sensors

MDPI

Article

Application Layer Packet Processing Using PISA Switches

Ismail Butun [1,2,*], Yusuf Kursat Tuncel [2] and Kasim Oztoprak [2]

[1] Department of Computer Engineering, KTH Royal University of Technology, SE-114 28 Stockholm, Sweden
[2] Department of Computer Engineering, Konya Food and Agriculture University, Konya 42080, Turkey; yusuf.tuncel@gidatarim.edu.tr (Y.K.T.); kasim.oztoprak@gidatarim.edu.tr (K.O.)
* Correspondence: butun@kth.se or ismail.butun@gidatarim.edu.tr

Abstract: This paper investigates and proposes a solution for Protocol Independent Switch Architecture (PISA) to process application layer data, enabling the inspection of application content. PISA is a novel approach in networking where the switch does not run any embedded binary code but rather an interpreted code written in a domain-specific language. The main motivation behind this approach is that telecommunication operators do not want to be locked in by a vendor for any type of networking equipment, develop their own networking code in a hardware environment that is not governed by a single equipment manufacturer. This approach also eases the modeling of equipment in a simulation environment as all of the components of a hardware switch run the same compatible code in a software modeled switch. The novel techniques in this paper exploit the main functions of a programmable switch and combine the streaming data processor to create the desired effect from a telecommunication operator perspective to lower the costs and govern the network in a comprehensive manner. The results indicate that the proposed solution using PISA switches enables application visibility in an outstanding performance. This ability helps the operators to remove a fundamental gap between flexibility and scalability by making the best use of limited compute resources in application identification and the response to them. The experimental study indicates that, without any optimization, the proposed solution increases the performance of application identification systems 5.5 to 47.0 times. This study promises that DPI, NGFW (Next-Generation Firewall), and such application layer systems which have quite high costs per unit traffic volume and could not scale to a Tbps level, can be combined with PISA to overcome the cost and scalability issues.

Keywords: software-defined networks; protocol independent switch architecture; programmable switches; P4; virtualization; stream processor; deep packet inspection

Citation: Butun, I.; Tuncel, Y.K.; Oztoprak, K. Application Layer Packet Processing Using PISA Switches. *Sensors* **2021**, *21*, 8010. https://doi.org/10.3390/s21238010

Received: 3 November 2021
Accepted: 25 November 2021
Published: 30 November 2021

1. Introduction

The telecommunication world is undergoing a great transformation. The most important aspect of this transformation is to switch from old hardware-dependent, vertical architectures to software-defined architecture. Although the use of NFV was a key improvement in a data plane with improved flexibility, Protocol Independent Switch Architecture (PISA) is one of the key elements with the accelerated performance and intelligent processing ability in the data plane during this change. The change in the architecture affects all stakeholders in a telecommunication operator infrastructure including applications. Legacy Applications written for legacy hardware are transformed into software-defined architecture.

Independent from the Software Defined Architectures, application identification became critical in the last decade. It positioned itself in the center of cyber-security, accounting, quality of service management, and similar services. One of the most important problems incurred by application identification is resource hungry behavior in itself. Next-Generation Firewalls (NGFW), and Deep Packet Inspection (DPI) systems are two of the most popular usage areas of application identification. DPI, as the name implies, inspects every packet that is running through the network deeply, and tries to classify it under a human-readable

name. It not only relies on packets metadata and headers but also packet payload, hence the name "Deep".

While L4 (OSI Layer-4) provides valuable information about a packet, it cannot give us any clue about the payload. In order to do that, packets must be inspected by maintaining the stateful information, and the payload must be constructed accordingly, so that it can be classified correctly [1]. With the help of L4 information, network-side security, such as stateful firewalls, can be built. Similar to NGFW processing packets in L7, DPI still needs to be inspected at L7. With the emergence of SDN architecture, DPI vendors switched from hardware to software based L7 DPIs. As they switch from hardware-dependent architecture to SDN based architecture, they lack the proper scalability to match the actual line speed of the switches. While the capacities of the data backbone increase, the systems depending on application identification became the bottleneck of the infrastructure.

In this transformation, the network applications become a Virtual Network Functions (VNF). Current state-of-the-art high performance software-based DPI systems (DPI VNFs) can scale up to 160 Gbps in a Virtual Machine running on top of powerful hardware [2]. As the demand increases, the telecom operators will need application identification systems like NGFWs and DPI systems running with the speeds in the order of Tbps of traffic classification in real time and such as in a single instance of DPI. The performance gain arises from the fact that the classification operation starts at the switch-level code data plane and continues in the user-plane.

In the meantime, several parallel works concentrated on various aspects (machine and deep learning methods, cybersecurity, etc.) of SDN. For instance, Ref. [3] argued about the necessity of providing quality of service (QoS) for each application on the network by the network operators, which can be accomplished by classifying network traffic associated with the applications. Authors have shown that deep learning models can be employed for classifying the network traffic, and residual network (ResNet) model outperforms the CNN convolutional neural network (CNN) model. In another work, cybersecurity related issues on the network layer are investigated [4]—for instance, detecting application-layer DoS attacks that utilize encrypted protocols by applying an anomaly-detection-based approach to statistics extracted from network packets.

In this paper, we aimed to introduce application layer processing capabilities of P4-based programmable switches and their usage in application layer processing. We investigated and proposed a solution for Protocol Independent Switch Architecture to process application layer data, enabling the inspection of an application content and triggering appropriate response. Protocol Independent Switch Architecture is a novel approach in networking where the switch does not run any embedded binary code but rather an interpreted code written in a purpose-specific language. The main motivation behind this approach is that telecommunication operators do not want to be locked in by a vendor for any type of networking equipment, developing their own networking code in a hardware environment that is not governed by a single equipment manufacturer. This approach also eases the modeling of equipment in a simulation environment as all of the components of a hardware switch run the same compatible code in a software model. The novel techniques in this paper exploit the main functions of a programmable switch to create the desired effect from a telecommunication operator perspective to lower the costs and govern the network in a comprehensive manner.

As stated before, the current demand in traffic growth puts a burden on the applications running in the application layer in the telecommunication world, although the performance and capacities of DPI systems and Next-Generation Firewalls do not grow with the demand of traffic growth; in addition, they cannot adapt themselves to the current revolution which migrates the networks into SDN-based systems. This paper proposes a solution using PISA switches with a DPI enabling application visibility (type identification) in an outstanding performance. The proposed architecture processes the packets in a network switch while selecting only necessary ones to the L7 based systems such as DPI and NGFW. The proposed solution distributes a load of DPI/NGFW systems into PISA switches and

DPI/NGFW systems. Using the proposed solution in a network allows the users to grow in the Tbps scale as well as benefit from Network/Service Function chaining, which will also remove the overhead of passing through all inspection systems for unnecessary traffic. The simulation studies demonstrate that this approach increments the performance of NGFW and DPI systems in the order of 40 times. Building such a flexible and scalable application visibility system is challenging. This study also tries to give an answer to how network operators should design their networks in order to benefit from such solution processing packets in L7 knowledge with the performance of L4; in other words, they should figure out how to scale out such system for a high volume of data in real time.

2. Background

2.1. Protocol Independent Switch Architecture (PISA)

The research on programmable switches led to the definition of a re-configurable match-action table (RMT) [5] based hardware that can be programmed with a domain-specific language. Protocol Independent Switch Architecture (PISA) is a special case of RMT that supports the P4 language as the default domain-specific language [6].

A typical PISA switch consists of a programmable parser, ingres match-action table, a queue, a set of registers to keep the state of variable, egres match-action table, and a programmable deparse as shown in Figure 1.

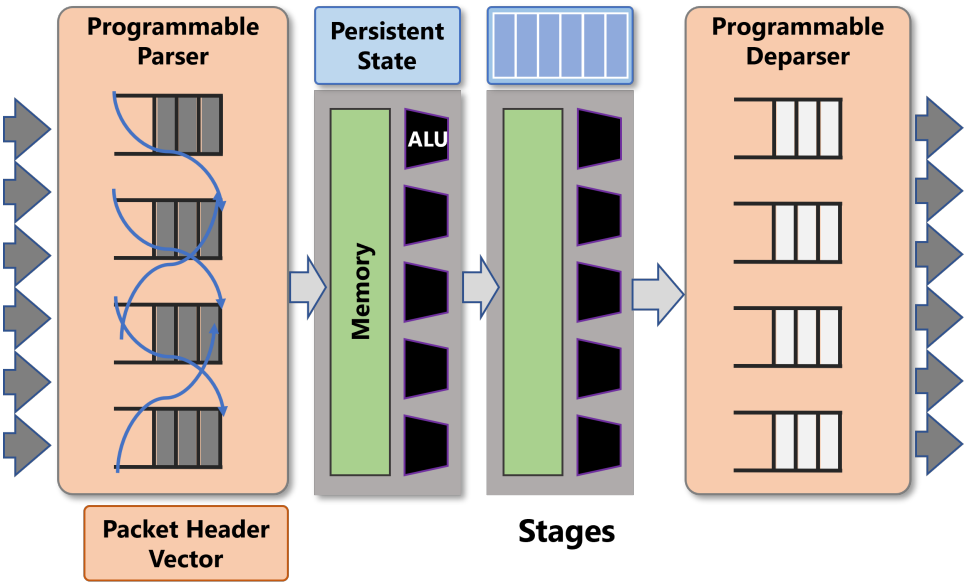

Figure 1. PISA match-action table processing pipeline (Source: Adapted from [7]).

The parser and deparser are programmed for processing user-defined packet header formats. The ingress and egress pipelines are the actual packet processing units that go through match-action tables in stages. Match-action tables match the header based on a set of rules that is controlled by control plane and performs the corresponding action on the packet. Actions use primitives to modify the non-persistent resources (headers or metadata) of each packet.

2.2. P4 Language

Although there are several studies developing and using programmable hardware [8–11], the early use of programmable hardware is to make telemetry data easy to use. Telemetry data are crucial for an automated future but generating telemetry data is not a trivial task. Adding more hardware and software to the routing and switching systems makes the current architecture more complex than ever. Since the telemetry data are generated at the packet level, the most logical way of doing this seems to be arising from the packet generating software at the hardware level, which leads us to P4, Programming protocol-independent Packet

Processors, as it is referred to in the original paper defining it [12]. P4 is a domain-specific programming language for packet-processing hardware such as a router, switch, network interface cards, and network function related appliances that work and data plane based on the decisions from the control plane as in Figure 2.

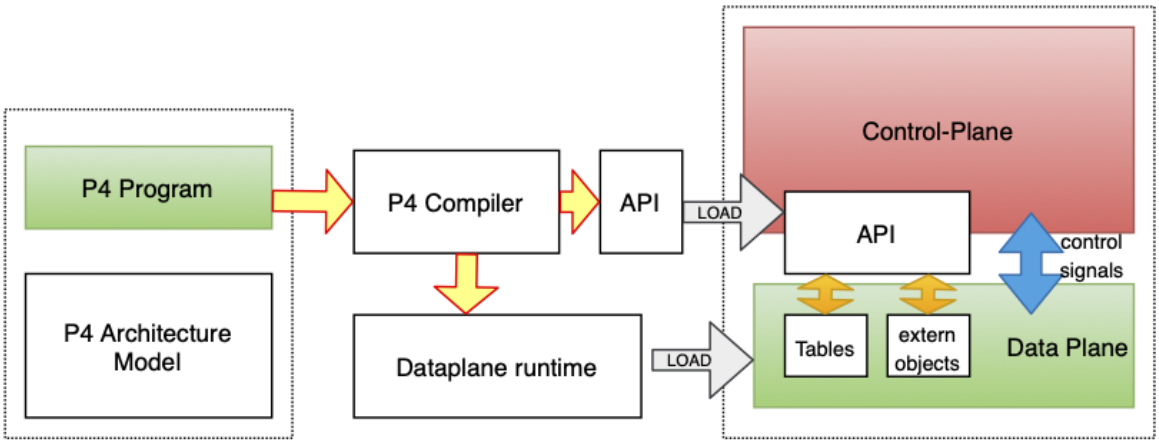

Figure 2. P4 Architecture (Source: Adapted from [9]).

In a typical PISA switch, execution of a P4 program is explained in Figure 3.

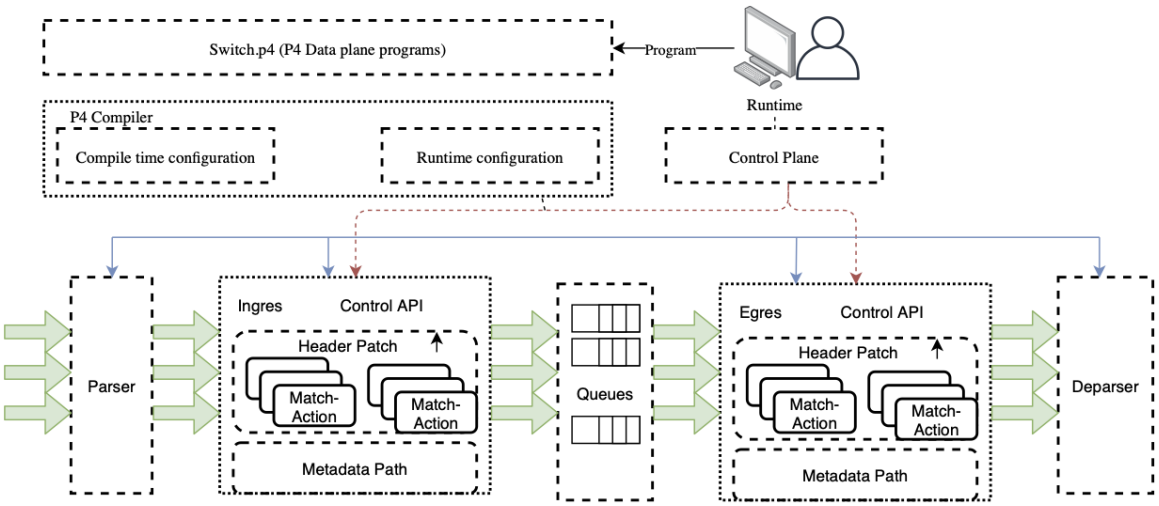

Figure 3. Pipeline execution in a P4-enabled switch (Source: Adapted from [13]).

1. The user develops a P4 program, which can be any type of network function, such as router, firewall, load balancer, or packet inspection switch.
2. P4 compiler compiles the program as a JSON file and sends it to the switch, which can be a physical switch or a software model of it.
3. The states of parser, match-parser, match-action tables, ingress, egress queue, and deparser is controlled by P4 execution.
4. The states of match-action tables are additionally controlled by control-plane which can change the behavior of the P4 code at run-time.

When the P4 compiler is placed between the program and the API, the P4 compiler translates the domain-specific language P4 code into a JSON file, which acts as an executable file for the PISA switch. The required CLI commands to configure to be switched are also sent with this JSON file, which typically contains the newly added match-action table names, ingress, and egress queue names to be created on the PISA switch. This JSON file is actually a series of match-action table entries that acts as an executable for the switch to change the state of its tables based on the incoming packet.

The control plane commands contains the necessary table initialization based on the packet processing actions. The implementation of P4 control plane commands may differ from each other depending on the switch type (physical or virtual), vendor and the version of P4 (P4-14, P4-16). The following commands are valid for Simple Switch Behavioral Model V2, P4-16: [14]

- `table_set_default <table_name> <action_name> <action_parameters>`
  is used to set the default action (i.e., the action executed when no match is found) of a table.
- `table_add <table_name> <action_name> <match_fields> => <action_parameters>`
  is used to set the action related to a specific match in a table.
- `mirror_add <source> <destination>`
  is used to mirror a specific port internally.

P4 programs ease the development of a network equipment code to a level for which only 128 lines are enough to build a simple IP switch with header validation [15]. Although the language itself is simple, there are other tools that emit P4 language code from another high-level language, such as the work done by the authors [13], P4HDL, which generates P4 code from a pseudo-code.

2.3. In-Band Telemetry with Programmable Switches

The above three requirements to develop a programmable hardware are not the only features addressed by P4. One of the most promising features of P4 arises in the telemetry. In-band Network Telemetry (INT) is defined in P4 language as one of the main applications [14]. Since P4 executes at the packet-processing level, it can rewrite every segment of the packet header, including the custom headers. This type of modification cannot be done in traditional statically programmed hardware-based network equipment. P4 helps set up a data plane by using the packet headers appropriately to collect even more information on the network's status than what we can determine using conventional methods [16]. The idea behind INT is to collect telemetry metadata for each packet, including routing paths for the packet, entry and exit timestamps, the packets' latency, queue occupancy in a given node, use of egress port connections, and the like. These measurements can be produced by each network node and sent in the form of a report to the monitoring system. Another way to embed them in packets is to delete them into allocated nodes at any node on the packet visits and connect them to the monitoring system. In a recent study, researchers used P4 INT experimental validation for telemetry based monitoring applications on the multi-layer optical network switches [17]. Using the telemetry data and the integrated software around it, semi-automatic congestion control over optical network switches can be achieved with the currently available SDN/NFV systems.

Although telemetry data can be collected in any way that is defined by P4 code, there are two types of telemetry that are defined in a standard P4 implementation [6]. As shown in Figure 4, telemetry data can be either embedded within a packet, which is called INT-MD, or extracted as a separate packet, called INT-XD. INT-MD is usually used by intermediate routers (switches to identify any type of problem that might occur along the path, which INT-XD is useful for external applications that do not need the payload of the original packet. In this experiment, INT metadata are used to help to the measurement of a switch internal state such as ingress/egress port ID, switch ID, queue occupancy, processing time, etc. These metrics are application agnostic and help in application-layer processing.

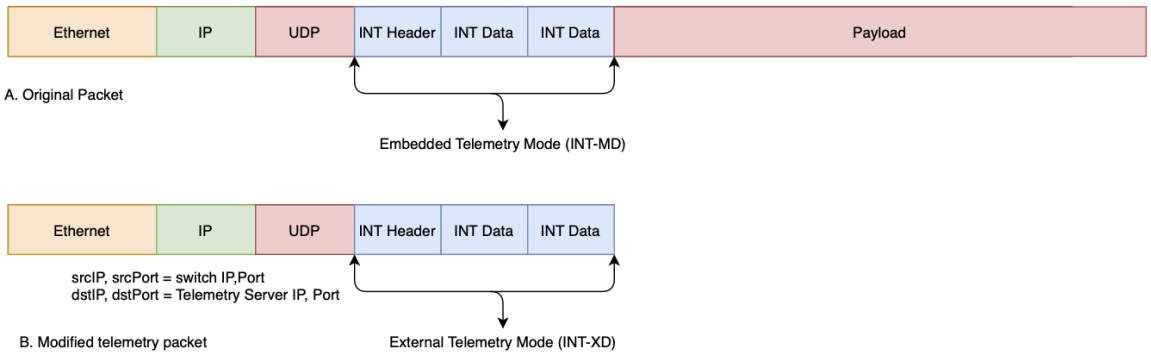

Figure 4. In-band network telemetry.

2.4. Real-Time Data Streaming

Real-time data streaming is shown to be beneficial for safety critical networks by removing possible bottleneck situations at the data accumulation joints, such as the data aggregator switches at the industrial networks [18]. In these networks, a possible delay in data would cause disastrous events, and data-streaming is a very good candidate solution as a remedy to this problem [19]. In the context of programmable switches, real-time data streaming is combined with telemetry to add application analytics, visibility, and troubleshooting features to a network stream. Apache Spark [20] and Apache Flink [21] are two of the most prominent pieces of software that is being used in streaming network telemetry data.

2.5. Deep Packet Inspection (DPI) and Application Layer Visibility

Deep Packet Inspection is important for telecommunication operators to gain more insight about the network and subscribers for revenue generation as well as cyber-security. A series of research [22,23] made in this area by the same author showed that subscriber profiling based on application level classification is critical for operators to increase the revenue and generate insight about the network. As the name implies, DPI inspects every packet with respect to the source, destination, header information, payload, and any other layer that is wrapped into it. Application layer visibility enable operators to distinguish between their subscribers and offer them new subscription services accordingly. As the video content is on the rise, operators can offer subscribers based on their use of online video services, such as Netflix, Amazon Prime, or Hulu. In addition, DPI is a supportive tool in employing Lawful Intercept or applying some appropriate filters to the Internet access of children.

3. Application Layer Processing with P4 Switches

The transformation from legacy systems into software defined architectures triggered the change in the hardware architectures. The demand for the change resulted in the development of PISA switches. The current state of the art in a PISA switch can scale up to 12.8 Tbps with a single ASIC/FPGA interface running with the speed of 400 Gbps. After the introduction of PISA switches into the production environment, the applications running in L4 such as Load Balancers, Volumetric DDoS attack detection, and prevention systems, port-based DNS applications are being ported into PISA switches. In this study, we aim to extend the use of PISA switches into L7 applications by designing a proper architecture. In the proposed architecture, by using PISA switches and its primary programming language P4, an application-level traffic analyzing system is proposed in a software-based emulation environment. It is basically combining L4 analytics of P4 architecture and L7 properties of the current state of the art in DPI or similar application layer inspection systems. The proposed architecture can be used to build a brand-new NGFW or DPI, by eliminating the complexities arising from switch dependent code.

3.1. Proposed System Architecture

The proposed system architecture in Figure 5 consists of five main components: PISA, Stream Processor, Control Plane, and Data Plane Configuration.

PISA: Programmable Switch that can run multiple instances of different P4 codes.

Data Plane: The generated P4 code for specific monitoring/telemetry/DPI/NGFW tasks. These P4 programs can be deployed according to specific task needs.

Control Plane: Programmable Switch related control plane engine to be placed. The control plane is aware of Data plane drivers and can communicate with the underlying switch according to the specific tasks. Although the proposed architecture supports any application specific task, from now on, the architecture will be coupled with DPI use case to make it easier to understand. This module is DPI-aware, which is fed from the specific packet stream, so that any decision to be made on the switch can be controlled by examining the specific packets.

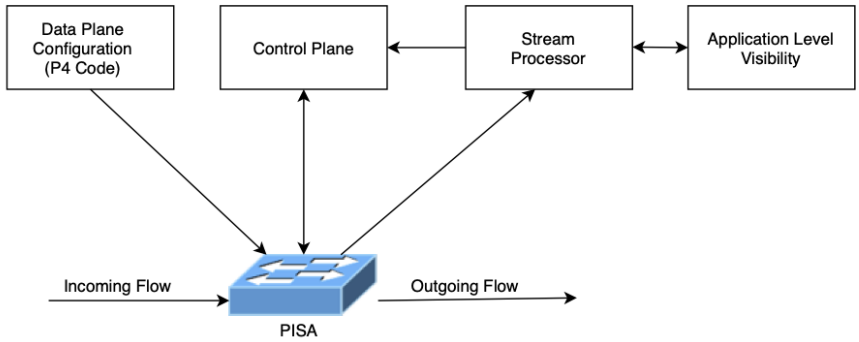

Figure 5. Proposed system architecture.

Stream Processor: The stream processor to operate on the matching stream patterns based on the decisions taken from data plane configuration. Specific telemetry tasks can be offloaded to the stream processor to decrease the workload over the switch or vice versa. Workload trade-off between the stream processor and the switch is based on the number of streams that matches a specific monitoring task.

Application-Level Visibility: Application-level visibility is the component that actually identifies the types of application based on their L4 to L7 properties, which is also called DPI. In a typical DPI system, a server with network interfaces is running the DPI application. There are two usage modes of DPI systems which are active and passive DPI systems. In the passive mode, they are fed by a mirror of the traffic and processes it offline. On the other hand, active DPI systems fall within the whole traffic and are supposed to process all the traffic piece by piece in real time.

In the proposed architecture, the PISA switch processes the packets in the network layer and can even process the flows in the transport layer and co-operates with the stream processor to identify the applications. This is the point where the aggregation–disaggregation of high performing PISA switch and application identification engines.

The PISA switch selects the minimal packets from the flows and forwards them to the stream processor/DPI engine to identify the applications and generate the actions among the predefined policies. The proposed architecture combines the power of PISA and L7 application inspection/classification/processing features by designing them together. The simulation results indicate that in the near future most of the systems using application awareness will re-design their systems running on top of PISA switches together with their redesigned applications as a stream processor. The following algorithm shown in Listing 1 explains our approach:

Listing 1: Pseudo Code proposed for the P4 switches.

```
While packet -> in ingres buffer
        Extract telemetry headers
        Put in Flow-Keys Telemetry Headers
        If Flow Not in Flow-Table
Create flow in Flow-Table
        Else IF Flow-Packet-Count.< 2
                Put Payload in Flow-Packets ...
                with Flow-Keys in Flow-Table
Continue
        Else
                Create telemetry header with ...
                    INT-XD options
                Send Flow-Table in Flow-Keys ...
                    to External Telemetry
```

The accurate accounting of the flows can also be done with P4 language. The accounting of a flow should include the following information: Considering the definition of the flow, for every flow, count number of packets, number of bytes, flow start time, flow end time, in addition to that, for TCP flows, TCP flags.

The P4 code on a switch would combine the accounting information and send the rest to the aggregator with a pseudo-code. Please see the Appendix A for the details of the mentioned pseudo-code.This pseudo-code works as the preprocessor of the flow, extracts the required fields and sends it to the stream processor for further processing.

Lastly, the traditional DPI systems has two operating modes:

- Inline
- Out-of-Band

In the inline mode, DPI systems are placed between the edge and core network, so that the traffic is processed as the flow continues. This operating mode enables DPI to apply policies directly on the flow, without requiring any other hardware. The biggest disadvantage of this approach is that the DPI becomes the weakest link of the network, it should be scaled at least as much as the aggregated sum of the traffic received from the edges.

In out-of-band mode, DPI acts like a simple traffic analyzing tool; it received the traffic passively from a mirror port of a network aggregation device, collecting all the traffic information and applying policies accordingly. In this mode, the biggest challenge is policy application, as the traffic is not directly passing through the DPI; it can only act on TCP traffic by sending TCP-resets to the source addresses, for example, to apply a restricted access policy to a particular destination address within the scope of the network. Other types of policy applications, such as bandwidth restriction, quality-of-service changes, etc., require control plane integration with the underlying network device.

Our architecture also combines the benefits of inline DPI devices with the out-of-band ones where the traffic is actively received on the switch, counted and reported on the aggregated external devices and the policies are actively applied as the event triggers occur.

3.2. Simulation Environment

To simulate the proposed architecture, the following components are built as a development and simulation environment:

P4 Simulation Environment: This is the default simulation target for BMV2 PISA switches, as shown in Figure 6, which includes Mininet by default and handles virtual NIC creating, switch port allocation, connecting the switch port to the host process, and running the rest of the packet flow.

Virtual Machine: This is the default virtual machine, built in a programmatic way with Vagrant, a developer friendly VM running environment, based on Ubuntu 14.04 (ubuntu/trusty64) and several other necessary components, as shown in Listing 2:

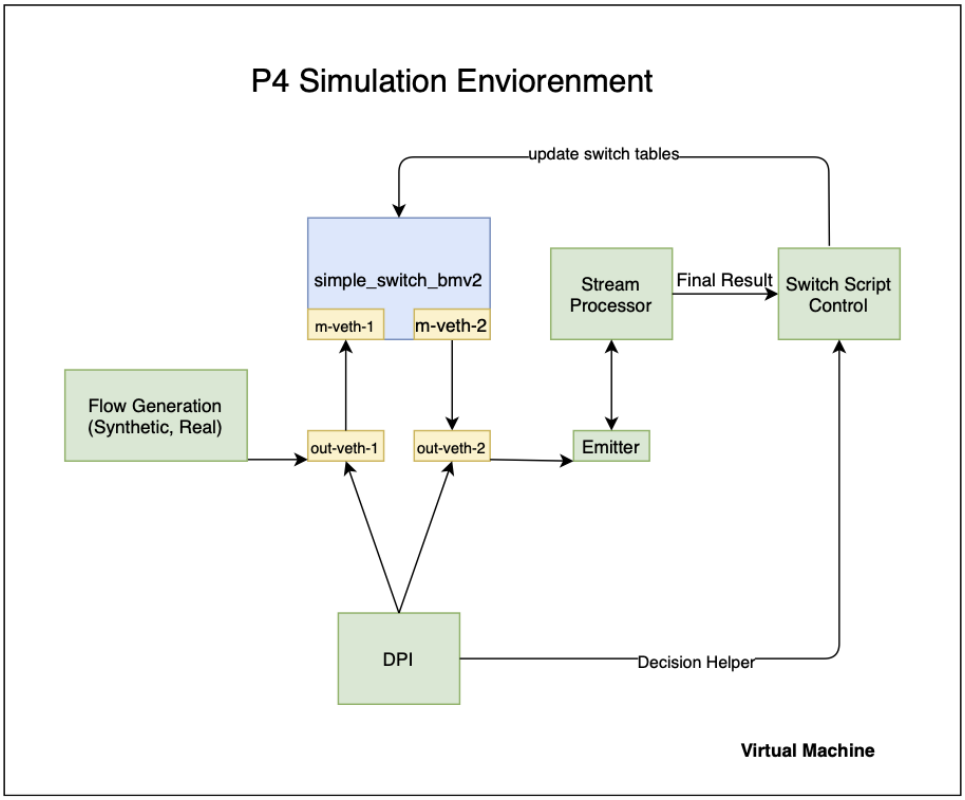

Figure 6. Simulation environment.

Listing 2: Virtual machine set-up.

```
Simple_switch_bmv2:
BMV2 software switch, based on Python2.7
        m-veth-1: Ingres mininet Switch Port
        m-veth-2: Egres mininet Switch Port
        out-veth-1: Ingrest Server Host Port
        out-veth-2: Egres Server Host Port
```

4. Experimental Study

In this experimental study, we have used a P4 simulation environment which was discussed above and presented in Figure 6. The following items describe each component of our experimental simulation environment in detail:

Flow Generation: This is the controlled flow generation tool, written in Go. Synthetic flows are created with Python, while real-flow is taken from the Canadian Institute for Cyber-Security [24].

DPI: Deep Packet Inspection module written in Go, based on nDPI [25].

Emitter: Flow emitter that reads from the mirroring port, extracts metadata header information written by Data-Plane and sends the rest of the packet for stream processor. This module is also Apache-Spark aware; the final result of the telemetry query is calculated by the Emitter module.

Stream Processor: The streaming processor for the rest of the flows that match the final criteria for the expected output. In this simulation, we used Apache-Spark as a stream processor. The stream processor will be upgraded to Apache-Flink for better scalability.

Switch Script Control: This script controls the switch tables to update the relevant switch tables under control.

The parameters for running the simulation are adjusted according to the following criteria:

- Session is TCP (Session has 3-way handshake);
- Session is UDP (Session has no 3-way handshake);
- Session is detected by nDPI;

- Session is not detected by nDPI.

4.1. Experiment-1: Application Identification Performance Improvement DPI Application Classification on Mixed Flow Captures

Our hypothesis is that, to identify an application in a packet, a few bytes in a flow (one or two packets depending on the application) should be enough to determine the type of application correctly. Keeping this in mind, we must first identify the session in a packet. This use case demonstrates the performance improvement in DPI systems by eliminating the number of packets by some factor.

In order to adjust the parameters of this identification, we first analyzed the packet stream with nDPI, counting the number of identified protocols and the number of packets that are included in each stream. We then start reducing the number of packets in each stream and run the protocol identification with nDPI once again, comparing the results of identification with the previous run. By reducing the number of packets each time, we calculated the number of identified protocols in each reduced packet stream.

Session Identification in an IP flow is based on two different IP sessions:

4.1.1. TCP Session

SrcIP, DstI, SrcPort, DstPort, TCPSeqNum

TCP Session Identification is based Source IP, Destination IP, Source Port Destination Port and the TCP Sequence Number. The TCP session is established after the 3-way handshake as shown in Listing 3:

Listing 3: Pseudo Code proposed for the 3-way handshake.

```
---
Source -> Destination (SYN+Seq #)
Destination -> Source (SYN ACK+Seq #)
Source -> Destination (ACK+Seq #)
---
```

After the last ACK of the source, Sequence Number is incremented for a flow in the TCP session. Actually, it comes from the nature of TCP. It starts randomly and increments by the amount of the data transferred in each packet. The same is valid for ACK number.

The packets that will be reduced should be the packets after this 3-way handshake packet. To identify the flows, we will use the packet SYN ACK, and the response to the third packet—in other words, the first two packets of the server (or destination to source).

4.1.2. UDP Session

SrcIP, DstI, SrcPort, DstPort

UDP is a connectionless protocol; there is no clear definition of a UDP session. Every packet may create a flow independently. Basic identification for UDP flow consists of Source IP, Destination IP, Source Port, and Destination Port. Since Source Port is randomly allocated depending on the OS (which is called ephemeral ports), any flow that is using the same source port is considered as the same UDP session.

4.2. Sample Packet Captures

To study the flow reduction, we used the sample captures from nDPI that is used for verification of protocol identification. The capture files consist of 183 files, containing more than one protocol in one capture file. Twenty-two files that are too small for reduction (having packets less than 2) are excluded from study. One packet especially crafted for testing invalid packet type is also excluded since we are interested in valid packets, leaving us 160 packet captures.

To reduce the flow, the following pseudo-code is used as shown in Listing 4:

Listing 4: Pseudo Code proposed for the reduced algorithm.

```
---
network_packets = rdpcap(infile)
sessions = network_packets.sessions()

for key in sessions:
        pktCount=0
        for pkt in sessions[key]:
                if (pktCount < 2):
                        write(pkt, outfile)
                        pktCount = pktCount + 1

---
```

In this code, sessions are extracted by the criteria of whether they are TCP or UDP sessions. As mentioned earlier, for the TCP session, 3-way handshake packets are excluded from the session, whereas, for a UDP session, there is no precondition to exclude the packets. We use the second packets of the 3-way handshake as the first packet of the flow.

After the extraction of sessions, an nDPI sample classifier is used to classify the application in each reduced capture by replaying the capture file on the switch.

The following Table 1 summarizes the results of the experiment:

Table 1. Rates for Test Captures.

μ REDUCTION RATIO	82%
μ REDUCTION FACTOR	5.5
μ DETECTION RATE	84%

The full data are available in Table A1.

4.3. Experiment-2: TCP-Based Application Identification Using Real-Life Data

In the second experiment, we used the real captures from [24], namely the files in the dataset named PCAP-01-12_0750-0818.

There are 69 files located in this dataset, each containing a real world data capture that contains data from a real DDoS attack along with different types of traffic.

To see the effect of TCP, we extracted the TCP streams and used the extracted streams to send to the simulation.

For the sake of convenience to the readers, the results in Table A3 are summarized in the following Table 2:

Table 2. Rates for real-life captures using only TCP streams.

μ REDUCTION RATIO	97.88%
μ REDUCTION FACTOR	47.16
μ DETECTION RATE	95%

4.4. Experiment-3: Application Identification in Full Stream Using Real-Life Data

In the final experiment, using the same capture files in Experiment-2, we treated the streams as is, sending them directly to the switch. The following results in Table 3 are achieved.

Table 3. Rates for real-life captures using full streams

μ REDUCTION RATIO	84.73%
μ REDUCTION FACTOR	6.5
μ DETECTION RATE	99.83%

Full results are given in Table A2.

4.5. Results and Discussion of the Experiments

The experimental study on the packet captures showed us that 2-packet reduction of a flow is accurate enough to identify a flow.

In Experiment 1, the decrease in detection rate is mostly caused by TLS encryption, which shows us that further study is needed to identify encrypted flow as shown in Table 1. An ML based approach would be implemented to success in application identification of all flows. Based on the results from Table A1, 125 out of 160 packet captures are correctly identified. Sixteen out of 160 packet captures could not be identified. Normally, 160 out of 160 packets would be identified correctly. In addition, 125 files were identified correctly; 16 not identified at all (0 identification); 19 partially identified; 16 non-identified protocols are completely encrypted protocols; 125 identified protocols, mixed partially TLS and plain protocols; and 19 partially identified protocols are mixed partially TLS and plain protocols. Detection Rate drops with the reduced flow. (i.e., as we reduce the flow, we also lose important flow information that is needed for packet identification, short flows) The reason for not identifying these packet captures are that they are mostly encrypted protocols, which require more than two packets to identify. We will expand the experiments according to this. Since the flows used in Experiment 1 are taken from nDPI's test captures, they consist of an artificially selected short flow containing all of the applications that nDPI can identify.

In Experiment 2, the results in Table 2 showed that it is possible to increase the detection rate while the reduction rate is also increased. This is due to the fact that there are only 17 protocols detected in TCP streams as indicated in Table A3, and most of them are not TLS-based protocols, or can be identified without deep inspection of the remaining payload.

In Experiment 3, the results in Table 3 indicated that, if we include UDP streams, the accuracy goes even higher, but the reduction rate decreases. This behavior is expected since the number of detected applications in Table A2 is 84, more than the number applications detected in TCP streams, but the number of packets in UDP streams is lower than the number of packets in TCP streams. This result is in line with results obtained from the study in [26].

5. Conclusions

The results of this study indicate that the application layer data processing can be done with PISA switches. We do not always need complex techniques to inspect the packets in L7, and a simple flow-based packet reduction can achieve significant accuracy to identify the flows and add application-level visibility over the network. Streaming processing combined with switch-level applications helps us build strong networking applications. In-band Network Telemetry is in the central position of a programmable switch that distinguishes and separates them from the traditional switches. The proposed method constructs a Network Processor with a specific task from each PISA-stream processor pair. The simulation results indicate that using such PISA switches in the center of all network traffic will increase the performance of such systems on the order of tens of folds. The use of such (proposed) systems will solve the capacity problems experienced with applying full network service chaining. In other words, by using a single PISA switch and tens of stream processors with different features (DPI, NGFW, etc.) on different ports, it constructs a big traffic exchange fabric with dynamically attached Network Processors of different types with very low costs.

The results of this study demonstrate that the proposed system reduces the traffic load of such systems by a factor of 5.5 to 47.0 times with acceptable application identification. Applying some ML based approaches would increase the success rate as if all traffic is going through legacy systems with the higher power of proposed systems. In addition, real traffic scenarios indicate that the performance gain would reach up to a factor of 40 on average by using the statistics in this study [26].

The studies in the literature and our experimental studies demonstrated that PISA switches are the glue for the SDN-NFV couple increasing the performance of such systems. One of the major problems of the NFV based application layer processing systems were the network packet processing performance bottleneck; however, the proposed solution offering an architecture avoids the performance bottleneck of both PNF (Pyhsical Network Function) and VNF (Virtual Network Function) systems by decreasing the network packet load.

6. Future Study

DPI, NGFW (Next-Generation Firewall), and such application layer systems that have quite a high cost per unit traffic volume and could not scale to a Tbps level can be combined with PISA to overcome the cost and scalability issues. Practical applications are expected to be available in the upcoming years, maybe even months.

Encrypted network traffic identification with P4 language is one of the main future areas of study for this thesis. In-band Telemetry seems to be a good place to start this study, as it tells us about the characteristics of a flow on a packet level. In this kind of an analysis, AI/ML methods can provide a lot of help in defining the features of traffic. As stated above, the use of PISA switches will allow the operators to collect in-band telemetry information, which will also create the building ground for Zero Touch Networking (ZSM) once the networks are utilized with the use of proposed systems. Once ZSM features are injected into the infrastructures, operational costs and outage times will decrease dramatically.

Another area of interest based on this study could be Digital Twins in Telecommunication Networks. As PISA switches allow you to model the hardware in a software environment, it would be straightforward to build a DT (Digital Twin) of a telecom network and feed forward the actual data and commands towards the active network. Particularly, the data center network can be modeled completely using the DTs of core and edge network devices. Telcos can gain an advantage from this by running different scenarios on their DT based on different types of network flows. These network flows can be adjusted to plan the data center network topology according to the SLA of the customers.

Author Contributions: Conceptualization, K.O. and Y.K.T.; methodology, K.O. and Y.K.T.; software, Y.K.T.; validation, Y.K.T., K.O. and I.B.; formal analysis, K.O. and Y.K.T.; investigation, K.O., Y.K.T. and I.B.; resources, K.O., Y.K.T. and I.B.; data curation, K.O. and Y.K.T.; writing—original draft preparation, K.O., Y.K.T. and I.B.; writing—review and editing, K.O., Y.K.T. and I.B.; visualization, Y.K.T.; supervision, K.O.; project administration, K.O.; funding acquisition, I.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: Authors would like to acknowledge helpful staff of the MDPI Sensors for their endless help during the publication phase of our paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Code proposed for P4 switches is provided as follows in Listing A1:

Listing A1: Code proposed for P4 switches.

```
// Flow key registers
reg_src_ip = Register();
reg_dst_ip = Register();
reg_proto  = Register();
reg_l4  = Register();

// Flow statistics registers
reg_pkt_count = Register();
reg_byte_count = Register();
reg_time_start = Register();
reg_time_end = Register();
reg_flags = Register();
initialize_registers(hdr: PacketHeader, index:
HashIndex, md: Metadata):
reg_src_ip[index] = hdr.src_ip;
reg_dst_ip[index] = hdr.dst_ip;
reg_proto[index] = hdr.proto;
reg_l4[index] = hdr.l4;
reg_pkt_count[index] = 1;
reg_byte_count[index] = length(hdr.ethernet) + hdr.ip_len
reg_time_start[index] = md.timestamp;
reg_time_end[index] = md.timestamp;
reg_flags[index] = hdr.tcp_flags;

with pkt = ingress.next_packet():
hdr = parse(pkt);
md = pkt.metadata;
index = hash({hdr.src_ip, hdr.dst_ip, hdr.proto, hdr.
l4});
collision =
hdr.src_ip != reg_src_ip[index]
|| hdr.dst_ip != reg_dst_ip[index]
|| hdr.proto
!= reg_proto[index]
|| hdr.l4
!= reg_l4[index]
if collision:
// Export info and keep track of new flow
flow_record = { reg_src_ip[index],
reg_dst_ip[index],
reg_proto[index],
reg_l4[index],
reg_pkt_count[index],
reg_byte_count[index],
reg_time_start[index],
reg_time_end[index],
reg_flags[index] }
emit({hdr.ethernet, flow_record});
initialize_registers(hdr, index, md);
else:
// Update statistics of current flow
reg_pkt_count[index] += 1;
reg_byte_count[index] += length(hdr.ethernet)
+ hdr.ip_len
reg_time_end[index] = md.timestamp;
reg_flags[index] ||= hdr.tcp_flags;
```

Table A1. Detected protocols from the capture files.

	BYTES		**PACKETS**		**DET.STAT.**		**REDUCE RATE (%)**
	ORG	RDC	ORG	RDC	POS.	NEG.	
anydesk	2,962,572	767	6963	8	1	0	99.97
exe_download	734,335	328	703	4	1	0	99.96
exe_download_as	542,265	328	534	4	1	0	99.94
tor	3,106,096	3524	3859	42	4	0	99.89
whatsappfiles	467,113	760	620	8	1	0	99.84
wireguard	791,758	1576	2399	4	1	0	99.80
ps_vue	2,242,710	5184	1740	15	3	0	99.77
tls_long_cert	121,969	380	182	4	1	0	99.69
ftp	1,158,196	3805	1192	12	3	0	99.67
quic-mvfst	408,962	1414	353	2	1	0	99.65
git	76,165	376	90	4	1	0	99.51
netflix	6,323,017	32,776	6999	217	5	0	99.48
coap_mqtt	954,917	5505	8516	51	3	0	99.42
dns-tunnel	80,668	528	438	8	1	0	99.35
bitcoin	596,362	4816	637	24	1	0	99.19
wa_video	998,593	8587	1567	38	6	0	99.14
ssh	41,738	401	258	4	1	0	99.04
quic_t51	589,126	5664	642	4	1	0	99.04
quic-28	252,865	2782	253	4	1	0	98.90
bittorrent_ip	519,514	6512	479	8	1	0	98.75
skype-conf	44,487	616	200	4	1	0	98.62
dns_exfiltr	80,745	1149	300	4	1	0	98.58
instagram	3,009,247	47,580	3443	122	7	0	98.42
tls_verylong_ce	23,381	380	48	4	1	0	98.37
check_mk_new	22,594	391	98	4	1	0	98.27
quic-mvfst-22	300,063	5232	490	4	1	0	98.26
bad-dns-traffic	108,542	1934	382	12	1	0	98.22
capwap	108,037	2113	422	21	2	0	98.04
anyconnect-vpn	1,088,929	23,234	3001	166	17	0	97.87
openvpn	64,263	1392	298	12	1	0	97.83
webex	902,823	19,937	1580	223	6	0	97.79
bittorrent_utp	43,553	979	86	4	1	0	97.75
facebook	31,951	752	60	8	1	0	97.65
nintendo	357,057	9156	1000	66	3	0	97.44
simple-dnscrypt	47,340	1344	111	16	1	0	97.16
443-opvn	12,677	380	46	4	1	0	97.00
Oscar	11,090	352	71	4	1	0	96.83
google_ssl	9780	328	28	4	1	0	96.65
nest_log_sink	137,036	4806	1000	60	3	0	96.49
modbus	9129	358	102	4	1	0	96.08
quic046	93,697	3723	100	4	1	0	96.03
fix	145,778	5858	1261	48	1	0	95.98
weibo	279,507	11,287	498	104	6	0	95.96
tls_esni_sni_b	16,811	696	38	8	1	0	95.86
pps	2,307,979	104,799	2557	243	4	0	95.46
http-crash-	3544	168	9	2	1	0	95.26
smb_deletefile	33,172	1660	101	4	1	0	95.00
WebattackXSS	4,946,124	248,266	9374	2641	1	0	94.98
teams	1,554,287	78,248	2817	267	15	0	94.97
dnp3	51,786	2752	543	32	1	0	94.69
wechat	707,438	43,775	1672	287	15	0	93.81
s7comm	6580	408	55	4	1	0	93.80
telegram	374,409	25,197	1566	119	15	0	93.27
youtube_quic	198,575	13,389	289	12	2	0	93.26
1kxun	664,361	45,690	1439	297	16	0	93.12

Table A1. *Cont.*

	BYTES		PACKETS		DET.STAT.		REDUCE RATE (%)
bittorrent	312,904	21,595	299	74	1	0	93.10
ja3_lots_of1	7614	528	27	4	1	0	93.07
ja3_lots_of2	5396	380	11	4	1	0	92.96
wa_voice	187,832	13,276	736	76	11	0	92.93
viber	157,311	12,098	424	81	9	0	92.31
youtubeupload	130,326	10,358	137	12	1	0	92.05
dropbox	110,884	9056	848	48	1	0	91.83
amqp	27,354	2284	160	12	1	0	91.65
iphone	232,616	21,922	500	138	12	0	90.58
skype	708,140	71,068	3284	639	13	0	89.96
WebattackSQLinj	32,264	3384	94	36	1	0	89.51
quic	360,998	37,893	518	34	4	0	89.50
hangout	3230	340	19	2	1	0	89.47
ssdp-m-search	1653	174	19	2	1	0	89.47
BGP_Cisco_hdlc	1305	144	14	2	1	0	88.97
dos_win98_smb_	10,055	1130	220	9	3	0	88.76
skype_unknown	537,720	60,508	2146	537	13	0	88.75
netbios	30,922	3546	260	24	2	0	88.53
sip	51,847	5966	112	11	3	0	88.49
whatsapp_l_call	223,130	26,502	1253	187	11	0	88.12
rx	29,643	3641	132	18	1	0	87.72
6in4tunnel	43,341	5326	127	26	5	0	87.71
android	143,354	18,809	500	167	14	0	86.88
ajp	7414	1020	38	10	2	0	86.24
quic_q46	21,721	3028	20	4	1	0	86.06
quic_q50	20,914	3048	20	4	1	0	85.43
ethereum	264,111	39,317	2000	260	2	0	85.11
malware	8625	1347	26	10	4	0	84.38
teamspeak3	2223	354	13	2	1	0	84.08
quic_q39	25,625	4131	60	4	1	0	83.88
iec60780-5-104	12,561	2034	147	24	1	0	83.81
whatsapp_login	32,369	5963	93	19	7	0	81.58
whatsapp_voice_	34,319	6492	261	52	3	0	81.08
quic-mvfst-exp	27,029	5272	30	4	1	0	80.50
netflowv9	14,128	2832	10	2	1	0	79.95
ftp_failed	2132	476	18	4	1	0	77.67
smpp_in_general	1552	347	17	4	1	0	77.64
EAQ	26,563	6732	197	82	2	0	74.66
upnp	10,248	2928	14	4	1	0	71.43
fuzz-2020-02	158,043	46,445	366	125	3	0	70.61
quic-29	9746	3011	15	4	1	0	69.11
quic-24	8360	3029	15	4	1	0	63.77
zabbix	955	376	10	4	1	0	60.63
4in4tunnel	970	388	5	2	1	0	60.00
quic-27	13,367	5664	20	4	1	0	57.63
quic-mvfst-27	13,367	5664	20	4	1	0	57.63
quic_q46_b	7500	3239	20	4	1	0	56.81
fuzzing	32,268	15,422	131	81	3	0	52.21
mongodb	3388	1648	27	16	2	0	51.36
mssql_tds	17,172	8728	38	20	1	0	49.17
malformed_dns	6004	3096	6	4	1	0	48.43
quic-23	7671	3956	20	4	1	0	48.43
fuzz-2006	99,986	53,930	691	399	9	0	46.06
dnscrypt-v2-doh	230,431	132,987	577	136	1	0	42.29
skype_udp	459	278	5	3	1	0	39.43
teredo	3150	1980	24	14	1	0	37.14
quic_t50	8708	5664	12	4	1	0	34.96

Table A1. *Cont.*

	BYTES		PACKETS		DET.STAT.		REDUCE RATE (%)
smbv1	1365	895	7	4	1	0	34.43
diameter	2124	1488	6	4	1	0	29.94
websocket	561	428	5	4	1	0	23.71
steam	11,516	10,218	104	97	1	0	11.27
kerberos	30,139	29,412	77	75	4	0	2.41
encrypted_sni	2382	2382	3	3	1	0	0.00
tls-esni-fuzzed	2382	2382	3	3	1	0	0.00
4in6tunnel	2284	2284	4	4	1	0	0.00
mysql-8	463	463	4	4	1	0	0.00
ubntac2	1928	1928	8	8	1	0	0.00
filtered	21,595	21,595	74	74	1	0	0.00
dnscrypt-v1	321,274	321,274	608	564	2	0	0.00
WebattackRCE	210,131	210,131	797	797	2	0	0.00

Table A2. Application identification in full stream.

APPNAME	REDUCED BYTES	ORIGINAL B.	REDUCED PACKET	ORIGINAL P.	REDUCTION %
AFP	75.888	142.848	136	256	46.88%
Amazon	222.810	3.539.200	1.892	10.959	93.70%
AmongUs	74.772	187.488	134	336	60.12%
Ayiya	70.308	167.400	126	300	58.00%
BitTorrent	264.492	566.928	474	1.016	53.35%
BJNP	110.484	223.200	198	400	50.50%
CAPWAP	110.484	225.432	198	404	50.99%
CiscoVPN	90.636	174.456	166	318	48.05%
Cloudflare	3.432	57.108	52	290	93.99%
COAP	205.344	429.660	368	770	52.21%
Collectd	94.860	180.792	170	324	47.53%
CPHA	149.544	305.784	268	548	51.09%
DHCP	188.802	575.730	355	1.259	67.21%
DHCPV6	6.178	238.728	42	1.624	97.41%
DNS	1.285.798	1.528.164	11.150	12.354	15.86%
Dropbox	118.296	232.128	212	416	49.04%
EAQ	162.936	363.816	292	652	55.21%
Facebook	78.980	83.836	804	848	5.79%
FTP_CONTROL	9.736	27.940	148	430	65.15%
Github	8.592	8.986	92	96	4.38%
GMail	20.928	704.538	192	4.458	97.03%
Google	2.274.505	44.542.510	23.970	142.071	94.89%
GoogleServices	115.472	2.215.516	964	9.062	94.79%
GTP	263.376	565.812	472	1.014	53.45%
H323	159.588	351.540	286	630	54.60%
HTTP	799.416	17.123.436	11.300	59.257	95.33%
HTTP_Proxy	3.132	3.252	52	54	3.69%
IAX	118.296	241.056	212	432	50.93%
ICMP	380.064	6.251.658	4.052	48.536	93.92%
ICMPV6	5.548	88.904	62	954	93.76%
Instagram	74.948	77.950	484	512	3.85%
IPsec	279.632	590.996	500	1.058	52.68%
IRC	95.976	213.156	172	382	54.97%
iSCSI	212.040	449.748	380	806	52.85%
Kerberos	48.228	124.116	90	226	61.14%
LDAP	94.860	249.984	170	448	62.05%
LinkedIn	15.346	17.774	144	168	13.66%
LISP	156.240	330.336	280	592	52.70%

Table A2. *Cont.*

APPNAME	REDUCED BYTES	ORIGINAL B.	REDUCED PACKET	ORIGINAL P.	REDUCTION %
LLMNR	149.644	304.968	282	588	50.93%
MDNS	213.722	678.891	416	2.023	68.52%
Megaco	46.872	103.788	84	186	54.84%
Memcached	8.052	15.864	18	32	49.24%
Microsoft	76.694	784.104	640	2.493	90.22%
Microsoft365	5.064	144.776	44	314	96.50%
MsSQL-TDS	2.640	3.600	44	60	26.67%
NetBIOS	134.656	300.524	248	582	55.19%
NFS	111.600	243.288	200	436	54.13%
NTP	54.684	112.716	98	202	51.49%
OpenVPN	105.024	224.436	190	404	53.21%
OSPF	21.368	880.742	228	9.307	97.57%
Playstation	75.012	167.760	138	306	55.29%
Radius	213.156	444.168	382	796	52.01%
RDP	110.964	221.568	206	406	49.92%
Reddit	9.332	10.292	88	96	9.33%
RemoteScan	190.836	379.440	342	680	49.71%
RTSP	45.756	100.440	82	180	54.44%
RX	8.928.188	25.862.372	16.002	46.350	65.48%
sFlow	131.688	280.116	236	502	52.99%
SIP	245.320	512.044	446	924	52.09%
SMBv1	1.458	16.524	6	68	91.18%
SMBv23	9.192	12.360	152	204	25.63%
SOCKS	64.092	141.096	122	260	54.58%
SOMEIP	386.136	850.392	692	1.524	54.59%
SSDP	169.968	230.160	418	766	26.15%
SSH	254.024	7.116.608	3.406	46.888	96.43%
Starcraft	90.396	196.416	162	352	53.98%
Syslog	128.340	262.260	230	470	51.06%
TeamViewer	116.064	247.752	208	444	53.15%
Telnet	7.080	8.520	118	142	16.90%
Teredo	107.136	234.360	192	420	54.29%
TFTP	51.336	109.368	92	196	53.06%
TINC	100.440	223.200	180	400	55.00%
TLS	229.370	16.020.578	2.900	35.105	98.57%
Twitter	12.500	12.828	132	136	2.56%
UBNTAC2	106.020	233.244	190	418	54.55%
UbuntuONE	7.114	3.997.352	80	3.252	99.82%
VHUA	80.352	181.908	144	326	55.83%
Viber	686.340	1.487.628	1.230	2.666	53.86%
VMware	217.620	501.084	390	898	56.57%
Wikipedia	24.352	26.832	280	296	9.24%
WireGuard	112.716	255.564	202	458	55.90%
Xbox	229.896	510.012	412	914	54.92%
XDMCP	100.440	213.156	180	382	52.88%
YouTube	14.960	15.360	92	96	2.60%

Table A3. TCP-based Detection Applications and Reduction Rates.

APPNAME	REDUCED BYTES	ORIGINAL B.	REDUCED PACKET	ORIGINAL P.	REDUCTION %
Amazon	49.814	3.358.944	752	9.755	98.52%
CiscoVPN	240	360	4	6	33.33%
Cloudflare	3.432	57.108	52	290	93.99%
FTP_CONTROL	9.612	27.816	146	428	65.44%
Google	556.515	42.745.086	7.630	124.991	98.70%
HTTP	796.644	17.120.664	11.256	59.213	95.35%
HTTP_Proxy	240	360	4	6	33.33%
ICMP	532	1.024	6	12	48.05%
Microsoft365	264	20.034	4	40	98.68%
MsSQL-TDS	2.640	3.600	44	60	26.67%
Playstation	240	360	4	6	33.33%
RDP	480	600	8	10	20.00%
SMBv23	8.700	11.868	144	196	26.69%
SSH	253.900	7.116.484	3.404	46.886	96.43%
Telnet	6.600	8.040	110	134	17.91%
TLS	223.754	16.014.962	2.808	35.013	98.60%
UbuntuONE	1.510	3.991.232	20	3.188	99.96%

References

1. Yazici, M.A.; Oztoprak, K. Policy broker-centric traffic classifier architecture for deep packet inspection systems with route asymmetry. In Proceedings of the 2017 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), Istanbul, Turkey, 5–8 June 2017, pp. 1–5. [CrossRef]
2. Sandvine Inc. Virtual ActiveLogic—Hyperscale Data Plane for Next, Generation Telco Networks. Available online: https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2020/Datasheets/Network%20Optimization/Sandvine_DS_Virtual_ActiveLogic.pdf (accessed on 20 June 2021).
3. Lim, H.K.; Kim, J.B.; Heo, J.S.; Kim, K.; Hong, Y.G.; Han, Y.H. Packet-based network traffic classification using deep learning. In Proceedings of the 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Okinawa, Japan, 11–13 February 2019; pp. 046–051.
4. Zolotukhin, M.; Hämäläinen, T.; Kokkonen, T.; Siltanen, J. Increasing web service availability by detecting application-layer DDoS attacks in encrypted traffic. In Proceedings of the 2016 23rd International Conference on Telecommunications (ICT), Thessaloniki, Greec, 16–18 May 2016; pp. 1–6.
5. Bosshart, P.; Gibb, G.; Kim, H.S.; Varghese, G.; McKeown, N.; Izzard, M.; Mujica, F.; Horowitz, M. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 99–110. [CrossRef]
6. Kim, C. *Programming the Network Dataplane*; ACM SIGCOMM: Florianopolis, Brazil, 2016.
7. Gupta, A.; Harrison, R.; Canini, M.; Feamster, N.; Rexford, J.; Willinger, W.; Sonata: Query-driven streaming network telemetry. In Proceedings of the 2018 conference of the ACM special interest group on data communication, Budapest, Hungary, 20–25 August 2018; pp. 357–371.
8. Wang, S.Y.; Hu, H.W.; Lin, Y.B. Design and Implementation of TCP-Friendly Meters in P4 Switches. *IEEE/ACM Trans. Netw.* **2020**, *28*, 1885–1898. [CrossRef]
9. Yan, Y.; Beldachi, A.F.; Nejabati, R.; Simeonidou, D. P4-enabled Smart NIC: Enabling Sliceable and Service-Driven Optical Data Centres. *J. Light. Technol.* **2020**, *38*, 2688–2694. [CrossRef]
10. Fernández, C.; Giménez, S.; Grasa, E.; Bunch, S. A P4-Enabled RINA Interior Router for Software-Defined Data Centers. *Computers* **2020**, *9*, 70. [CrossRef]
11. Kundel, R.; Nobach, L.; Blendin, J.; Maas, W.; Zimber, A.; Kolbe, H.J.; Schyguda, G.; Gurevich, V.; Hark, R.; Koldehofe, B.; et al. OpenBNG: Central office network functions on programmable data plane hardware. *Int. J. Netw. Manag.* **2021**, *31*, e2134. [CrossRef]
12. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
13. Hang, Z.; Wen, M.; Shi, Y.; Zhang, C. Programming protocol-independent packet processors high-level programming (P4HLP): Towards unified high-level programming for a commodity programmable switch. *Electronics* **2019**, *8*, 958. [CrossRef]
14. The P4.org Applications Working Group. In-Band Network Telemetry (INT) Data Plane Specification. Available online: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf (accessed on 10 March 2021).
15. The P4 Language Consortium. Getting Started with P4 Language. Available online: https://p4.org/p4/getting-started-with-p4.html (accessed on 15 March 2021).

16. Parol, P. P4 Network Programming Language—What Is It All About? Available online: https://codilime.com/p4-network-programming-language-what-is-it-all-about/ (accessed on 21 March 2021).
17. Sgambelluri, A.; Paolucci, F.; Giorgetti, A.; Scano, D.; Cugini, F. Exploiting telemetry in multi-layer networks. In Proceedings of the 2020 22nd International Conference on Transparent Optical Networks (ICTON), Bari, Italy, 19–23 July 2020; pp. 1–4.
18. Sari, A.; Lekidis, A.; Butun, I. Industrial networks and IIoT: Now and future trends. In *Industrial IoT*; Springer: Cham, Switzerland, 2020; pp. 3–55.
19. Butun, I.; Almgren, M.; Gulisano, V.; Papatriantafilou, M. Intrusion Detection in Industrial Networks via Data Streaming. In *Industrial IoT*; Springer: Cham, Switzerland, 2020; pp. 213–238.
20. Zaharia, M.; Xin, R.S.; Wendell, P.; Das, T.; Armbrust, M.; Dave, A.; Meng, X.; Rosen, J.; Venkataraman, S.; Franklin, M.J.; et al. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* **2016**, *59*, 56–65. [CrossRef]
21. Apache Foundation. Apache Flink - Stateful Computations over Data Streams. Available online: https://flink.apache.org/ (accessed on 13 February 2021).
22. Oztoprak, K. Subscriber Profiling for Connection Service Providers by Considering Individuals and Different Timeframes. *IEICE Trans. Commun.* **2016**, *E99.B*, 1353–1361. [CrossRef]
23. Oztoprak, K. Profiling subscribers according to their internet usage characteristics and behaviors. In Proceedings of the 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, USA, 29 October–1 November 2015; pp. 1492–1499. [CrossRef]
24. Sharafaldin, I.; Lashkari, A.H.; Hakak, S.; Ghorbani, A. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In Proceedings of the 2019 International Carnahan Conference on Security Technology (ICCST), Chennai, India, 1–3 October 2019; pp. 1–8.
25. Deri, L.; Martinelli, M.; Bujlow, T.; Cardigliano, A. ndpi: Open-source high-speed deep packet inspection. In Proceedings of the 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, Cyprus, 4–8 August 2014; pp. 617–622.
26. Jurkiewicz, P.; Rzym, G.; Boryło, P. Flow length and size distributions in campus Internet traffic. *Comput. Commun.* **2021**, *167*, 15–30. [CrossRef]