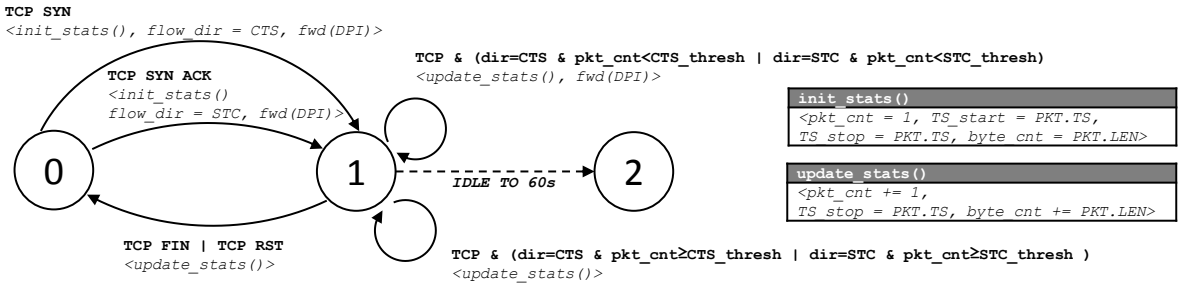


Fig. 1: Extended Finite State Machine (EFSM)

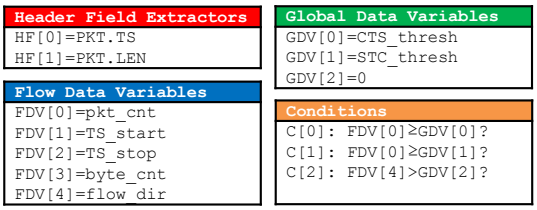


Fig. 2: EFSM parameters

on the controller for the final collection (slow, on a time scale in the order of flow lifetimes) of the statistics. Finally, the communication between the controller and the classifier eventually enables application-aware forwarding.

III. Proposed approach

Here we first briefly describe the stateful data plane provided by OPP and then present the DPI offloading approach as an OPP application.

Open Packet Processor (OPP). The OPP architecture allows programming stateful application inside SDN switches, by making them able to modify forwarding policies without relying on the controller. Each flow is associated with a persistent context including a state and data variables. OPP defines an abstraction based on Extended Finite State Machines: the evolution of the forwarding is described by a state machine where each state defines a forwarding policy and state transitions are triggered by packet-level events, time-based events or evaluations of conditions (computed on flow state and/or packet header fields).

OPP has been built as a stateful OF extension: stateful processing is enabled by adding a table (the flow context table) to keep track of the context of a flow: when a packet arrives, its corresponding context is extracted and conditions are evaluated. The state and the outcome of the conditions are available in the OF flow table as match fields. The user can thus express a flow-state-dependent forwarding and can update its context through new data plane actions. The controller specifies the set of header fields used to read/write the flow context table and configures the EFSM with an 1:1 mapping from state transitions to OF flow entries. This limited set of rules is shared by all the flows and their evolution is tracked in the flow context table. Those entries (1 per flow) are dynamically added and modified by the switch itself in the fast path. The viability of the OPP approach has been demonstrated with an HW proof-of-concept and a SW prototype [7].

DPI offloading application with OPP. In order to implement the DPI offloading as an OPP application, we have considered a pipeline made of two tables: first, packets are processed by a stateless table, which selects the output interface (for instance according to the IP prefix), and then by a second stateful table, which implements the actual offloading module. We are interested in counting the packets of each TCP flow and, based on that value, deciding whether to clone the traffic towards the DPI. For traffic analytics we need also the total number of packets and bytes of each flow and its duration. Without the access to the entire flow the DPI would be unable to measure such information. For the sake of clarity, we focus on the abstract EFSM description[1], rather than on low-level flow rules. The EFSM in Fig.1 contains 3 states: 0 is used for new flows or completed ones (after the reception of TCP FIN or RST), 1 for active flows and 2 for flows not properly terminated. The definition of the flow (e.g. the entity for which we are interested in keeping a state) is a TCP connection, thus the controller configures the lookup-scope and update-scope to the 5-tuple $IP\_proto|IP\_src|IP\_dst|TCP\_src|TCP\_dst|$. The flow context associated to each flow includes a state plus five flow data variables whose content is reported in Fig.2. All the flows are initially set to a DEFAULT state 0: as soon as we receive the TCP SYN[2] of a new flow, a new flow context table entry is created as a consequence of a state transition to 1. In addition, the first four flow data variables are initialized ($init\_stats()$ block in Fig.1), the last one is set to 0 (a constant value associated to the client-to-server, CTS, direction) and the packet is sent to the DPI. A similar initialization takes place when the TCP SYN ACK is received in the reverse direction: a distinct value in the initialization of the fifth flow data variable allows to consider direction-dependent thresholds. Flows in state 1 are associated with an idle state timeout, the dashed arrow: a flow from which we do not receive any packet within 60 sec moves to state 2. The controller periodically polls the flow context table to retrieve all the statistics related to flows in state 2 to make free space for new flows. Any subsequent data packet belonging to an established TCP connection matches state 1, is forwarded to the DPI and updates its own statistics ($update\_stats()$ block in Fig.1). This policy is retained until the number of packets of a flow exceeds the configured threshold: from that moment on its statistics still get updated, but packets are not cloned to the DPI. As soon as a TCP FIN or RST packet is received, statistics get their last update and a state transition to state 0 is performed. In addition, a state notification message is sent to the controller to report the statistics of the flow just completed. The notification is necessary to avoid loosing collected data due to the periodic flushing of expired state entry in the switch (i.e. entries in DEFAULT state). Statistics of the reverse

[1]State transitions are represented with arrows whose labels encode the matching condition (in bold) and actions (enclosed in angle brackets)

[2]OPP prototype supports matching on TCP flags and on the timestamp and the size of the packets

direction will be collected and deleted via the same push-based mechanism as soon as the TCP FIN or RST is received or by the periodic polling of the controller (pull-based mechanism).

As already described, the EFSM structure is mapped to low-level OF rules: the match portion is inserted in the $match$ part of the flow entry, while the forwarding actions and flow data variables update operations are put in the $action$ part, together with state update action of the arrow itself. The controller also configures conditions, global data variables and header field extractors whose content is reported in Fig.2. On the other hand states and flow data variables are stored and updated in the flow context table by the data plane itself at run time.

Comments. In the description above, we focused on the DPI offloading for TCP traffic, but it is straightforward to easily add another stateful table to handle also UDP traffic. Without this extension, we need to add a stateless high priority rule in the first table to forward all the DNS traffic also to the DPI: this rule helps the traffic classification and have been omitted just for the sake of clarity. The proposed solution is fully programmable: we can tune the thresholds bidirectionally, the value of the timeout and controller polling interval.

The flexible programmability provided by OPP is such that we might remove the thresholding component to obtain an in-datapath flow statistics collector in SDN mixing push and pull mechanism to retrieve the data. In addition, the decoupling of the forwarding from the counting/filtering allows a seamless integration with different forwarding need: we can for example modify the forwarding, in an application-aware fashion, as soon as the outcome of the classification is available at the controller, without interfering with the collection of the statistics.

IV. Experimental validation

The proposed approach has been validated by implementing the SDN application on top of the OPP software prototype [7] and the controller application is available as open-source software at [8].

Classification accuracy. A 12-h trace of domestic traffic (including mobile devices as well as PCs) was used to evaluate the impact of traffic filtering on the classification accuracy. We considered as the ground truth the outcome of the classification of the whole traffic trace. For the DPI engine, we selected nDPI [9], an open-source high-speed DPI library. We define the accuracy as the percentage of flows whose classification outcome does not change when we classify the filtered trace. For example, an accuracy of 100% means that traffic filtering did not impact at all the classification of the DPI.

Fig.3a shows the accuracy in 3 scenarios:

- Client To Server (CTS): the DPI receives a variable number of packets sent from the client[3].
- Server To Client (STC): the DPI receives a variable number of packets sent from the server[3].
- BOTH: the DPI receives a variable number of packets from the two directions (the same number for both)

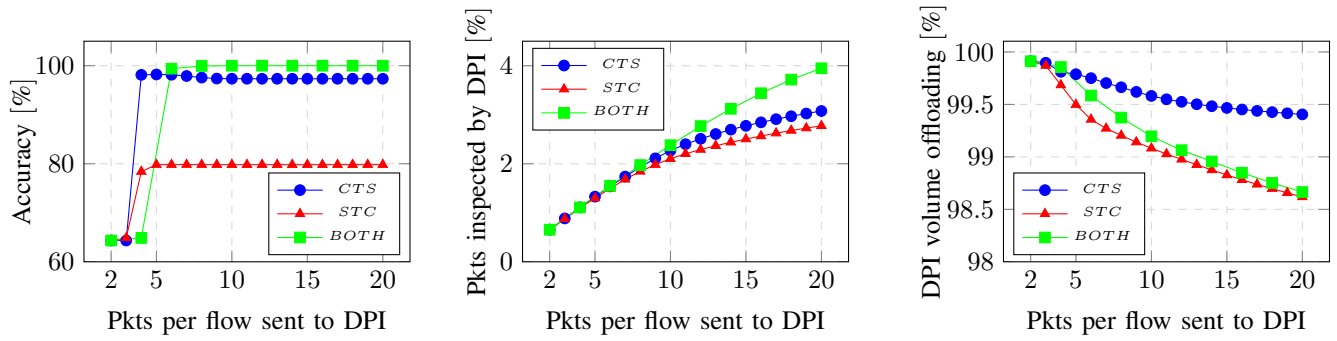
X-axis reports the number of per-flow packets processed by the DPI. In general, for equal number of packets, sending packets from both directions is better than favoring one direction. For example with 10 packets in "BOTH" case (5 per direction), we obtain an accuracy of 100% compared to 79% and 97% of the STC and CTS respectively. On the other hand, CTS outperforms STC starting from the fourth packet. This behavior happens because the 3rd packet of the CTS direction is the SSL Client Hello message which contains the Server Name field (in plaintext), and therefore is of remarkable importance for the flow classification. The results obtained confirm that analyzing a limited number of initial packets of TCP flows leads to a small loss in the classification accuracy: two packets are enough to reach 64% accuracy, while 5 packets are sufficient to reach 99.95% accuracy. A slight decrease in the accuracy curve is visible when sending 7 (or more) packets to the classification engine in the CTS scenario. This behavior is accountable to the fact that protocol signatures in the DPI were not designed for a scenario when only client packets are received. As a matter of fact, in this specific case, the statistical analysis conducted by the inspection engine misclassifies HTTP traffic as being P2P, upload dominant, applications.

Impact of traffic filtering. In this subsection we evaluate the impact of traffic filtering in terms of the total number of packets and bytes which have to be processed by the DPI. By looking at Fig.3a we can choose the required accuracy and then we can appreciate the resulting resource savings reported in Fig.3b. For example if we forward to the DPI 4 packets from each direction we can avoid processing 98% of packets of the whole traffic trace, while still obtaining a 99.9% accuracy. Fig.3c reports an analysis in terms of offloaded bytes. Apart from the huge traffic reduction, we can observe that a same number of packets sent to the DPI implies different volumes of bytes in the three scenarios. The gap between CTS and STC is due to the asymmetric nature of the client-server paradigm.

In addition to the analysis of our small scale traffic trace, we tested a 1h traffic trace made publicly available by CAIDA [10], monitoring a 10GigE Internet backbone link and comprising 2.7 TB of traffic from 56M TCP connections. The anonymization and payload removal prevent us to assess the classification accuracy, but we can still evaluate the impact of traffic filtering. Results are shown in Fig.4 by varying the number of packets sent to the DPI and by focusing only on the most promising scenario ("BOTH"). Despite not being able to evaluate the accuracy, we increased the maximum packet threshold from 20 up to 100 to show that, even with a threshold an order of magnitude greater, the resource saving is considerable. According to the distribution of the number of packets per flow (omitted due to lack of space) 95% of flows have no more than 40 packets: thus, by setting the threshold to 40 packets, even if the DPI processes entirely the 95% of the flows, it avoids 82% of traffic in terms of packets and 92% in terms of bytes. On average the DPI has to process a rate of 426 Mbps (127 kpps) rather than 5.7 Gbps (741 kpps) of unfiltered traffic.

Memory requirements. Our application requires just 8 EFSM table flow entries. On average, the CAIDA trace requires around 900K state entries. With respect to the first set of rules, this one is traffic-dependent. According to [6] an OPP ASIC prototype supports 256 EFSM table entries (in TCAM) and

[3]The first packet of the reverse direction is also sent to the DPI



(a) (b) (c)

Fig. 3: Classification results for a 12h domestic traffic trace

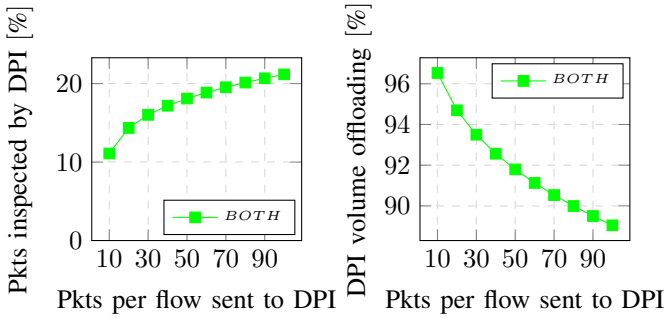


Fig. 4: Classification results for a 1h CAIDA traffic trace

1 million of entries for the flow context table (in SRAM), meeting our requirements[4].

V. Related works

Several works evaluated the effect of the traffic sampling on classification accuracy either for early stage classification or to limit the amount of traffic to be processed by the DPI. Sampling has been proposed and analyzed for different categories of classifiers, either based on the payload inspection [12], [13] or on the statistics of the packets [14]. The offloading mechanism we propose does not depend on the method adopted for the classification: we are interested in the offloading of the sampling logic down to the switches. Most optimizations operate on the DPI implementation itself, but with our approach we are able not only to reduce the computional load on the DPI box, but also the required network bandwidth. To the best of our knowledge the only work which addressed the same problem with a SDN-based approach is [15]. The authors propose a traffic classification solution based on OpenState [16], a stateful OpenFlow extension. With respect to our work, their approach is not completely decoupled from the controller (an interaction is still needed at the beginning of each new flow) and thus can suffer from controller processing load and latency. A difference in their solution is the ability to further reduce the number of packets sent to the classifier when the DPI sends a feedback. Altough we do not believe this would bring a significant added gain, we can trivially extend the EFSM to include such an optimization. A final advantage of our solution is the ability to compute in-data path statistics (number of bytes and packets and duration of the flow) useful for a posteriori analysis and that would not be available at the traffic classifier without processing entirely all the flows.

[4]Depending on the instantaneous number and duration of active connections and the frequency of the polling from the controller, the number of state entries available might not be enough. In this case we would require an architecture design [11] able to deal with the access to slower but larger memories.

VI. Conclusion

In this paper we have proposed and proved the feasibility of an approach for exploiting stateful SDN data planes to entirely delegate the filtering logic for traffic classification down to the switches. We have implemented it as an OPP application and we showed an interesting use case where all the information required for the application logic can be kept locally without relying on an external controller. The proposed approach is viable both for HW and SW implementations. Our numerical analysis shows that our solution offloads dramatically the DPI node, reducing the amount of traffic volume that the DPI should inspect up to 98.6% compared to a standard (inspect-all) setup.

References

[1] "BEBA project home page," http://www.beba-project.eu/.

[2] "Cisco's 2016 Annual Security report," http://www.cisco.com.

[3] D. Naylor *et al.*, "The cost of the "s" in https," in *ACM CoNEXT*, 2014.

[4] "Google Transparency Report (Dec 2016)," https://www.google.com/transparencyreport/https/metrics/.

[5] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM CCR*, 2008.

[6] G. Bianchi *et al.*, "Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing," *CoRR*, vol. abs/1605.01977, 2016. [Online]. Available: http://arxiv.org/abs/1605.01977

[7] "BEBA repository," https://github.com/beba-eu/.

[8] "Controller application repository," https://git.io/vSI9H.

[9] L. Deri *et al.*, "ndpi: Open-source high-speed deep packet inspection." in *IEEE IWCMC*, 2014.

[10] "The CAIDA UCSD Anonymized Internet Traces - Chicago 2015-02-19," http://www.caida.org/data/passive/passive_2015_dataset.xml.

[11] C. Cascone *et al.*, "Relaxing state-access constraints in stateful programmable data planes," *ArXiv e-prints*, Mar. 2017.

[12] S. Fernandes *et al.*, "Slimming down deep packet inspection systems," in *IEEE INFOCOM Workshops*, 2009.

[13] N. Cascarano *et al.*, "Improving cost and accuracy of dpi traffic classifiers," in *ACM SAC*, 2010.

[14] L. Bernaille *et al.*, "Early application identification," in *ACM CoNEXT*, 2006.

[15] T. Zhang *et al.*, "On-the-fly traffic classification and control with a stateful sdn approach," in *IEEE ICC*, 2017.

[16] G. Bianchi *et al.*, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM CCR*, 2014.