NETWORK SECURITY WITH SMART SWITCHES

by

Sahil Gupta

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences

Rochester Institute of Technology
Rochester, New York
May, 2023

NETWORK SECURITY WITH SMART SWITCHES

by

Sahil Gupta

Committee Approval:

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

Dr. Hrishikesh B. Acharya Date
Dissertation Advisor

Dr. Yin Pan Date
Dissertation Committee Member

Dr. Minseok Kwon Date
Dissertation co-advisor

Dr. Sumita Mishra Date
Dissertation Committee Member

Dr. Bruce Hartpence Date
Dissertation Defense Chairperson

Certified by:

Dr. Pengcheng Shi. Date
Ph.D. Program Director, Computing and Information Sciences

NETWORK SECURITY WITH SMART SWITCHES

by

Sahil Gupta

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

The inspection of packet contents, Deep Packet Inspection (DPI), is an important component in network security. However, DPI is provided by complex black-box firewalls which the network administrator has no choice but to trust. This raises the question: Can network administrators build their own DPI-capable filter using a standard programmable switch?

The commonly-accepted answer is that standard switches are not powerful enough; the standard they support (the P4 language) does allow users to specify how to parse packet headers, *but not packet payload fields (e.g. URL)*, as required by DPI. Even though software-defined networks are quite capable of handling various tasks, ranging from firewalling to flow analysis, these are all based on intelligent use of packet headers. DPI tasks, like URL filtering, still require dedicated middleboxes – or, if we insist on SDN solutions, middleboxes *in addition to* SDN. If we insist on developing a solution on the switch itself, we need either custom switch hardware, or heavy support from the SDN controller or an external firewall.

This dissertation challenges this common consensus. For our first contribution, we demonstrate that clients send packets with a predictable structure, so a P4 switch *can* perform some DPI (enough for URL filtering). We then develop and demonstrate a URL-filtering firewall, DiP, completely in the data plane, taking no external help from the SDN controller, firewalls, etc. DiP is a proof-of-concept, but is quite robust, handles multiple protocols (HTTP(S), DNS), and outperforms standard netfilter firewall by orders of magnitude.

However, DiP is not truly a general firewall: it is very specifically a URL filter, and it depends on the strong constraint of predictable URL location in a packet, which may not hold in future. Thus for our final contribution, we present a novel approach that allows *general* Deep Packet Inspection

(DPI) – i.e. inspection of the packet payload – in the data plane, using P4 alone. We make use of the fact that in P4, a switch can clone and recirculate packets. One copy (clone) can be recirculated, slicing off a byte in each round, and using a finite-state machine to check if a target string has yet been seen. If the target string is found, the other copy (original packet) is discarded; if not, it is passed through.

Our approach allows us to build DeeP4R, the first general-purpose application-layer firewall (URL filter) in the data plane, and to achieve essentially line-rate performance while filtering thousands of URLs, on a commodity programmable switch. We can therefore argue with assurance that any platform that supports P4 is powerful enough for Deep Packet Inspection, and in future it may be possible to use programmable switches for this task, rather than dedicated firewalls.

Acknowledgments

The completion of this dissertation and the research behind it would not be possible without the guidance, support, and encouragement from many individuals. I would like to take this opportunity to express my earnest and heartfelt gratitude towards them.

First and foremost, I would like to give special thanks to my advisor, Dr. H.B. Acharya, for his constant support and mentorship throughout my Ph.D. tenure at Rochester Institute of Technology. He has taught me how to become a good and effective researcher. The meaningful discussions that we have shared during my research have been truly inspirational to me.

I would also like to express my sincere appreciation to my Ph.D. dissertation committee members, Dr. Miseok Kwon, Dr. Sumita Mishra and Dr. Yin Pan. Without their critical observations, suggestions and thoughtful feedback it would not have been possible to come up with all the solutions to the research questions answered in this dissertation. I thank them from the bottom of my heart for managing time from their busy schedule for all of my research review meetings. I would also like to thank Dr. Bruce Hartpence for serving as my dissertation defense chair. I would also like to thank Dr. Pengcheng Shi for the support throughout my Ph.D. tenure. A very special thanks to Lorrie Jo Turner and Min-Hong Fu who helped me a lot by taking care of all the administrative task related to my Ph.D. life and let me focus more on my research work. I would like to thank my friends and lab mates Garegin Grigoryan , Gaurav Wagh, Payap Sirinam, and others for their support and encouragement throughout this entire Ph.D. journey. I would like to express warm thanks to Dr. Devashish Gosain, of Max Planck Institute, for his help in every step of the journey. I also thank Dr. Nate Foster of Cornell University, Vladimir Gurevich (from Intel), Andy Fingerhut (from Intel), open source community of p4.org and ICRP community of Intel, for their help in learning about the P4 platform and data plane programming. Finally, I am grateful to have wonderful friends and family who supported me emotionally throughout my journey at RIT.

Contents

1 Introduction **1**

2 Background: P4 and Programmable Switches **6**

3 Related Work **10**

3.1 Programmable Data Plane and the P4 Platform 10

3.2 Software-Defined Network Security 11

3.3 Deep Packet Inspection with P4 11

3.4 Immediately Related Work 12

4 Domain Name Field Study (Predictable Internet Clients) **13**

4.1 Model of Protocol Packets. 13

4.2 Field Study and Observations. 14

4.3 Choosing Start and End Positions. 15

5 DiP: Simple In-Switch Deep Packet Inspection **17**

5.1 Overview : How DiP Works 17

5.2 Challenges : DPI in the Data Plane 18

5.2.1 Challenge 1. Parsing 18

5.2.2 Challenge 2. Platform Constraints 19

5.3 System Design and Implementation 19

5.3.1 Deep Packet Inspection with P4 19

6 DeeP4R: Deep Packet Inspection in P4 using Packet Recirculation. 22

6.1.1 Architecture of Deep4R. 22

6.1 System Design 23

6.1.2 Processing of a Packet. 25

7 Experimental Setup. 28

7.1 Testbed Layout 29

7.2 Testing Workflow 30

7.2.1 Switch Setup: DiP 30

7.2.2 Switch Setup: DeeP4R 31

7.2.3 Traffic source and sink 31

7.2.4 Routing 32

7.2.5 Firewall Setup 32

7.2.6 Measurement Collection 32

8 DPI-in-P4 (DiP): Experimental Results. 33

8.1 Evaluation 33

9 Deep4R: Experimental Results. 38

9.1 Experimental Results 38

10 Discussion: DiP and DeeP4R. 44

10.1 DiP : Analysis and Limitations 44

10.2 DeeP4R: Analysis and Limitations 47

10.3 DiP and DeeP4R: working together? 49

10.4 Our Work In Context 50

10.4.1 P4-based Network Security, DiP and DeeP4R. 50

10.4.2 DiP and DeeP4R: Impact. 51

11 Concluding Remarks. 53

List of Figures

2.1 4 Pipe Tofino ASIC [1] 7

4.1 Partition of DNS packet. 14

4.2 Partition of HTTP GET packet. 14

4.3 Partition of TLS Client Hello Packet. 14

5.1 DiP: packet parsing and matching in the switch.
First, the packet arrives at the Ingress port. Next, the (ingress) parser separates different headers (eg. TCP), and particularly the user-defined header containing the URL. Finally, the control block matches the user-defined header field to see if there is a rule to drop it. On a successful match, the entire packet is dropped. 20

6.1 DFA to match evil.com and bad.com. 23

6.2 Life cycle of a packet processed in Deep4R. The *supervisor* table actions correspond to the decision blocks, and the *DFA* table corresponds to the bolded block "update status". 23

7.1 Experimental setup: Client machine fetches HTTP or HTTPS traffic (web pages, including large files) from Server. In separate runs, we pass identical traffic through our Tofino-based switch (running Dip / Deep4R), and through a server running Netfilter firewall, for a fair comparison. 29

8.1 Packet processing time: Avg time a packet spends within firewall. 34

8.2 Packet processing time (log scale): With 10k flows through firewall. 34

8.3 Queue occupancy: Under heavy cross-traffic (i.e. 10k flows), increasing firewall rules do not impact queue occupancy. 35

8.4 Impact of packet size on packet processing time. 36

8.5 Throughput (log scale): Impact of increasing firewall rules. 37

9.1 E2E Delay vs Filtered Domains. 39

9.2 Device Delay vs Number of Filtered Domains. 40

9.3 E2E Delay vs Parallel Flows. 41

9.4 Device Delay vs Parallel Flows. 41

9.5 Dropped Packets vs Parallel Flows. 42

9.6 Impact of increasing firewall rules on Throughput. 43

10.1 Device Delay vs Packet Size. 48

List of Tables

4.1 URL position in packets, as Min Start – Max End. 15

4.2 Parsing and Pattern-matching Accuracy, Alexa top-10k sites (using the given start and end positions to extract URL). 16

6.1 Our example DFA, expressed in match-action table rules. 1 is the start state. 11 is the only accept state i.e it indicates that the URL was seen. Note that the last transition from 10 to 11 not only updates the state in the label header, it also writes to `decision` – hence the 1 in bold. 25

Chapter 1

Introduction

This thesis focuses on the use of software-defined network (SDN) switches for deep packet inspection (DPI). We begin with this introductory chapter, describing what these terms mean, why it might be important to perform DPI purely in the data plane, and what our contributions are, ending with a brief summary of the structure of the dissertation.

Software Defined Networks (SDN) are networks with flexible switches. The switches have match-action tables called *flow tables*, which allow them to operate on packets, and these tables can be written to the switch on-the-fly using standard protocols (such as OpenFlow). The ability to configure flow tables, and thus the logic of packet processing, is decided by a separate machine or process called the SDN controller. The main value-add of SDN is that the packet processing logic is flexible, and under the control of the network admin using a standard API (in contrast to closed switches, which can only be configured using manufacturer-specific configuration languages).

Switches and routers – particularly Software-Defined Network (SDN) switches – have been successfully used to implement network-layer firewalls [57], flow analysis [16], and a wide range of other functions. Part of the reason for this remarkable versatility is that a small number of packet headers (source IP, source port, destination IP, destination port, protocol, etc.) are key for a variety of networking tasks. However, more advanced techniques, such as the detection of malicious traffic or malware signatures, require *Deep Packet Inspection* (DPI), i.e. the inspection of packet payloads and not just packet headers. For example, a Network Intrusion Detection System (NIDS) needs DPI to identify if a packet carries the signature of an attack such as Heartbleed [21].

The current state-of-the-art in DPI is still provided by old-school dedicated middleboxes, such as Cisco Firepower Threat Defense [2], SonicWALL TZ/NSA/SuperMassive Series [11], Fortinet FortiGate [3] , etc. These solutions treat the network administrator as a *consumer* – the admin has no option other than to trust the manufacturer for strong security guarantees (i.e. that the firewall is not itself malicious [14], does not violate user privacy, etc). Further, such middleboxes are usually on-path rather than in-path [61], and may only inspect a sample of traffic so as not to become a bottleneck. A comprehensive line-rate filtering solution is very expensive, and even modest firewalls may be out of the reach of small businesses. Such lack of access was one of the original motivations for developing Software-Defined Networks [27]. And finally, such firewalls are not only black boxes taken on trust by the network administrator; they are hard to audit, present a high-value target for attacks, and compromise many users when they leak.

It is interesting that Deep Packet Inspection is *not* implemented using programmable switches, when there already exists a standard language (P4) that allows users to specify packet schemas[1]. Naively, this should imply that a programmable switch can parse packets and extract fields from HTTP, TLS, etc., headers. As soon as a switch can (extract and) filter traffic by, e.g., site URL or file type, it becomes an application layer firewall. What is the reason that such solutions do not replace (or at least compete with) black-box firewalls?

In the early days of SDN, researchers did propose such ideas – for example, Sekar's CoMB architecture [56] built on the Click modular router [42]). But *current SDN platforms are not intended for Deep Packet Inspection*[2]. The *P4*$_{16}$ standard makes this explicit [19].

- P4 is not a Turing-complete language; the P4 packet "parser" really just extracts slices of bits ("slice" meaning, a given length at a given offset). The parser cannot loop, and cannot properly handle the following cases:
  - Fields of variable length.
  - Fields which may or may not be present.
  - Fields present in random order.
- The above cases are required to parse headers of important application-layer protocols, such as HTTP. (HTTP has 47 fields, which are mostly optional; important fields for filtering, such as URL, are variable-length).

[1] In a P4 program, the user defines the structure of packets of a protocol. A switch loaded with the appropriate definition can parse headers of novel protocols just like TCP or IP headers, but *subject to some restrictions*, as we discuss in detail below.

[2] Most likely this decision was made to ensure that complex parsers do not slow down packet processing.

Thus while P4-compatible switches have some flexibility, *it is not straightforward to use them for general DPI.* If a network admin wishes to build their own DPI-capable infrastructure, the consensus is that they must *either* use specialized platforms – eg. nVidia DPU [7], custom switches with non-standard extensions (`extern` logic implemented on NetFPGA), etc – *or* they can have the switch outsource some work to an external server [34], and provide enough servers to process traffic at line rate. This is a very different proposition than adding some off-the-shelf programmable switches to the network, and it is hardly surprising that the admins of enterprise networks and ISPs prefer to invest in a standard commercial middlebox.

At this point, we make an important observation. *An application-layer firewall can be valuable even if it only performs a few simple cases of DPI.* More involved DPI, such as content censorship (eg. social media or email) is usually performed with the help of an end-point on the provider or the client; for the common case in traffic inspection – blocklisting of websites – it is sufficient to detect the URL. And the URL is usually present in plaintext in HTTP traffic, in HTTPS traffic (the Server Name Indication field), and in DNS traffic. If it is present *at a predictable location* in network packets, then this common case of DPI can indeed be solved.

We now come to our first contribution.

- In chapter 4, we report on our field study, which shows that even for theoretically "free-form" protocols such as HTTP(S), the header has a predictable structure in actual web traffic. Hence, these protocols *can* reliably be parsed in the data-plane, and the domain name extracted.

  In other words, even simple SDN switches (not designed for DPI) can perform URL filtering in practice, thanks to the predictability of browser clients, and the low rate of adoption of more-secure protocols such as encrypted SNI and DNS-over-TLS.

Our study shows that even if it is challenging to build a *fully general application-layer firewall* in the data plane, it may be possible to build a *URL filter for traffic from practical Internet clients.* This immediately raises the question of how such a (limited) firewall can be implemented, and what its performance would be like. This brings us to our second contribution.

- We develop DPI-in-P4 (DiP), a dataplane firewall capable of simple deep packet inspection, on a real, cheap, easily-available SDN switch (Netberg Aurora 710) [5] [3]. DiP works with

[3]Our implementation runs on the Intel P4-based Tofino ASIC [4], but it can be ported very simply to another