Santa Clara University

Scholar Commons

Computer Science and Engineering Master's Theses

Engineering Master's Theses

4-2023

Comparison of P4 Programmable Software Switches as WiFi Access Points

Kaustubh Pimparkar

Follow this and additional works at: https://scholarcommons.scu.edu/cseng_mstr

 Part of the Computer Engineering Commons

DocuSign Envelope ID: E6F645B9-611D-47FA-8F87-CA127ECC95F2

Santa Clara University

Department of Computer Science and Engineering

Date: April 4, 2023

I HEREBY RECOMMEND THAT THE THESIS PREPARED

UNDER MY SUPERVISION BY

Kaustubh Pimparkar

ENTITLED

Comparison of P4 Programmable Software Switches as WiFi Access Points

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

DocuSigned by:
Dr. Behnam Dezfouli
6ED175D6B91A4FF...

Thesis Advisor
Dr. Behnam Dezfouli

DocuSigned by:
Dr. Sean Choi
8B8EF2A5E9EF4DF...

Thesis Reader
Dr. Sean Choi

A. Amer
A. Amer (Apr 5, 2023 13:24 PDT)

Associate Chairman of Department
Dr. Ahmed Amer

Comparison of P4 Programmable Software Switches as WiFi Access Points

By

Kaustubh Pimparkar

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science and Engineering
in the School of Engineering at
Santa Clara University,

April 4, 2023

Santa Clara, California

Acknowledgments

I would first like to thank my advisor, Dr. Behnam Dezfouli, for his support, advice, and consistent encouragement during my thesis work. His expertise and guidance were invaluable in developing my study, and I will be forever thankful for the opportunity to collaborate with him.

I'd also like to thank my colleagues at SIOTLAB, Ananya Gopal, Chakrapani Chitnis, and Vikram Ramanna; their feedback and support have helped shape my ideas, and I am grateful for the insightful discussions and brainstorming sessions we have had. Big shout-out to Busayo and Priscilla, for being such awesome lab partners.

I am also grateful to my friends and family for their encouragement throughout my academic career. Their faith in my skills has been a continual source of motivation and inspiration in my life.

Comparison of P4 Programmable Software Switches as WiFi Access Points

Kaustubh Pimparkar

Department of Computer Science and Engineering
Santa Clara University
Santa Clara, California

April 4, 2023

ABSTRACT

WiFi wireless access points must adapt to complex network operations as connected devices and their bandwidth requirements continue to rise. The use of software switches supporting Programming Protocol-independent Packet Processors (P4) in wireless access points has drawn significant attention due to their versatility, flexibility, and potential to overcome the limitations of conventional network designs. T4P4S and BmV2 are the two most widely used P4 programmable software switches, and comparing the performance of these switches allows network operators to select the most suitable switch for specific use cases, such as minimizing latency or maximizing throughput while reducing resource consumption of the underlying hardware.

This thesis examines the performance and resource consumption of T4P4S and BmV2. We first examine their design spaces and then evaluate their performance in the presence of various traffic types. Based on our experimental findings, the efficient packet processing pipeline of the T4P4S switch makes it a superior alternative to the BmV2 switch for executing P4 programs. The presented results and analysis provide a better understanding of design decisions and identify potential performance bottlenecks for P4 programmable software switches.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software-Defined Networking (SDN) is a technique that separates the control plane and data plane operations of conventional networking hardware. SDN distributes the data plane over multiple network devices while centralizing the control plane in a software-based controller [1–3]. To further enhance the flexibility of such networks, Programming Protocol-Independent Packet Processors (P4) can describe the data plane of network devices and specify packet processing behavior on different types of network hardware, such as switches. Since P4 is protocol-independent, it can be used to express the behavior of any networking protocol and can also be used to develop customized protocols, which makes P4 versatile. P4 enables network managers to specify how network devices process packets, which gives them greater control over the behavior of the network [4, 5]. P4 and SDN are often used together to develop a network architecture that is more adaptable and programmable in practice [6, 7]. With P4, for example, network devices can prioritize particular kinds of data traffic, such as audio or video traffic [4]. P4 can also describe the entire network's behavior, including how traffic should be routed, and policies should be enforced.

WiFi Access Points (APs) are networking devices that offer wireless connectivity to users and IoT devices such as laptops, smartphones, and thermostats. By using P4 on APs, the AP can handle network traffic more flexibly and effectively. P4 provides the capability to control the packet processing of these devices, allowing

for customization of the network's functionality to meet specific requirements [4,8,9].

Although P4 software switches have several advantages over conventional hardware switches, there is no comprehensive comparison of these switches in terms of the resource utilization of the underlying hardware. Previous research has primarily concentrated on evaluating P4 software switches based on throughput and latency metrics, with little attention paid to their functionality in real-world scenarios. Network performance metrics like latency and throughput of software switches have been the primary focus of previous studies, and comparing resource utilization of the underlying hardware has been neglected. This gap is especially crucial when energy consumption matters or when the AP functionality is implemented on resource-constraint devices such as the Raspberry Pi. Hence, it is beneficial to evaluate P4 software switches by subjecting them to network traffic and measuring the underlying hardware's performance and resource consumption in handling network load.

Our work aims to determine the performance of P4-enabled APs running on a resource-constraint device—the Raspberry Pi board. This study enables researchers and network operators to understand, enhance and deploy the performance of P4-enabled APs based on application demands and required flexibility. In this thesis, we compare and analyze the most widely used P4 programmable software switches, BmV2 and T4P4S P4, throughput, energy consumption, CPU utilization, and memory usage while switching different types of traffic. This is an important step in determining if these switches are suitable for the type of network traffic being considered. Furthermore, this thesis aims to identify the P4 switch that can provide the highest throughput while consuming the least resources. Thanks to its more efficient packet processing pipeline, we will show that T4P4S performs better than BmV2 in both throughput and resource consumption. Our work lays the founda-

tion that can be used to develop P4 switches with higher throughput and efficient resource utilization that are tailored specifically for AP networks.

The rest of this thesis is organized as follows. Chapter 2 overviews the existing work. A review of the components used in our research work and the design space of software switches are explored in Chapter 3. In Chapter 3, we also analyze the P4 software switches' design aspects to determine their strengths and weaknesses, then compare and contrast those characteristics. Chapter 4 describes the experimental design, testing methodologies, and measurement tools used for this research. We also present an analysis of the results and highlight the advantages that T4P4S has over BmV2 in terms of throughput and resource consumption. Finally, in Chapter 5, we discuss our implications for designing and implementing P4 software switches in APs and potential future research in P4-enabled software switches.

Chapter 2

Related Work

In recent years, software switches have become popular as a versatile and cost-effective way of performing network operations. Several research papers investigate the performance of modern software switches. For instance, Ben et al. [10] compare Open vSwitch (OVS) throughput versus Linux's bridge and the Linux kernel's IP forwarding. According to their research findings, OVS cannot achieve 2 Gbps when using 64B packets. The same authors conduct additional research into OVS throughput and its latency. The preliminary benchmarks with Lagopus [11] and Ryu Controllers [12] produce a throughput of fewer than 20 Mpps in [13] when using 64B packets. Rajagopal et al. [14] test OVS-DPDK throughput with 1 Gbps NICs and port/flow mirroring. Their results showed that OVS-DPDK achieved significantly higher throughput than the standard OVS implementation, with throughput improvements ranging from 3x to 10x, depending on the traffic pattern.

Liu et al. [15] conducted a comparative analysis of various software switches in a virtualized environment. The switches considered in the study were Snabb [16], OVS, OVS-DPDK, and Linux Bridge. The author's primary objective was to evaluate the switches' performance in terms of throughput and latency, highlighting their architectural differences and support for virtual network functions (VNFs) and Network Functions Virtualization (NFV). The study findings indicate that OVS-DPDK demonstrated superior performance in both throughput and latency due to the integration of the DPDK library [17], enabling bypassing the kernel and

direct interaction with the NIC hardware. On the other hand, Snabb demonstrated lower latency than OVS and Linux Bridge, primarily due to its streamlined and lightweight design, resulting in reduced processing time and minimal overhead.

Singh et al. [18] conducted a comparative study between Virtual Path (VP) and OVS-DPDK. Their study aimed to evaluate the performance of these switches in terms of throughput and packet loss under various scenarios of VNF loopbacks. The study results showed that VP outperformed OVS-DPDK in terms of throughput, especially in scenarios where the number of VNF loopbacks was high. However, OVS-DPDK showed lower packet loss compared to VP in some scenarios. These results indicate that VP leverages vectorization techniques to process multiple packets simultaneously, resulting in lower packet loss. On the other hand, OVS-DPDK uses a polling mechanism to achieve high packet processing rates, which may result in higher packet loss under certain conditions.

In another study, Niu et al. [19] compared the throughput and latency of two virtualized switches, BESS [20] and ClickOS [21], in a service chaining loopback scenario. The study aimed to evaluate the performance of these technologies in terms of packet forwarding delay, throughput, and CPU utilization. The study results showed that BESS outperformed ClickOS in terms of packet forwarding delay and throughput, while both switches had similar CPU utilization. Furthermore, their results show that BESS uses a user-space packet processing model that eliminates the overhead of kernel processing, resulting in lower packet forwarding delay and higher throughput. On the other hand, ClickOS uses a lightweight hypervisor that provides VNFs with low overhead, resulting in similar CPU utilization compared to BESS.

Rodriguez et al. [22] compare the throughput of BESS, VP, and OVS-DPDK using only physical interfaces of the host computer. In their benchmarking experi-

ment, the authors evaluated the performance of these software switches with packet sizes spanning from 64 to 9000 bytes using a 10 Gbps network link. BESS outperformed both VP and OVS-DPDK in terms of throughput, as demonstrated by their results. BESS achieved up to 24.52 Mpps with 64-byte packets, whereas VP and OVS-DPDK only achieved up to 19.17 Mpps and 16.45 Mpps, respectively, with the same packet size. Additionally, the authors observe that VP has lower latency than BESS and is more suitable for latency-sensitive applications.

In addition to the research that has been done in the past, several studies have been conducted to compare the processing costs of software switches on the underlying hardware. Agesen et al. [23] analyze software-based and hardware-based virtualization solutions for the x86 architecture and their influence on system performance under different circumstances. Their results indicate that hardware-based solutions outperform software-based techniques when the CPU is the bottleneck. This study offers useful insight into the trade-offs between software-based and hardware-based virtualization strategies and demonstrates the performance advantages of hardware-based alternatives. Smith et al. [24] analyze the CPU usage and throughput of SR-IOV, Netmap Passthrough, OVS-DPDK, and Snabb in two different test scenarios: inter-VM forwarding and 1-VNF loopback. Our work is unique because we consider the resource utilization of P4 programmable software switches on the underlying hardware.

In [25], Fernandes et al. suggests a novel approach to the design of software switches for cloud networking. Their research utilizes a hybrid architecture consisting of a software-implemented packet processing pipeline and a hardware-implemented packet forwarding engine. This architecture aims to increase the switches' throughput while reducing CPU and memory use. Their work reveals that this hybrid architecture can significantly improve the forwarding performance of a

switch, achieving a substantial improvement in packet processing rate and a reduction in CPU use and memory consumption compared to standard software switches. This study is relevant to our thesis since the authors compare the processing cost of the software switches in different scenarios; however, none of these studies consider the energy consumption of software switches. In a comparative study by Redruello et al. [26], several software switches' CPU utilization, throughput, and latency, including OVS, Linux Bridge, and Click, were analyzed. A test bed was developed to evaluate the performance of the switches under different packet sizes and throughputs. Their findings suggest that packet size and flow rate significantly impact the performance of the switches. Furthermore, the study underscores the importance of optimizing the configuration and parameters of software switches to achieve optimal performance. Rang et al. [27] compared OVS and Linux containers to determine their effectiveness in handling NFV workloads. They evaluated the performance of OVS and Linux containers based on CPU and memory usage, throughput, and latency. Their results indicate that OVS outperformed Linux containers in terms of packet forwarding rate and network speed, despite requiring more CPU and memory resources.

Chapter 3
Background

3.1 P4 Programming Language

P4 is a programming language designed to enhance the data plane. P4 was developed in response to the issues posed by the need for flexible, adaptable, and efficient packet processing. P4 applications are developed in a high-level programming language that describes how network packets are processed [4]. This includes configuring the packet header format, processing the packet, performing actions (such as modifying or deleting the packet), and updating packet metadata.

Conventional network devices such as switches and routers are often manufactured from the bottom up, with device vendors relying on prebuilt chips from third-party manufacturers. These fixed-function chips determine the functionality of such networking devices. Adding new capabilities to the device can be time-consuming because the chip's architecture is fixed and cannot be easily modified. In contrast, P4-enabled devices allow users to have a "top-down" methodology where the programmer defines the network feature set in a P4 program, which is then compiled and loaded into the network device. This requires a chip to have a set of programmable and fixed-function blocks in the packet processing pipeline.

The P4 language specification defines a model describing a specific networking device's packet processing pipeline components. This model describes how different function blocks of a device work together to handle packets [28]. The device

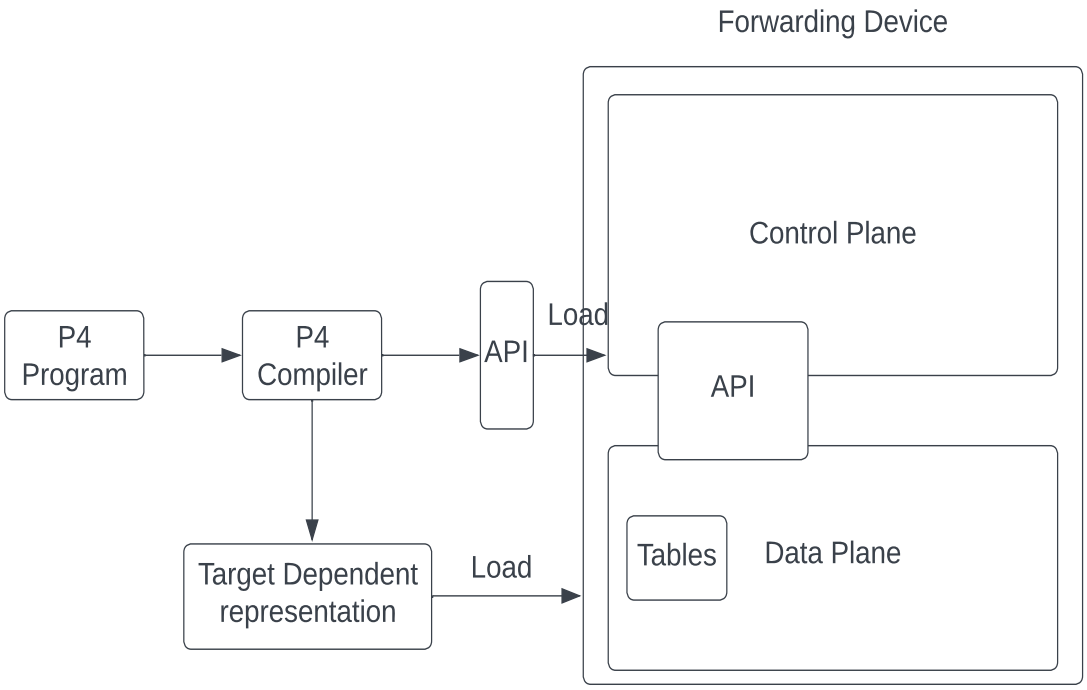


Fig. 3.1: P4 Program Flow

manufacturer provides all the information required about the function blocks in the model that can and cannot be programmed. The first block in this pipeline is the parser, which collects packet headers and sends them to the subsequent processing control blocks. After that, the control blocks do further work on the packet, like running match-action table chains, checking and recalculating checksums, and deparsing the packet.

Figure 3.1 shows the general flow of P4 program compilation to a network device. Once the user creates a P4 program to describe the forwarding behavior of a network device, the program is passed to a compiler, which converts the P4 into a target-dependent representation of the code to be loaded into the device (for example, JSON in the case of BmV2). Once the code has been loaded into the device, control plane software such as P4Runtime can modify the match/action table entries or the tables themselves [4]. Protocol independence is one of P4's major

benefits. P4 enables network managers to create their packet processing functions, enabling them to manage new protocols and network designs. This flexibility is important as networks evolve and new applications and protocols are developed.

3.2 Behavioral Model Version 2 (BmV2)

Behavioral Model Version 2 (BmV2) is a P4-enabled software switch implementation frequently used in network function virtualization (NFV). The BmV2 software switch is based on the P4 behavioral model and provides network researchers and developers with high flexibility. In addition, BmV2 facilitates network function experimentation and quick prototyping [29]. BmV2 is intended to be modular and versatile, allowing researchers and developers to test new network protocols and topologies [30]. In addition, it can be used to develop network functionalities such as load balancing, traffic engineering, and security rules since it supports a variety of P4-based packet processing pipelines, including stateful and stateless processing.

Figure 3.2 illustrates the design of a BmV2 simple switch. The BmV2 switch operates in userspace and retrieves packets from the NIC driver using the pcap library [30].

Figure 3.3 shows multiple stages that comprise the packet processing pipeline in the BmV2 switch, beginning with the ingress parser, which converts the packet from its raw bit form into headers based on a programmer-specified parser specification. This stage also determines which packet headers are recognized (such as Ethernet or IP) and their processing order. Following packet parsing, an ingress match-action, also known as an ingress control function, determines the action for further packet processing. Programmers can define Longest Prefix Match (LPM) or exact header matching, followed by the action to be taken on the packet, such as forwarding or

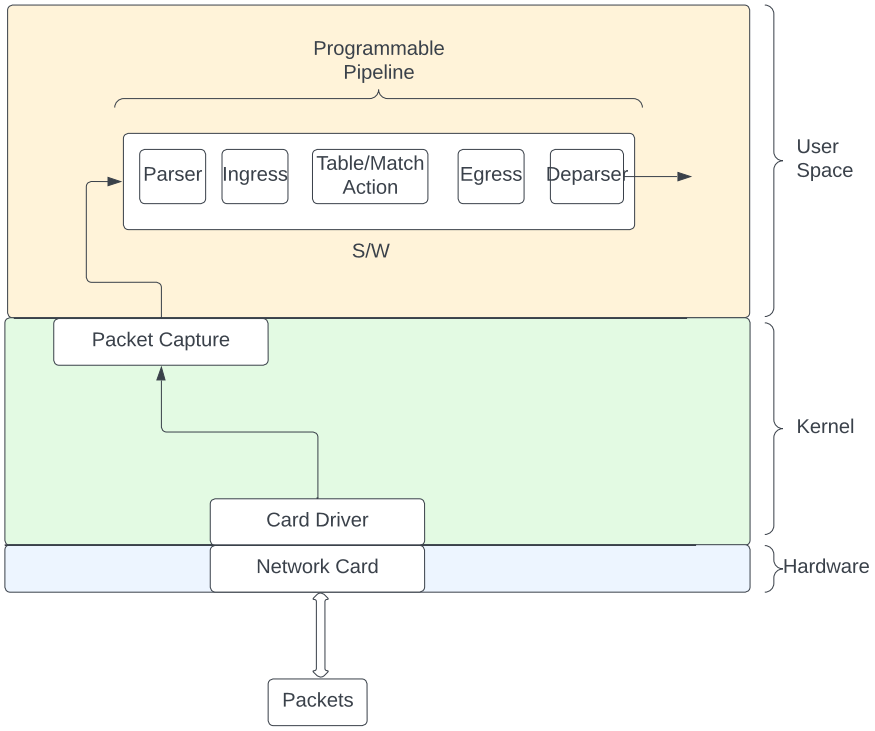


Fig. 3.2: Simple Switch Architecture

dropping it. Once complete, the packet is placed in the egress deparser's queue for egress processing. An egress match-action or egress control function processes the packet upon dequeuing. The egress deparser specification does the separation and deparsing of packet headers into a bit representation on output. The packet is then transmitted from the switch [31].

In the research and development community, BmV2 is frequently used for testing and experimenting with novel network protocols and topologies. In addition, it has been used in several research initiatives, such as congestion control, load balancing, and network security [30]. BmV2 is also utilized in academic contexts to teach SDN principles [30] and flexibility of P4-enabled software switches.

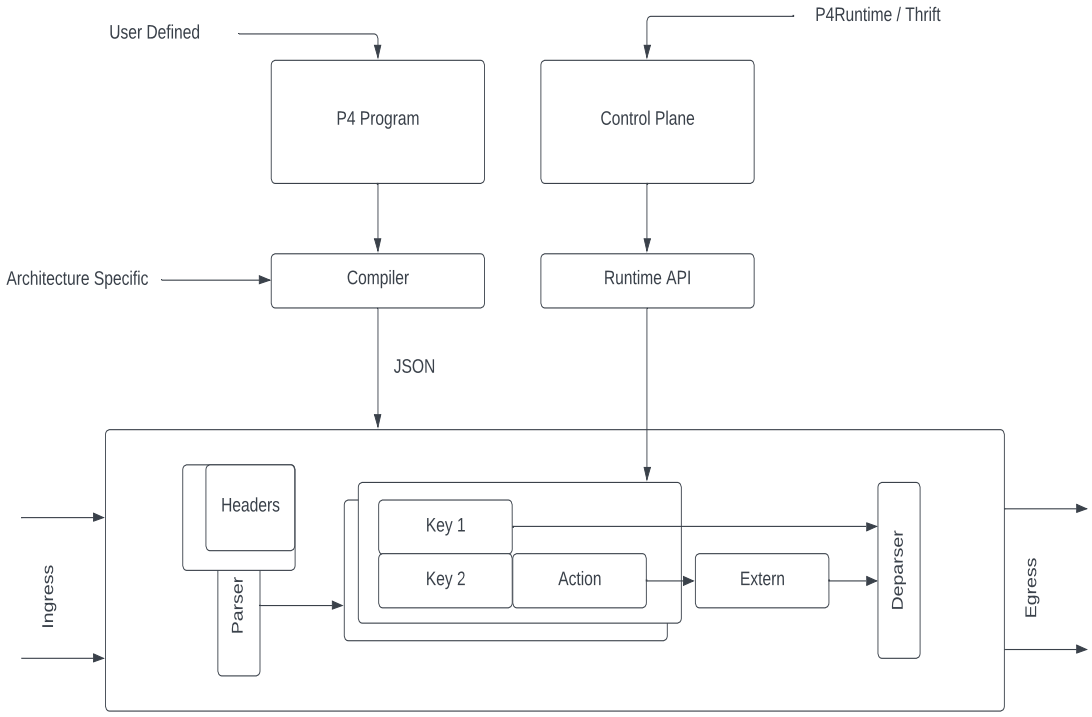


Fig. 3.3: BmV2 Packet Processing Pipeline

3.3 T4P4S

T4P4S is a $P4_{16}$ programming language multi-target software switch. T4P4S was developed to overcome the shortcomings of current $P4_{16}$ compilers, which were either single-target or lacking essential features such as multi-target abstraction support [32]. The term target represents the underlying hardware that runs the P4 software switch. Due to the differences between the various architectures that can host P4 software switches, implementing the P4 programming language in a portable manner is difficult. The suggested solution is to create a separate compiler for each target architecture; however, this approach is not viable for all targets.

T4P4S switch aims to achieve retargetability by compiling the high-level P4 program's data plane pipeline without modifying it for specific targets. The design includes a hardware abstraction layer that separates the switch software into

two parts: the Networking Hardware Abstraction Library (NetHAL) and Core. NetHAL implements all hardware-dependent features in a protocol-independent manner, whereas Core is the pipeline implementation compiled from the declarative P4 program. Separating these two components increases modularity, simplifies maintenance, and enables the switch software to be retargeted with minimal effort. A general interface (abstraction layer) unifies diverse architecture goals, and preliminary findings indicate that this approach can compete with off-the-shelf solutions for traditional data plane tasks [33].

The T4P4S switch generates a switch program consisting of a core divided into slow and fast path components that can interact with different hardware targets via the NetHAL interface. The switch also provides a low-level control plane interface for filling match-action tables, setting default actions, querying counters, and passing data plane information to the control plane. The Core component implements forwarding logic at a hardware-independent abstraction level derived from the P4 description. The table applications and forwarding actions are translated into C functions, and the Core implements table applications' control and data flow. In contrast, the NetHAL implements actual table lookups and hardware-level functions of the matched actions.

T4P4S is built to be extremely adaptable and scalable, making it appropriate for a wide range of applications. It is written in C++ and supports $P4_{14}$ and $P4_{16}$. T4P4S supports packet processing at several pipeline levels, such as parsing, matching, and action execution. T4P4S is intended to be simple to use and to interact seamlessly with current network infrastructure. It supports different output formats for multiple architectures and has a simple command-line interface for setting the switch [32].

3.4 P4Pi

P4Pi is a platform for running P4 applications on Raspberry Pi devices [34]. It is developed on top of Raspbian Lite OS, enabling users to run P4 programs on RPi devices using P4Pi, a low-cost way to experiment with and develop networking applications. In addition, P4Pi configures the Raspberry Pi as an AP upon installation, making it simple to connect to and configure [34]. Additionally, P4Pi includes all the components required to install and execute P4 [35]. This eliminates the need for users to install additional software or configure their systems, making it easier to get started with P4 programming. P4Pi supports two P4-compliant software switches, BmV2 and T4P4S. These switches enable the testing and execution of P4 applications on a Raspberry Pi device. P4Pi also allows users to compile and load P4 applications on these switches, allowing them to test and experiment with their programs in a real-world setting [34].

Overall, P4Pi is a strong platform that offers a comprehensive solution for executing P4 applications on Raspberry Pi devices. Its user-friendly AP, pre-installed software components, support for P4-compatible switches, and program loading capabilities make it an excellent choice for developers and students interested in experimenting with P4 programming on low-cost hardware.

3.5 Software Switch Design Space

3.5.1 Architecture

The architecture of a software switch plays a critical role in determining how packet processing is organized and implemented. According to [36], the two primary cate-

gories of software switch architectures are *self-contained* and *modular*.

A self-contained software switch is designed to operate as a single process, with all processing functions occurring concurrently. This architecture requires minimal effort to set up and is hence self-contained. The data flow in this architecture is predetermined, and all processing functions occur in a single process. As a result, self-contained switches offer better performance and are more cost-effective than modular switches. In addition, they are easy to set up and configure, making them popular for simple network configurations.

A modular software switch is designed with preset network functions that can be combined into a "forwarding graph". Each node in the forwarding graph can be an independent thread or process, offering greater flexibility in adding specialized network operations. A modular design allows for dynamic configuration and modification of the switch's behavior, making it useful for complex networks that require customized processing functions and are subject to frequent changes. However, modular switches are more complex to set up and manage than self-contained switches. They require careful design and configuration of the forwarding graph, which can be time-consuming and may increase the risk of errors. Additionally, modular switches suffer from additional overhead due to the communication between nodes in the forwarding graph. The main advantage of using modular switches is their flexibility. They offer greater flexibility in adding specialized network operations, as new nodes can be easily added to the forwarding graph. This allows for customization of the packet processing pipeline to meet the network's specific needs. In contrast, self-contained switches have a predetermined data flow, making them less flexible than modular switches.

The choice of software switch architecture depends on the specific requirements of the network and the level of flexibility and customization needed for packet pro-

cessing. For example, self-contained switches are a popular choice for simple network configurations. In contrast, modular switches are useful for complex networks that require customized processing functions and are subject to frequent changes. The decision between the two architectures depends on the trade-off between performance, scalability, flexibility, and complexity.

3.5.2 Programming Paradigm

The development of software switches is significantly impacted by their intended use cases, particularly regarding the packet processing paradigm they employ. Structured programming involves the use of pre-defined rules and instructions for packet processing, whereas match/action programming utilizes packet classification algorithms to match header data and execute corresponding actions. Various software switches often use structured programming for traffic routing across virtual network functions (VNFs). BmV2 and T4P4S switches utilize the match/action paradigm, as noted in [37].

3.5.3 Processing Model

The packet processing model can be classified into *Run-to-Completion (RTC)* and *pipeline* models. In the RTC model, a single thread or core handles the entire packet processing logic before forwarding or dropping the packet. This model is simple to implement, and it can achieve high performance on modern multi-core processors. In contrast, the pipeline model divides packet processing into multiple stages, each handled by a different thread or core. Each thread processes a specific part of the packet processing logic, and the packet moves from one stage to another until it is forwarded or dropped. This model is more complex and requires careful

coordination between threads to avoid race conditions and deadlocks.

Most packet processing switches, such as T4P4S, use the RTC architecture to reduce context-switching costs and achieve high performance. This means that a single thread handles the entire processing logic for a packet before moving on to the next packet. On the other hand, BmV2 uses the pipeline paradigm, where multiple threads process different stages of the pipeline for packets.

3.5.4 Programming Language

The choice of a programming language for a software switch is influenced by a number of criteria, including performance requirements and programmability. Due to their efficiency, broad feature set, and portability across many platforms, the majority of high-speed packet processing software frameworks are often developed with C and/or C++. T4P4S is developed in C, whereas BmV2 has its code base in C++; both of these switches have Python-based runtime implementations.

Table 3.1: Taxonomy of P4 Software Switch

	Architecture	**Processing Model**	**Programming Paradigm**	**Programming Language**
`BmV2`	Modular	Pipeline	Match/Action	C++/Python
`T4P4S`	Self-Contained	Run to Completion	Match/Action	C/Python

Chapter 4

Test Methodology and Evaluation

4.1 Measurement Platform

For our experiments, we use P4Pi on a Raspberry Pi 4 Model B. This board uses a Broadcom SoC: Broadcom BCM2711, quad-core (ARM v8), 4GB Memory, 5.0 GHz IEEE 802.11ac wireless and Gigabit Ethernet. For the traffic generation, we used Ubuntu 22.04 installed on a bare metal server.

4.2 Energy Measurement

Energy Measurement Platform for IoT devices (EMPIOT) [38] is a system developed to accurately measure the energy consumption of IoT devices in real time. EMPIOT provides precise energy consumption measurements and enables developers to evaluate the energy efficiency of their applications. In addition, the platform is designed to be a cost-effective, user-friendly solution that can easily adapt to different scenarios.

EMPIOT comprises an INA219 shield that is linked to the host Raspberry Pi device, which collects energy consumption data. The IoT device under test is then powered through this setup, and the collected data is transmitted to the host computer for analysis. EMPIOT features versatile raw data collection and energy measurement capabilities and offers flexible support for various measurement

parameters, such as measurement interval, range, and duration, to accommodate diverse measurement requirements.

4.3 Testbed

Figure 4.1 show the testbed setup used for conducting experiments on the software switch under test. We first set up the Raspberry Pi with an image of P4Pi and use it as an AP. P4Pi provides T4P4S and BmV2 switches as separate Linux services out of the box. While conducting experiments on each switch, we made sure that the other switch's service had been stopped and disabled so that there was no conflict. To ensure a fair comparison between the switches, a very basic packet forwarding P4 program was loaded into the switch under test. This was to ensure that the complexity of the P4 program in no way hampered the performance of these switches. Next, we used IPerf3 to generate TCP traffic from the client to a server, connected to P4Pi as shown in Figure 4.1. We monitored the traffic with Wireshark on each of the hosts' interfaces to capture pcap files. The collected pcap files were then examined to identify network performance factors.

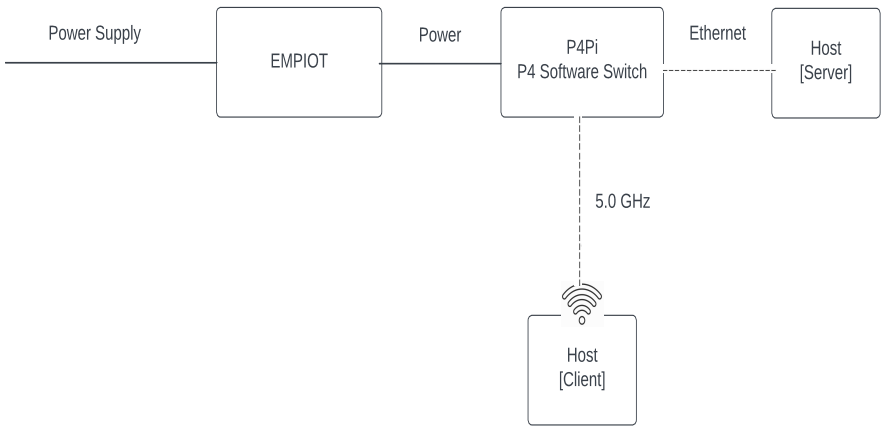


Fig. 4.1: Testbed Setup

EMPIOT was installed on a second Raspberry Pi to measure power consumption, which was then used to power the AP. Three different energy measurements were recorded: baseline measurement when no traffic was flowing, BmV2 measurement when traffic was passing through the BmV2 switch and similarly T4P4S measurement. During these measurements, 600,000 timed samples were collected with EMPIOT, and Python was used to analyze the collected data. These data points were plotted over box plots and regression lines to understand the trend and distribution of the data. CPU and memory consumption data were collected directly from the AP using the psutil library [39] and analyzed. The Raspberry Pi AP was rebooted between each experiment to ensure all caches and buffers were cleared. This was important to ensure that each experiment was conducted under identical conditions and avoid potential biases that could affect the results.

4.4 Throughput Evaluation

Figure 4.2 shows the comparison between the throughput capabilities of T4P4S and BmV2, considering different packet sizes. The T4P4S processing model has undergone substantial rounds of improvement, which is the primary reason why it performs noticeably better than BmV2 in terms of throughput. The RTC paradigm of T4P4S ensures that each thread operates on a single packet at a time, which reduces resource contention between the threads and increases throughput. On the other hand, BmV2 uses the pipeline paradigm, which separates packet processing into stages, and each thread handles one stage of the pipeline. This approach leads to threads waiting longer to acquire resources for packet processing, thereby reducing throughput.

Another reason that leads to lower throughput of BmV2 is that it uses atomic

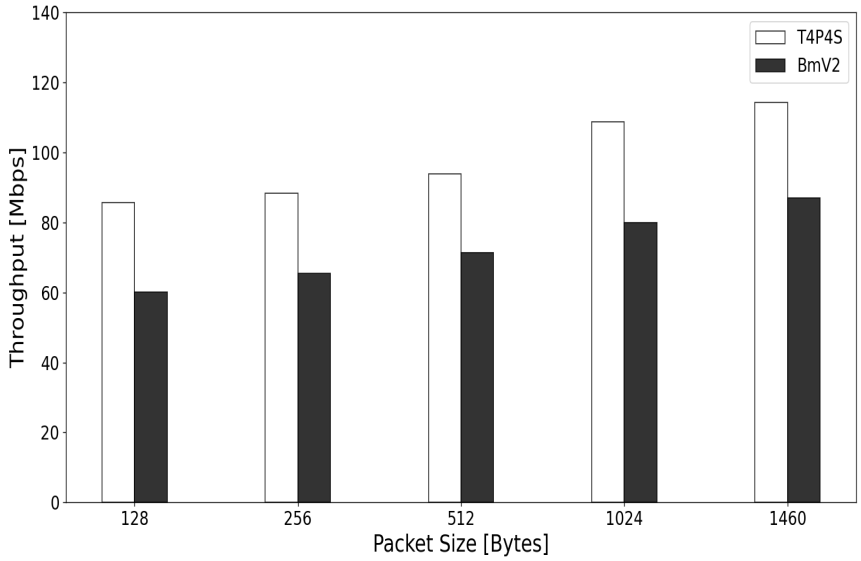


Fig. 4.2: Throughput Comparison

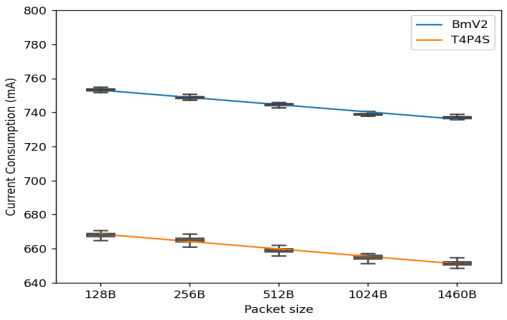
instructions to lock resources between multiple match/action table pipelines. In contrast, T4P4S avoids atomic instructions by using lock-free double buffering to avoid resource locking when threads are accessing the match/action tables to process packets. This approach involves maintaining two copies of the same table, with each copy consisting of active and passive entries for the table. In case a thread has to modify these tables, modifications are first made to the passive replica, and after the modifications have been made, the active and passive duplicates are exchanged after a specific wait time. This ensures consistency for additional threads reading from the replica in parallel. The changes are subsequently propagated to the replica, which is now inactive. This approach makes sure that the active duplicate is never modified and reading entries from the table for the threads remains unlocked, which enhances performance.

4.5 CPU, Energy Consumption, and Memory Consumption

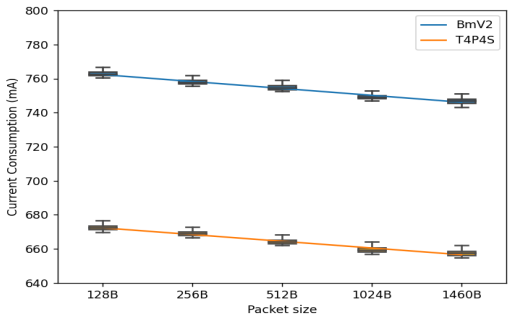
Figures 4.3 and 4.4 compare the CPU and energy usage of T4P4S and BmV2 software switches at varied flow rates and packet sizes. T4P4S utilizes the resources of the underlying hardware better than BmV2, even under heavy network loads, as demonstrated by the results.

T4P4S adopts a thread-per-packet architecture, increasing the chances that a thread will operate on the same core with the match table action execution cached for the packets. This decreases contention for shared resources, such as match/action tables, among the threads. This also ensures that threads are efficiently scheduled and executed without unnecessary overhead, reducing CPU and energy usage. In contrast, the pipeline architecture of BmV2 results in increased competition for shared resources, such as cache, between the threads, resulting in greater CPU and energy usage.

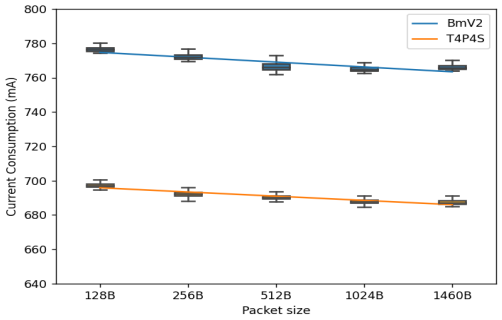
Figure 4.5 shows the memory consumption of two software switches, T4P4S and BmV2, for different packet sizes and under four different flow rates. T4P4S has slightly better performance in terms of memory usage than BmV2. This is because the packets in the queue are processed more rapidly in T4P4S than in BmV2. The graph indicates that with an increase in flow rate and packet size, memory usage increases because increased flow rate or packet size leads to a greater volume of packets that require storage and processing in the software switch's memory. Also, T4P4S utilizes a memory pool allocator, a more efficient memory management technique than general-purpose memory allocation strategies [32]. The memory pool allocator allocates a block of memory in advance and uses it as a source of memory for small, fixed-size allocations. This technique enables faster memory allocation and



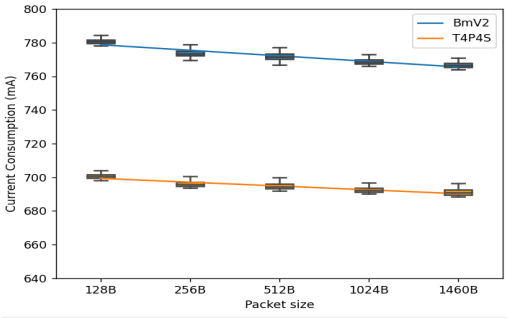
(a) Flow Rate: 10Mbps



(b) Flow Rate: 50Mbps

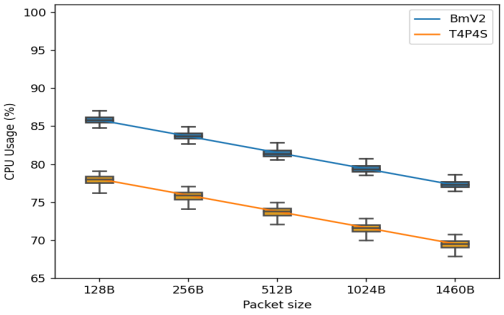


(c) Flow Rate: 80Mbps

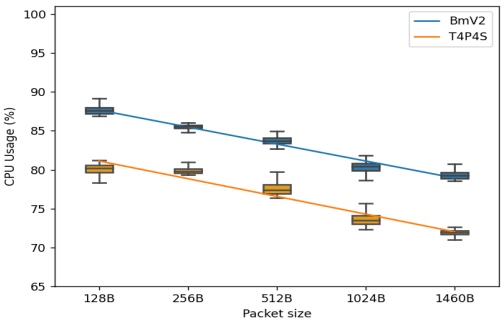


(d) Flow Rate: 100Mbps

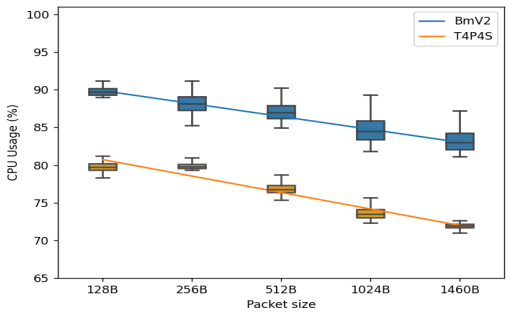
Fig. 4.3: Energy Consumption



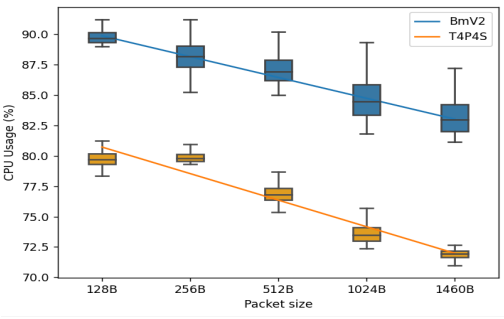
(a) Flow Rate: 10Mbps



(b) Flow Rate: 50Mbps

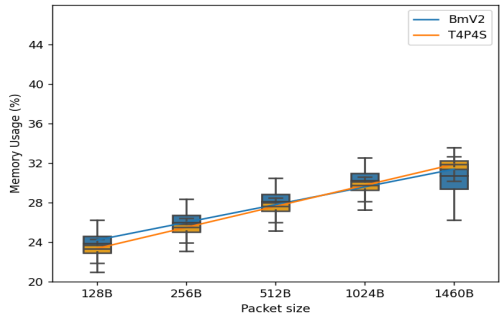


(c) Flow Rate: 80Mbps

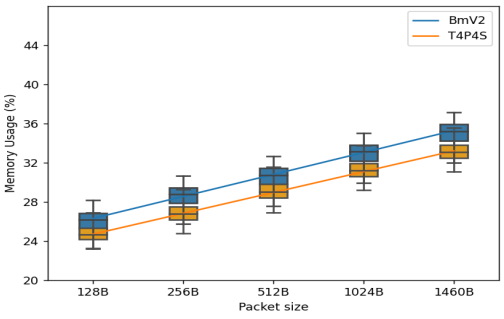


(d) Flow Rate: 100Mbps

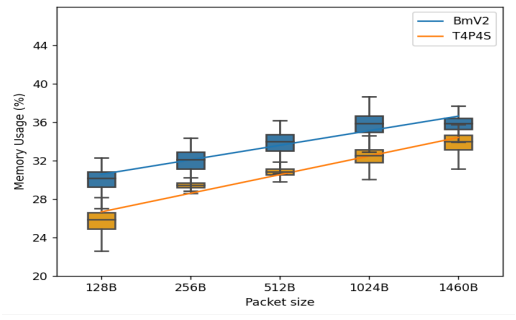
Fig. 4.4: CPU Utilization



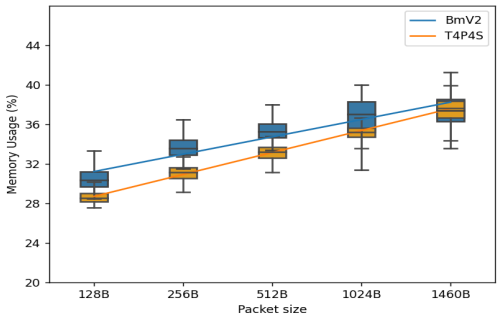
(a) Flow Rate: 10Mbps



(b) Flow Rate: 50Mbps



(c) Flow Rate: 80Mbps



(d) Flow Rate: 100Mbps

Fig. 4.5: Memory Utilization

deallocation compared to general-purpose memory allocators, as it eliminates the need to request and release memory from the operating system for each allocation and deallocation process. This method also decreases the danger of memory fragmentation, which can happen when memory is repeatedly allocated and deallocated for packet processing, resulting in wasted memory space. BmV2 uses a general-purpose memory allocator, which is less efficient for tiny, fixed-size memory allocations. This results in increased memory consumption and fragmentation, which harms the memory usage of the BmV2 switch.

Chapter 5

Conclusion and Future Work

In this study, we began with a discussion of prior work undertaken in the comparison of software switches. We found that although comparative studies of software switches exist, there is no research conducted on the resource utilization of P4-enabled software switches.

Chapter 3 provided an analysis of the architectures of the two switches under consideration, T4P4S and BmV2, and a brief overview of the components used in our experiments. Additionally, the chapter explored the software switch design space and highlighted the notable differences and similarities that exist between the P4-enabled software switches, which must be taken into account when designing such switches. The experiment results presented in Chapter 4 show that T4P4S uses lesser resources and has a greater throughput than the BmV2. This research is based on the fact that T4P4S is architecturally distinct from BmV2 in terms of how packets are processed in these switches. In particular, we identified that the RTC paradigm used by T4P4S leads to reduced context switching costs, which, in turn, results in lower CPU and energy consumption. Furthermore, the RTC paradigm minimizes resource contention between threads and enhances the overall throughput of T4P4S when compared to the pipeline architecture of BmV2. In the pipeline model, packet processing is split into stages, resulting in more frequent context switches during packet processing, which increases the processing cost. Additionally, the pipeline architecture causes longer wait times for threads to acquire resources,

leading to reduced throughput.

While our study compares P4-enabled software switches, evaluating the performance and efficiency of various P4 compilers and target platforms is a potential future addition. The choice of compiler and target platform can significantly influence the overall performance of a P4-enabled switch, even though P4 offers a high level of flexibility and programmability [40]. Parameters such as energy consumption and scalability are among the most important ones when performing such study. In addition, future research may investigate the effect of different network layouts and topologies on the performance of P4-based networks and compare the performance of P4-based networks to that of conventional, fixed-function network hardware.

Bibliography

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Wang, and N. Gude, "Towards an open, extensible, and scalable network control plane," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 68–73.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] P. Bosshart, D. Daly, R. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, K. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[5] P. Bosshart, D. Daly, D. Estrin, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, K. Talayco, A. Vahdat, and G. Varghese, "P4runtime: How a protocol-independent logical switch api enables sdn," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 498–511.

[6] E. Haleplidis, N. Finn, B. Varga, S. Jalali, S. Amante, D. Lübben, and S. Bryant, "P4 language: Control plane meets data plane," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2015, pp. 1–2.

[7] Q. Hu, Y. Li, Y. Liang, G. Han, and J. Zhou, "A survey on software-defined network and openflow: from concept to implementation," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2017, pp. 1959–1964.

[8] M. U. Farooq, K. Saleem, and M. Waseem, "A survey on internet of things architectures," *Journal of Network and Computer Applications*, vol. 84, pp. 23–44, 2017.

[9] S. M. M. V. G. Gustavo Caiza, Santiago Chiliquinga, "Software-defined network (sdn) based internet of things within the context of low-cost automation," *IEEE International Conference*, 2018.

[10] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, K. Amidon, M. Wang, and A. Vahdat, "Design and implementation of the open vswitch," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 2, pp. 1–4, 2015.

[11] M. Kobayashi, M. Katsube, K. Kozu, K. Sajima, and Y. Ohara, "Lagopus: an openflow switch conforming to the specification and beyond," in *Proceedings of the 2013 ACM SIGCOMM conference on SIGCOMM*, 2013, pp. 475–476.

[12] R. Hasegawa, M. Kobayashi, K. Matsuzawa, K. Izumi, and K. Sajima, "Ryu: A next generation openflow controller for openstack," in *2014 IEEE 22nd International Conference on Network Protocols (ICNP)*, 2014, pp. 1–2.

[13] D. Merich, F. Raumer, F. Wohlfart, and G. Carle, "Performance characteristics of virtual switching," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 120–125.

[14] R. Rajagopalan, M. Imbriaco, G. Nogueria, and C. Schlesinger, "Network functions virtualization on cots: Performance opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 148–154, 2016.

[15] F. Liu, H. Qian, R. Han, M. Li, and X. Zhang, "Virtual switch performance: netmap vs sr-iov," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 110–117, 2016.

[16] Snabb, "Snabb: Simple and fast packet networking," https://snabb.io/, 2023, accessed on: April 4, 2023.

[17] DPDK, "DPDK," https://www.dpdk.org/, accessed on: April 4, 2023.

[18] V. Singh, R. Namdeo, and A. Verma, "Performance comparison of virtual switches in software defined networks," *International Journal of Computer Networks and Applications*, vol. 6, no. 2, pp. 23–29, 2019.

[19] Z. Niu, H. Xu, L. Liu, Y. Tian, P. Wang, and Z. Li, "Unveiling performance of nfv software dataplanes," pp. 13–18, 2017.

[20] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "Softnic: A software nic to augment hardware," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155, May 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-155.html

[21] F. Martins, I. Ahmed, A. Akella, K. Gopalan, J. Haridas, L. Iftode, and J. Rexford, "Clickos and the art of network function virtualization," in *11th USENIX*

Symposium on Networked Systems Design and Implementation (NSDI 14). USENIX Association, 2014, pp. 459–473.

[22] J. Rodriguez, A. Wundsam, K. Xu, N. Egi, and S. Seo, "Benchmarking p4 switches: a survey," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 48–54, 2019.

[23] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," *ACM Transactions on Computer Systems*, vol. 40, no. 2, pp. 1–23, 2021.

[24] J. Smith and S. Brown, "Performance analysis of sdn switches with hardware and software flow tables," *Journal of Network and Computer Applications*, vol. 96, p. 102863, 2021.

[25] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to bofuss: The basic openflow userspace software switch," *Journal of Network and Computer Applications*, vol. 165, p. 102685, September 2020.

[26] M. Redruello, R. González, D. Melendi, J. L. García-Dorado, R. Rubión, and A. Azcorra, "Comparative analysis of software switches for cloud networking," *Computer Networks*, vol. 102, pp. 172–187, 2016.

[27] T. Rang, "[pdf] nfv performance benchmarking with ovs and linux containers | semantic scholar," https://www.semanticscholar.org/paper/NFV-performance-benchmarking-with-OVS-and-Linux-Rang/9ceb68be095f0557c5aaa97b83bfe743e9cbc265, 2019, [Accessed: March 19, 2023].

[28] P. Bosshart, D. Daly, B. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, E. Talayco, A. Vahdat, G. Varghese, D. Walker, and H. Xu, "P4 language specification," https://p4.org/p4-spec/docs/P4-16-v1.2.2.html, 2021, [Online; accessed 15-March-2023].

[29] M. Jafarian, H. Kim, S. Son, M. Caesar, and N. Feamster, "Beyond OpenFlow: A Deep Dive into the Programmable Dataplane," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, ser. CoNEXT '17. ACM, 2017, pp. 17–30, [Online; accessed March 13, 2023].

[30] "Bmv2," https://github.com/p4lang/behavioral-model, accessed: 2023-03-13.

[31] A. Shukla, S. Fathalli, and T. Zinner, "P4CONSIST: Toward consistent P4 SDNs," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Dallas, TX: IEEE, Nov. 2019, pp. 1–7.

[32] G. Bianchi, A. Pescapè, and G. Ventre, "T4p4s: a multitarget p4 compiler," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2018, pp. 1–7.

[33] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, and G. Carle, "High-performance match-action table updates from within programmable software data planes," in *Proceedings of the 16th ACM International Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: ACM, 2021, pp. 121–136. [Online]. Available: https://doi.org/10.1145/3493425.3502759

[34] S. Laki, R. Stoyanov, D. Kis, R. Soule, P. Vörös, and N. Zilberman, "P4pi," vol. 51, no. 3. ACM, 2021, pp. 17–21.

[35] "P4 language specification," https://p4.org/p4-spec/docs/, accessed: March 13, 2023.

[36] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, and S. Azodolmolky, "Software-defined networking: A comprehensive survey," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2014, pp. 1–9.

[37] M. Izquierdo, P. L. Rodriguez, and P. J. Marron, "Evaluation of software switch architectures for NFV," *Computer Networks*, vol. 135, pp. 73–86, 2018.

[38] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, "Empiot: An energy measurement platform for wireless iot devices," *Journal of Network and Computer Applications*, vol. 121, pp. 135–148, November 2018.

[39] G. Rossi and G. Rodola', "psutil - cross-platform system and process utilities module for python," Jan. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.4741562

[40] Z. Li and J. Yuan, "A review on p4-programmable data planes: Architecture, research efforts, and future directions," *Computer Communications*, vol. 173, pp. 221–238, 2021.