Demo: Simple Deep Packet Inspection with P4

Sahil Gupta*, Devashish Gosain†, Garegin Grigoryan‡, Minseok Kwon.*, and H. B. Acharya.*
*Rochester Institute of Technology, USA † Max-Planck-Institut fur Informatik, Germany ‡ Alfred University, USA
Email: *sg5414@rit.edu, * acharya@mail.rit.edu

***Abstract*—The P4 language allows "protocol-independent packet parsing" in network switches, and makes many operations possible in the data plane. But P4 is not built for Deep Packet Inspection – it can only "parse" well-defined packet headers, not free-form headers as seen in HTTPS etc. Thus some very important use cases, such as application-layer firewalls, are considered impossible for P4. This demonstration shows that this limitation is not strictly true: switches, that support only standard P4, are able to independently perform tasks such as blocking specific URLs (without using non-standard "extern" components, help from the SDN controller, or rerouting to a firewall). As more Internet infrastructure becomes SDN-compatible, in future, switches may perform simple application-layer firewall tasks.**

I. Introduction

Modern (software-defined) networks are flexible and powerful: they have been used for load balancing [1], [2], detecting network attacks (notably denial-of-service [3], port scans [4]), and simple (network-layer) firewalls [5]). It is natural to ask whether such "smart switches" can also perform more complex packet processing tasks – such as blocking particular URLs, session sniping (terminating attack traffic), and so on – reducing the need for heavy security infrastructure such as middleboxes, NIDS such as Snort, etc.

The fundamental building block for such tasks is Deep Packet Inspection (DPI), i.e., the ability to match patterns *inside a packet payload, and not just the (IP or TCP) header*. In this paper, we focus on an important specific case, which we call "simple DPI" – detecting the URL of a malicious website. In HTTP, the name is the "Host" field of the HTTP header in a GET request; in HTTPS, it is the "SNI" field in the TLS client hello request; and in DNS, it is the "query.name" field in the query request. Thus, any of these protocols can be used to detect (or deny) access to attack sites.

However, while specialized packet processors (such as nVidia DPU) can provide full DPI functionality, standard SDN switches – by which we mean, switches that support the OpenFlow and P4 standards – cannot. The P4 standard [6] explains that P4 allows users to specify how a switch should *parse* a packet: a user writes a P4 program to define a packet schema (and switches running this program slice matching packets into fields, following the schema). But such parsing *does not allow loops*. As a result, there is no "runtime flexibility" where a field can be matched if we do not know its position in the packet *a priori*.

Research Question: Our goal is to detect a specific field (site name) in three application-layer protocols: HTTP, HTTPS, and DNS. This cannot be done in *all possible* cases, as these protocols (as per RFC standards) are highly flexible, and allow (a) optional fields, (b) variable field ordering, and (c) variable-length fields. So our research question is, "is it possible (in *all practically significant* cases) to detect URLs of (malicious) sites, using only standard (P4 compliant) switches?"

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

II. Approach.

We note that the challenge in simple Deep Packet Inspection consists of two components: the variability in *start position*, and the variability in *length* of the pattern (i.e. URL) to match.

1) We suggest that *in practice*, the start position of the URL is usually consistent; the number of cases is small.
2) In order to deal with matching patterns of variable (finite) length, we can pad all patterns to the same (greatest required) length, if there is a "don't-care" symbol available.

To test (1) above, we conduct a survey, using the three most popular browsers (Google Chrome, Microsoft Edge, Mozilla Firefox), on different operating systems (Microsoft Windows 10, and Ubuntu 18.04 LTS) to access Alexa top-100 websites. Our original plan was to use Alexa top-1000, but we found 100 sites were sufficient to generate several thousand TLS and DNS connections (to CDN sub-domains, image servers, etc.)

For (2), we make use of the don't-care match function provided by TCAM in standard SDN switches.

III. Results.

A. Observational Study: URL Location and Length.

We consider the packet as a string, beginning with three sections (X, Y, Z) as shown in Figure 1.

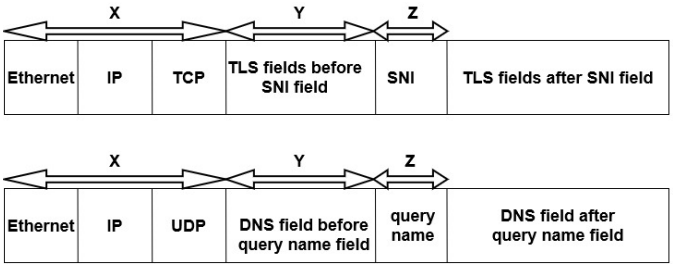

Fig. 1: Random variables in offset.

1) Variation in random variable X: X varies very predictably, based on three factors.

1) IP protocol: IPv4 or IPv6.
2) Transport layer protocol: UDP or TCP.
3) TCP with or without an options field.

This variability can be handled by the P4 parser, by having separate rules for each case. It is simple to identify the case (type of IP is a field in the L2 header; type of transport is the next-protocol field in IP header; TCP header length is exactly 12 bytes longer with an Options field; and the port field in transport header usually indicates the application protocol).

2) Variation in random variable Y: In our study Y was completely predictable for a given protocol – 13 bytes for DNS, 127 bytes for TLS (Figure 2).

Browser vs OS	Mozilla firefox	Google Chrome	Microsoft Edge
Windows 10	Total packets: 1743 Offset Y = 127 bytes	Total packets: 3216 Offset Y = 127 bytes	Total packets: 2812 Offset Y = 127 bytes
Linux Ubuntu 18.04 LTS	Total packets: 3230 Offset Y = 127 bytes	Total packets: 1307 Offset Y = 127 bytes	Total packets: 1432 Offset Y = 127 bytes

Browser vs OS	Mozilla firefox	Google Chrome	Microsoft Edge
Windows 10	Total packets: 512 Offset Y = 13 bytes	Total packets: 5587 Offset Y = 13 bytes	Total packets: 4237 Offset Y = 13 bytes
Linux Ubuntu 18.04 LTS	Total packets: 42238 Offset Y = 13 bytes	Total packets: 5075 Offset Y = 13 bytes	Total packets: 6777 Offset Y = 13 bytes

Fig. 2: Y is constant for a given protocol.

3) Variation in random variable Z: In alexa top 100, URLs varied in length from 10 ("www.qq.com") to 24 ("www.thestartmagazine.com"). Once the entire list of URLs to be blocked is known, the network operator specifies TCAM match rules where each URL is padded with don't-cares to the length of the longest one.

We will demonstrate with packet captures that the URL start location is stable, and length variation in Z is limited.

B. Experimental Study: Overhead of Simple DPI

We show that our assumptions are reasonable for practical cases (e.g., switch supports a high-level header definition with Y+Z bytes, and TCAM matching supports Z bytes), through a demonstration of our P4 application-layer firewall in action.

Using Mininet [7], we emulate a network: a P4 switch S1 (emulated using the standard BMV2 model) connects two hosts H1 and H2. H1 generates mixed censored and benign traffic to H2, which runs HTTPS, DNS, and HTTP services (i.e. masquerades as the target sites and the DNS server).

While this is a preliminary study, we can see that the overhead compared to baseline is quite small – in the slowest case (HTTP), the delay is a few milliseconds for 50 match rules on 10 parallel connections – *even in simulation using Mininet on a VM*. A real switch (e.g. Tofino 2) will improve performance by orders of magnitude.

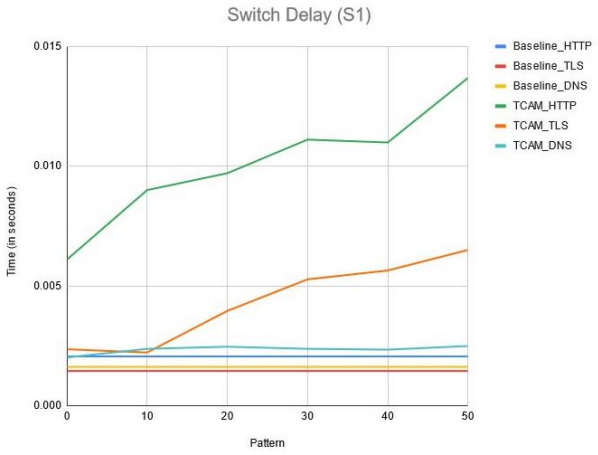

Fig. 3: Switch Delay Vs. Number of Rules.

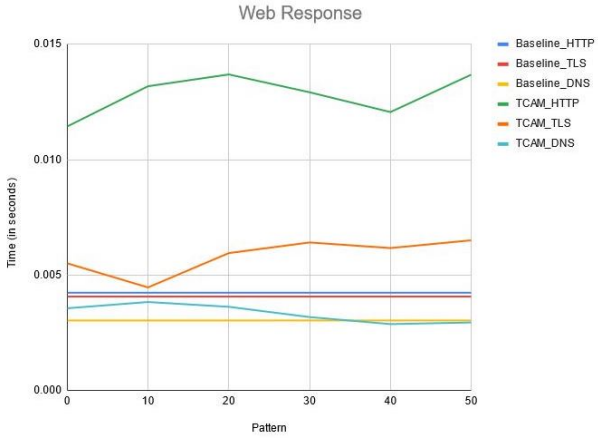

Fig. 4: Total Response Time Vs. Number of Rules.

IV. Concluding Remarks

Our demonstration shows that in practice, simple DPI can be performed by standard (P4-compliant) switches. Two issues still remain.

- A very short packet, where the *entire packet* is shorter than the padded match pattern with don't-cares, will slip through the firewall.
- We are yet to demonstrate simple DPI on a real switch, for a direct comparison with existing middleboxes.

We are developing a more advanced approach to address the first issue. We invite collaborators, who are willing to share access to an actual switch, to help address the second.

References

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

[2] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2890955.2890968

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

[4] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A84–A95, 2019.

[5] K. Zhang, D. Zhuo, and A. Krishnamurthy, "Gallium: Automated software middlebox offloading to programmable switches," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 283–295. [Online]. Available: https://doi.org/10.1145/3387514.3405869

[6] M. Budiu and C. Dodd, "The p416 programming language," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, pp. 5–14, 2017.

[7] B. Lantz and B. O'Connor, "A mininet-based virtual testbed for distributed sdn development," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 365–366. [Online]. Available: https://doi.org/10.1145/2785956.2790030