Domain Name Security Inspection at Line Rate: TLS SNI Extraction in the Data Plane Using P4 and DPDK

Ali Mazloum*, Ali AlSabeh†, Elie F. Kfoury*, Jorge Crichigno*
*Molinaroli College of Engineering and Computing, University of South Carolina (USC)
†College of Sciences and Engineering, University of South Carolina Aiken (USCA)
Emails: amazloum@email.sc.edu*, ali.alsabeh@usca.edu†,
ekfoury@email.sc.edu*, jcrichigno@cec.sc.edu*

***Abstract*—A widely adopted approach to monitor HTTPS traffic leverages the Server Name Identification (SNI) extension of TLS. Generally, the hostname is transferred in plain text over the SNI field and Deep Packet Inspection (DPI) is used to parse the TLS header and extract the hostname. However, DPI is often performed on general-purpose processors and utilizes the kernel of the operating system, which results in an overhead to the network, especially under high traffic loads. To this end, this paper proposes offloading the identification of SNI hostnames to the data plane using P4 and the Data Plane Development Kit (DPDK). In the proposed system, a P4 Programmable Data Plane (PDP) switch is the first line of defense where most of the TLS traffic is processed. DPDK is the second line of defense which processes all TLS packets that require processing capabilities beyond what the P4 PDP switch provides. To support line rate pattern matching on the hostname, the DPDK application is offloaded to a SmartNIC, leveraging its Regex engine. Experiments on various recent and public datasets from different regions and platforms reveal that the P4 switch is capable of parsing 85%-99% of hostnames. Furthermore, performance analysis shows that the P4 switch and the DPDK application, respectively, inspect a hostname in around 1 microsecond (μs) and 7 μs, achieving an order of magnitude improvement over solution running on general-purpose processors.**

***Index Terms*—TLS, SNI, HTTPS, P4 programmable data plane switches, high-speed network monitoring, DPI, hardware acceleration, software acceleration, smartNICs**

I. Introduction

The HyperText Transfer Protocol (HTTP) is the foundation of Internet browsing, facilitating the exchange of information between web browsers and servers. To secure this communication, HTTP Secure (HTTPS) was introduced, which encrypts data and ensures the secure identification of web servers over a network. Since 2014, the adoption of HTTPS has surged, with Google Transparency Report data showing an increase in encrypted traffic from 50% in 2014 to 96% in 2024 [1]. Additionally, Google's investigation of the top 100 non-Google websites reveals that all of them support HTTPS, with 97% using it as the default protocol.

At the heart of HTTPS is the Transport Layer Security (TLS) protocol, which provides the necessary cryptographic layer for secure communication [2]. TLS not only enables traffic encryption but also supports a variety of extensions that are crucial for establishing HTTPS connections. One key extension is Server Name Indication (SNI) [3]. SNI is essential as a single physical server can host multiple virtual servers. Each virtual server has a unique certificate that is used during the encryption process. SNI allows the client to specify the virtual server to which it intends to connect by providing the server's hostname. When a server receives a TLS request, it processes the SNI extension to determine which certificate to use.

Since the SNI is transmitted during the TLS handshake, it is typically unencrypted. Modern security systems often use the SNI field to monitor HTTPS traffic, relying on Deep Packet Inspection (DPI) to do so. DPI, which operates on general-purpose Central Processing Units (CPUs) and Graphics Processing Units (GPUs), faces two major challenges: (1) it can become a bottleneck under high traffic rates [4], and (2) it raises privacy concerns [5]. To overcome these issues, multiple approaches have offloaded DPI to the data plane [4–7]. By running on the data plane, DPI is guaranteed to run at line-rate without any potential evasion of privacy.

Programmable Data Plane (PDP) switches are a main target of DPI applications. PDP switches combine the flexibility of software and the performance of hardware [8, 9]. P4, or Protocol-Independent Packet Processors, is a widely used language to program PDP switches. The proposed system leverages the flexibility and processing capabilities of the P4 PDP switches to offload TLS SNI inspection to the data plane. The P4 application is capable of extracting the server name and enforcing fine-grain and coarse-grain security policies. While the P4 application can handle the majority of TLS packets, the hardware resource constraints and the variable length of TLS structures prevent it from processing all packets. To overcome this limitation, the proposed system utilizes the Data Plane Development Kit (DPDK) technology to handle packets not processable by the P4 application. DPDK is a software acceleration framework that significantly enhances packet processing performance [10]. The deployed DPDK application runs on the data plane of a smart Network Interface Card (smartNIC). SmartNICs are specialized network cards with dedicated computational resources and hardware acceleration engines [10]. The DPDK application leverages the Regex hardware accelerator to enforce security policies. The contributions of this paper can be summarized as follows:

- Developing a P4 application that extracts the TLS SNI extension and enforces security policies at line rate.
- Overcoming the resource limitations of P4 PDP switches by incorporating DPDK technology.
- Using smartNIC hardware accelerators to enforce coarse-grain policies in DPDK.
- Significantly increasing DPI processing speed compared to methods running on general-purpose processors.

II. Related Work

The significant performance gains achieved by implementing DPI in P4 PDP switches have led to many DPI applications being offloaded to the data plane. Typically, P4-based DPI applications that are offloaded to the data plane use one of two approaches to address the hardware resource constraints of PDP switches.

The first approach limits the number of header fields to be parsed based on the hardware resources available in PDP switches. This method prioritizes maintaining line-rate packet processing. For example, Kim et al. [5] propose Meta4, a framework that monitors traffic by inspecting the DNS domain name within the data plane. In this approach, the parser is limited to extracting four subdomains, with each subdomain containing up to 15 characters. Similarly, Kaplan et al. [11] focus on performing DPI on DNS packets but enhance the parser to handle six subdomains with a total of 60 characters.

The second approach takes advantage of the recirculation feature in PDP switches to parse all header fields of interest. Recirculation allows the PDP switch to process the same packet multiple times. With each pass, a portion of the packet is parsed, and the packet is truncated before the next iteration. While this method removes the constraint on the number of header fields that can be parsed, it comes at the expense of line-rate processing. Alsabeh et al. [4, 7, 12] use packet recirculation to parse DNS packets of arbitrary length and extract all subdomains. Jepsen et al. [13] propose a system that searches the packet payload for specific string patterns [14]. This system converts a set of patterns into partitioned Deterministic Finite Automata (DFA), enabling parallel searches with packet recirculation. However, the major drawback of these approaches is the high processing time.

This paper proposes a novel system that combines both approaches to achieve line-rate DPI of TLS packets without restricting the number of characters that can be parsed. The hostname extracted from the parsed packets is used to enforce monitoring and security rules on the encrypted traffic.

III. Background

A. TLS

Transport Layer Security (TLS) is a cryptographic protocol that provides end-to-end security for data transmitted over an insecure network. TLS supports multiple extensions to facilitates the end-to-end communication. One of these extenstions is the Server Name Indication (SNI). When a client initiates a connection to a server, it sends a "Client Hello" message as part of the TLS handshake process. This message includes the SNI field, which contains the hostname of the desired website. The server then examines the SNI field to determine which certificate to present to the client. This process is necessary as a single server can host multiple TLS certificates. Almost all browsers, web servers, and operating systems support the SNI extension, which is also utilized by several security applications to monitor HTTPS traffic [15, 16].

B. Programmable Data Plane Switches and P4

Programmable Data Plane (PDP) switches are network devices that allow the user to define the packet processing logic in the data plane. P4, or Programming Protocol-Independent Packet Processors, is a widely adopted language to program the PDP switches. The main three components of the P4 PDP switches are the programmable parser, the programmable match-action pipeline, and the programmable deparser. The programmable parser of the P4 PDP switches is the key enabler for DPI. The parser operates as a finite-state machine. At each state, the parser either extracts a portion of the packet headers and stores them in pre-defined header structures or skips a pre-defined number of bits. Although this approach is suitable for parsing protocols with fixed length headers (e.g., parsing Ethernet or IP protocols), it imposes multiple challenges when parsing protocols with variable length headers. After parsing the headers, the programmable pipeline executes the packet processing behavior implemented by the programmer. Finally, the programmable deparser reassembles and forwards the packets. P4 has been used for diverse DPI applications such as enhancing QoS for media traffic [17], classifying malware in real time [18], and others [8].

C. Data Plane Development Kit and SmartNICs

Data Plane Development Kit (DPDK) is a software-based acceleration technique that enhances packet processing efficiency [10]. It provides a set of libraries and drivers to bypass the kernel. DPDK applications run on the userspace and have direct access to the ports of the Network Interface Card (NIC). Because of the kernel bypass, DPDK applications should implement all the packet processing functionalities required by the host (i.e., pulling packets from the NIC, parsering and processing the packets, and then taking a forwarding action). Besides running on the userspace of the server, DPDK applications can run directly on the computing units of smartNICs, allowing security and monitoring applications to be deployed on the smartNICs. The applications can also utilize the hardware accelerators provided by the smartNICs to further enhance the packet processing efficiency.

IV. TLS SNI Packet Processing in the Data Plane

A. System Overview

The proposed system is presented in Fig. 1. The P4 PDP switch identifies the TLS Client Hello packets and tries to extract the SNI hostname. If the extraction is successful, the

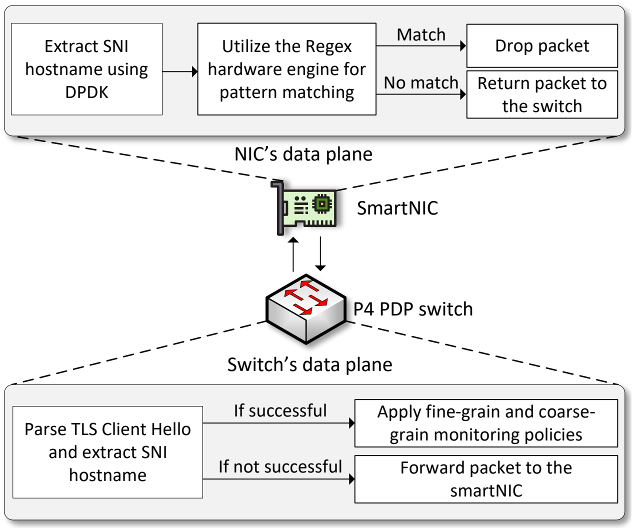
Extract SNI hostname using DPDK
Utilize the Regex hardware engine for pattern matching
Match
Drop packet
No match
Return packet to the switch
NIC's data plane
SmartNIC
P4 PDP switch
Switch's data plane
Parse TLS Client Hello and extract SNI hostname
If successful
Apply fine-grain and coarse-grain monitoring policies
If not successful
Forward packet to the smartNIC

Fig. 1: Overall architecture of the proposed system.

switch enforces fine-grained and coarse-grained monitoring and security policies on the hostname. The policies are managed at run-time from the control plane of the switch. On the other hand, if the switch fails to extract the hostname, it forwards the TLS Client Hello packet to the DPDK application hosted on the smartNIC. The application is capable of parsing and extracting the hostname from any TLS Client Hello packet. After extracting the hostname, the DPDK application applies the monitoring and security policies. The application utilizes the Regex engine of the smartNIC to apply the coarse-grain policies. If the hostname matches a security policy, the DPDK drops the corresponding packet. Otherwise, the packet is sent to the P4 PDP switch, which forwards the packet with no further inspection.

B. Parsing TLS in P4

The TLS protocol contains multiple variable-length headers. In the TLS Client Hello packet, the variable-length headers include the session ID, the list of supported ciphers, the compression methods offered by the client, and the list of TLS extensions. For each of these headers, the parser first extracts the length field and then uses the extracted length to transition to the corresponding state. For each possible transition (i.e., for each potential header length), the parser defines a separate state to parse a number of bytes equal to the length of the header.

Due to hardware resource constraints, it is not feasible to account for all possible header lengths and transitions. Instead, the parser defines states for the most commonly observed header lengths, maximizing the likelihood of successful parsing. The *advance* feature of the Tofino PDP switch is used to skip over all headers prior to reaching the SNI field. This feature is critical for achieving line-rate extraction of the hostname in the data plane. After skipping over the session ID, ciphers, and compression methods, the parser arrives at the TLS extensions. TLS supports various extensions, including the SNI extension, each of which is identified by its type and can have a variable length depending on its content. A loop is implemented in the parser to efficiently skip over these TLS extensions.

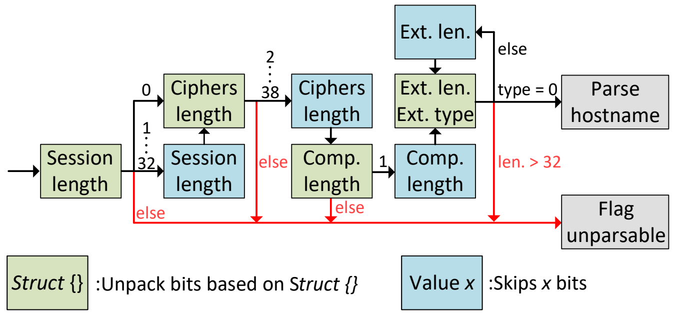

Fig. 2: Parsing TLS protocol in P4.

The parsing logic is illustrated in Fig. 2. Upon reaching the TLS protocol, the parser first extracts the session ID length. If the session ID length is zero, indicating the absence of a session ID, the parser transitions to the state responsible for extracting the length of the ciphers. If the session ID length is less than 32 bytes, the parser uses this value to transition to one of 32 predefined states, with each state responsible for skipping a number of bytes equivalent to the session ID length. After skipping the session ID, the parser transitions to the state responsible for extracting the length of the ciphers. If the session ID length exceeds 32 bytes, the parser flags the packet as *unparsable* and accepts it. All packets flagged as *unparsable* are forwarded to the DPDK application for further processing. The parser then follows the same logic to skip the ciphers and the compression methods (denoted as "comp." in the figure). It defines 36 states to handle different cipher lengths and a single state to parse the compression methods.

Once the parser skips the compression methods and reaches the list of TLS extensions, it implements a loop to skip over all extensions except the SNI extension. If the parser encounters an extension larger than 31 bytes, it flags the packet as *unparsable* and accepts it. The loop terminates when the parser identifies an extension of type 0, which corresponds to the SNI extension. Once the SNI extension is reached, the parser begins extracting the hostname.

Unlike other header fields in the TLS protocol that are simply skipped by the parser, the hostname must be extracted and stored in a header structure to enable hostname-based filtering. A straightforward approach would be to define separate header structures for each possible hostname length; however, this would overwhelm the parser, hindering its ability to perform efficient DPI. Instead, a small set of header structures is defined, and the parser dynamically combines these structures at runtime to extract and store variable-length hostnames. The current implementation supports a maximum hostname length of 31 bytes. To handle all possible lengths, five header structures are defined, each capable of holding 1, 2, 4, 8, and 16 bytes, respectively. Any hostname shorter than 32 bytes can be extracted and stored across these structures. For instance, if the hostname is *www.google.com*, which has a length of 14 bytes, the parser would divide it into three header structures of

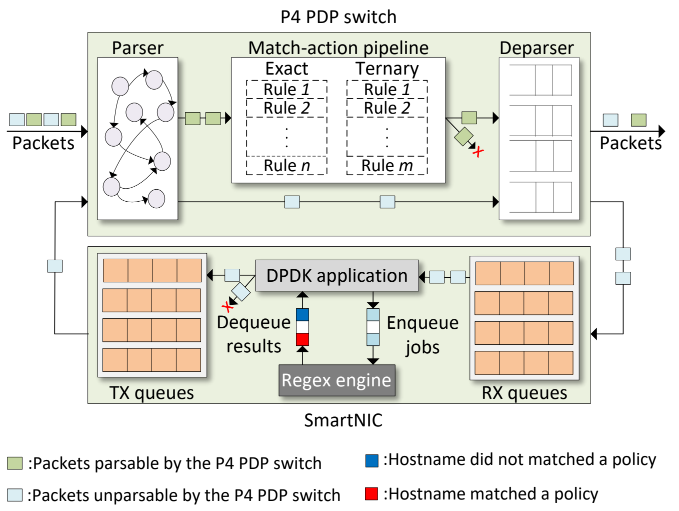

Fig. 3: Extending the processing capabilities of the P4 PDP switch using smartNIC. The smartNIC hosts the DPDK application responsible for parsing packets unparsable by the switch and enforcing security policies using the Regex engine.

sizes 2, 4, and 8 bytes, as 14 can be represented as $2^1+2^2+2^3$.

C. Applying Fine-grain and Coarse-grain Policies in P4

After extracting the hostname, the programmable match-action pipeline enforces both fine-grain and coarse-grain policies. Fine-grain policies are applied using exact matching, where the hostname is first hashed with CRC_{32} and then matched against a table. This table supports exact matching on up to 500,000 entries, allowing administrators to define a blacklist or whitelist of 500,000 hostnames. Coarse-grain policies, on the other hand, are enforced using ternary matching. Since ternary matching consumes more resources and the hostname is significantly longer than the 32-bit hash, this table is limited to 15,000 entries. It's also important to note that multiple entries may be needed to enforce a single rule in ternary matching.

D. Handling Unparsed Packets in DPDK

All packets flagged as *unparsable* by the P4 parser are forwarded to the DPDK application, as shown in Fig. 3. This application runs on the smartNIC, where it parses and extracts the hostname from all packets received from the P4 PDP switch. Unlike the P4 application, the DPDK application can handle any TLS packet without limitations on size or structure, including variable header lengths. Similar to the P4-based processing, the DPDK application enforces fine-grain and coarse-grain policies. Fine-grain policies use exact matching. However, since pattern matching in software significantly increases packet processing latency, coarse-grain policies are applied using the smartNIC's Regex engine. This engine performs pattern matching on the hostname against 15,000 rules within a few microseconds. It's important to note that this microsecond-scale latency is the end-to-end delay, including the time required for the DPDK application to enqueue the query to the Regex engine, the engine's time to extract the query, perform pattern matching, push the results to the appropriate queues, and the time for DPDK to dequeue the results.

To clarify more, the DPDK application submits a burst of jobs to the Regex engine and continues processing other packets. It then received a burst of results corresponding to the submitted jobs. In such approach, the Regex engine imposes no overhead on the processing time of the packets. However, this approach cannot be applied in the proposed system as the DPDK application takes action based on the results received from the Regex engine. For this, after submitting the hostname to the engine, the application halts till the result is received. Halting the application is reflected as a few microseconds increase in the packet processing time.

V. Evaluation

A. Experimental Topology and Datasets

The topology used to run the evaluation experiments consists of three servers connected through a P4 PDP switch and one smartNIC. Two servers are used to replay TLS traces from public datasets and browser traffic. The third server is used to host the smartNIC, where the DPDK application is running. The switch is the Edgecore Wedge100BF-32X [19] equipped with Intel's Tofino ASIC chip operating at 3.2 Tbps. The smartNIC is NVIDIA Bluefield-2.

The traces used for evaluation are mainly from Virus Total [20], CTU normal and botnet traffic [21], and Flowprint [22].

- *Virus Total* is a collection of various antivirus products and online scanning engines. Through the academic services Virus Total provides, hundreds of gigabytes of malware samples are obtained. After that, the metadata of the samples is used to select only those that use TLS connections. Finally, the filtered samples are instrumented in an isolated environment to capture their network traffic.
- *CTU normal and botnet traffic* is a dataset collected at CTU university. The dataset contains Zeek log files and real botnet traffic mixed with the normal background traffic. Zeek records the hostname of the TLS connections, along with various statistics.
- *Flowprint* is an approach that fingerprints mobile applications from encrypted network traffic. Its dataset is collected from 215 Andriod and 196 iOS applications in the US, India, and China.

In addition to the publicly available datasets, browser traces are collected by capturing network traffic that targets the top 1,000 Alexa websites from Chrome, Firefox, Samsung Internet, and UC browsers.

B. P4 Parsing Capabilities

This experiment aims to identify the portion of hostnames that the P4 application can process from the collected datasets. Fig. 4 shows the Cumulative Distribution Function (CDF) of TLS SNI hostnames, with the vertical black line representing the maximum hostname length that can be processed. Among the top 1 million hostnames from Cloudflare and normal CTU traffic, 0.31% and 5.9% of hostnames, respectively, exceed the 31-character limit and are therefore not parsable in P4. For

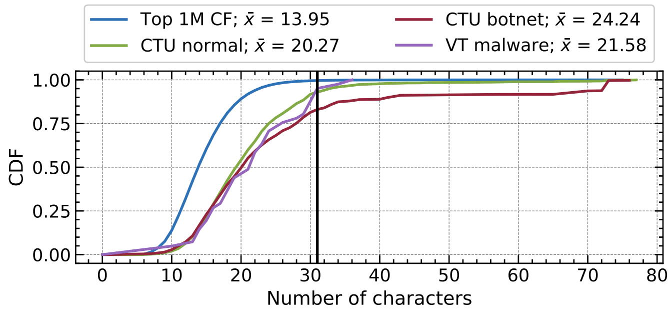

Fig. 4: CDF of DNS hostnames that are parsable by the P4 application. The vertical line indicates the maximum number of characters P4 application can process.

malicious hostnames from the CTU botnet and Virus Total, 15.9% and 4.8% of hostnames, respectively, are excluded for the same reason. Overall, the 31-character threshold proves to be effective, covering the vast majority of hostnames across both benign and malicious datasets. It is important to mention that the P4 application fails to process between 1% and 15% of the TLS Client Hello packets only and not from all TLS packets. All TLS packets other than Client Hello are fully parsable by the switch. For instance, the dataset obtained from CTU is around 8 GB of TLS traffic, out of which only 7,540 packets are TLS Client Hello.

C. Performance

1) Comparing the proposed system with Suricata: To evaluate the performance gain of the proposed system, it is compared to the Suricata Intrusion Detection/Prevention System (IDS/IPS) [23]. Suricata is running over a 32-core general-purpose CPU. A fair comparison between the proposed system and Suricata is achieved by (1) installing Suricata rules that only inspect the TLS SNI field, (2) and measuring the packet processing time incurred by the DPDK application as the time interval between a TLS packet leaves the P4 switch toward the DPDK application and the time the packet is received back by the P4 switch.

Fig. 5 depicts the CDF of the TLS packet processing time in the P4 switch, DPDK application, and Suricata. The P4 switch requires, on average, 700 nanoseconds (ns) and 900 ns to process TLS1.2 and TLS1.3 packets, respectively. The DPDK application requires, on average, 7.2 microseconds (μs) and 7.5 μs to process TLS1.2 and TLS1.3 packets, respectively. Suricata requires, on average, 92 μs and 95 μs to process TLS1.2 and TLS1.3 packets, respectively. For the vast majority of the packets, the P4 switch is performing the TLS SNI inspection, achieving two orders of magnitude processing time gain over Suricata. The DPDK application handles a minor portion of the packets, which still achieves an order of magnitude processing time gain over Suricata.

2) Evaluating the Regex engine in the proposed system: All the evaluations on the DPDK application show its performance by running on a single core of the smartNIC. The experiment did not differentiate between TLS1.3 and TLS1.2 packets. For stress testing, TLS packets are forwarded to the DPDK

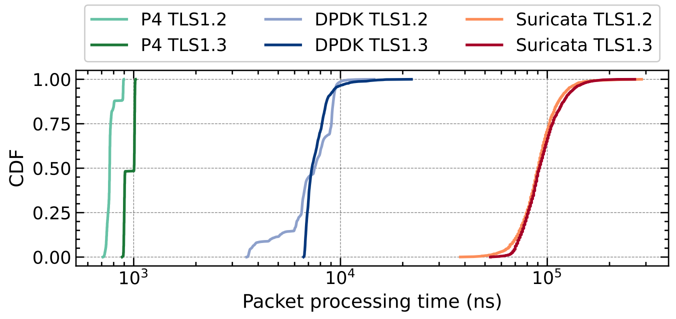

Fig. 5: Comparing TLS1.3 and TLS1.2 packet processing time in P4, DPDK, and Suricata.

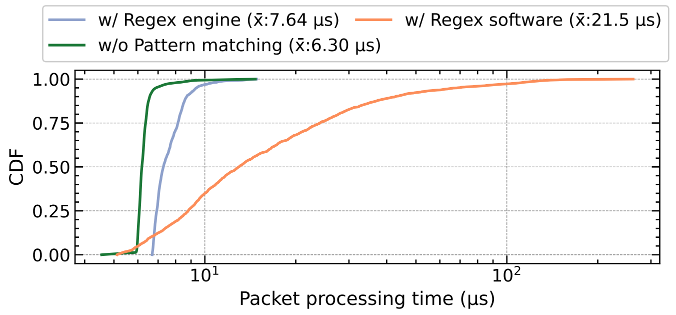

Fig. 6: Evaluating the packet processing time in DPDK without pattern matching and with pattern matching using the Regex engine and Regex software.

application at 1.5 Gbps. Fig. 6 shows the CDF of the packet processing time in DPDK w/o pattern matching and w/ pattern matching using the Regex engine and Regex software. Without pattern matching, DPDK requires, on average, 6.3 μs to process TLS packets. Enabling the pattern matching feature using the Regex engine of the smartNIC increases the average processing time by 1.34 μs to be 7.64 μs. Implementing pattern matching using the Regex software (i.e., using the Regex library) increases packet processing time by 15.2 μs to be 21.5 μs. Besides significantly reducing the average packet processing time when pattern matching is enabled, the Regex engine only adds 2 μs to the packet processing time in the worst-case scenario. On the other hand, using the Regex software adds more than 200 μs in the worst-case scenario.

VI. Challenges and Adopted Solutions

The first challenge in parsing the TLS protocol is implementing a loop in the parser to handle variable-length TLS extensions. A loop is crucial for two main reasons. First, in TLS1.3, the extensions are in random order, so the number of extensions that need to be parsed before reaching the SNI extension is unpredictable. Second, defining dedicated header structures and states for each extension would deplete the resources of the PDP switch [24]. By implementing a loop, the parser can process different extensions using a fixed amount of resources. Tofino P4 PDP switches allow for defining a stack of headers with identical structures and implementing a loop to populate this stack. However, using this method to parse variable-length headers is impractical because it consumes significant resources.

Defining a loop in the parser requires reusing the same set of states to parse multiple headers. The set of states used to form the parsing loop follow a unique processing behavior compared to the approach followed by the proposed system to parse TLS variable-length headers. The proposed system processes variable-length headers by first extracting the header length, storing it in a predefined header structure, and then skipping a number of bytes equivalent to the extracted length. While this method allows the parser to handle any header based on the extracted length, the parser cannot transition to the same state twice (i.e., form a loop). This is because header extraction is involved, and retransitioning to the same state would result in multiple extractions being stored in the same pre-defined header structure, which is not permitted by Tofino P4 PDP switches. To overcome this issue, the look-ahead feature is used. Look-ahead allows the parser to inspect a limited number of bits without extracting them.

A new challenge emerged when combining the look-ahead and advance functions of the PDP switches. The problem is that after matching on the bits obtained from the look-ahead function, the parser can skip a limited number of bytes. To clarify more, obtaining the extension length using the look-ahead function rather than extracting and storing the length in a header structure limits the number of bytes the parser can skip to 31. Because of this limitation, the parser flags packets with extensions larger than 31 bytes as unparsable. In order to skip more than 31 bytes, the programmer should design a chain of states such that each state skips at most 31 bytes and has a single transition. Otherwise, the P4 application will either not compile or the parser resources will rapidly deplete.

VII. Conclusion and Future Work

This paper presents a novel system for extracting the TLS SNI field in the data plane. The system has two components, a P4 PDP switch and a smartNIC. The P4 application running on the PDP switch is capable of parsing the majority of the TLS packets. Any TLS packets that require more resources than what the P4 PDP switch provides are forwarded to the smartNIC. The DPDK application running on the smartNIC parses all TLS packets coming from the P4 PDP switch. Both applications support fine-grain monitoring on 500,000 host-names. The P4 application utilizes 15,000 ternary matching rules to support coarse-grain monitoring, and the DPDK application utilizes the Regex engine to support pattern matching on 15,000 rules. Performance evaluation shows that the P4 PDP switch and the smartNIC require approximately 1 μs and 7 μs of processing time, respectively. Future work aims at optimizing the parser implementation to maximize the number of packets that can be processed by the P4 PDP switch.

Acknowledgement

This work was supported by the U.S. National Science Foundation, OAC-Core award number 2403360 and OAC-CC* award number 2346726.

References

[1] Google Transparency Report, "HTTPS encryption on the Web." [Online]. Available: https://tinyurl.com/2r6dre39.
[2] E. Rescorla, "The transport layer security (TLS) protocol version 1.3." [Online]. Available: https://tinyurl.com/3fek25t8.
[3] D. Eastlake, "Transport Layer Security (TLS) Extensions: Extension Definitions." [Online]. Available: https://tinyurl.com/5n8fu29d.
[4] A. AlSabeh, K. Friday, J. Crichigno, and E. Bou-Harb, "Effective DGA Family Classification using a Hybrid Shallow and Deep Packet Inspection Technique on P4 Programmable Switches," in *ICC 2023-IEEE International Conference on Communications*, pp. 3781–3786, IEEE, 2023.
[5] J. Kim, H. Kim, and J. Rexford, "Analyzing Traffic by Domain Name in the Data Plane," 2021.
[6] A. Kaplan and S. L. Feibish, "DNS water torture detection in the data plane," in *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*, pp. 24–26, 2021.
[7] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4DDPI: Securing P4-Programmable Data Plane Networks via DNS Deep Packet Inspection," in *Proceedings of the 2022 Network and Distributed System Security (NDSS) Symposium*, pp. 1–7, 2022.
[8] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends,"
[9] A. Mazloum, E. Kfoury, J. Gomez, and J. Crichigno, "A survey on rerouting techniques with P4 programmable data plane switches," *Computer Networks*, vol. 230, p. 109795, 2023.
[10] E. F. Kfoury, S. Choueiri, A. Mazloum, A. AlSabeh, J. Gomez, and J. Crichigno, "A comprehensive survey on smartNICs: Architectures, development models, applications, and research directions," *IEEE Access*, 2024.
[11] A. Kaplan and S. L. Feibish, "Practical handling of DNS in the data plane," in *Proceedings of the Symposium on SDN Research*, pp. 59–66, 2022.
[12] A. AlSabeh, K. Friday, E. Kfoury, J. Crichigno, and E. Bou-Harb, "On DGA detection and classification using P4 programmable switches," *Computers & Security*, vol. 145, p. 104007, 2024.
[13] S. Gupta, D. Gosain, M. Kwon, and H. B. Acharya, "DeeP4R: Deep Packet Inspection in P4 using Packet Recirculation," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2023.
[14] T. Jepsen, D. Alvarez, N. Foster, C. Kim, J. Lee, M. Moshref, and R. Soulé, "Fast String Searching on PISA," in *Proceedings of the 2019 ACM Symposium on SDN Research*, pp. 21–28, 2019.
[15] R. Asaoka, A. Nakao, M. Oguchi, and S. Yamaguchi, "Accuracy Improvement by Occurrence Probability of Service Identification based on SNI," in *2022 Tenth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 401–405, IEEE, 2022.
[16] PaloAlto, "URL filtering lookup and enforcement differences between clear text HTTP and encrypted HTTPS traffic." [Online]. Available: https://tinyurl.com/42jnxemm.
[17] E. Kfoury, J. Crichigno, and E. Bou-Harb, "Offloading media traffic to programmable data plane switches," in *IEEE International Conference on Communications (ICC)*, 2020.
[18] K. Friday, E. Kfoury, E. Bou-Harb, E. Bou-Harb, and J. Crichigno, "Inc: In-network classification of botnet propagation at line rate," in *27th European Symposium on Research in Computer Security*, 2022.
[19] E. Networks, "Wedge 100BF-32X." [Online]. Available: https://tinyurl.com/2xay8kky, Accessed on 09-06-2024.
[20] "VirusTotal." [Online]. Available: https://tinyurl.com/2p9cscna.
[21] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *computers & security*, vol. 45, pp. 100–123, 2014.
[22] T. Van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. Van Steen, and A. Peter, "FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic," in *Network and distributed system security symposium (NDSS)*, vol. 27, 2020.
[23] "Suricata." [Online]. Available: https://suricata.io/.
[24] A. Mazloum, A. AlSabeh, E. Kfoury, and J. Crichigno, "Security applications in P4: Implementation and lessons learned," *Computer Networks*, vol. 257, p. 111011, 2025.