Design and Deployment of a Testbed for SmartNIC and Programmable Data Plane Experimentation

Jose Gomez*, Samia Choueiri†, Ali Mazloum†, Sergio Elizalde†,
Amith GSPN†, Ali AlSabeh‡, Elie F. Kfoury†, Jorge Crichigno†
*Katz School of Business, Fort Lewis College, Durango, CO, USA
†Molinaroli College of Engineering and Computing, University of South Carolina, Columbia, SC, USA
‡College of Sciences and Engineering, University of South Carolina Aiken, Aiken, SC, USA
jagomez@fortlewis.edu, {choueiri, amazloum, elizalds}@email.sc.edu, amithgspn@sc.edu,
ali.alsabeh@usca.edu, ekfoury@email.sc.edu, jcrichigno@cec.sc.edu

***Abstract*—This paper presents the design and deployment of a virtualized testbed that facilitates experimentation and instruction in programmable network systems. The platform integrates Smart Network Interface Cards (SmartNICs), Programmable Data Plane (PDP) switches, and the Data Plane Development Kit (DPDK), within a cloud-based orchestration framework to reproduce high-performance, real-world networking scenarios. It supports line-rate processing, enabling real-time applications such as telemetry, encrypted traffic inspection, and malware detection. Through a series of use cases, we demonstrate how the testbed enables advanced experimentation by offloading infrastructure functions to the data plane, achieving low latency, high throughput, and high scalability. The system also provides users with guided labs for learners and is accessible via NETLAB+ for remote use. Future work includes federation with national-scale infrastructures such as FABRIC to broaden access and support multi-institutional collaboration.**

***Index Terms*—Testbed, P4, Programmable Data Plane (PDP), SmartNIC, Data Plane Development Kit (DPDK).**

I. Introduction

Modern network infrastructures face increasing demands in throughput, latency, and programmability, driven by applications such as cloud-native services, real-time analytics, and autonomous systems [1]. As Moore's Law [2] no longer holds and Dennard Scaling [3] has slowed, general-purpose CPUs are no longer sufficient to meet the performance needs of contemporary network workloads. These trends have prompted a shift toward specialized hardware capable of executing infrastructure functions directly in the data plane.

Infrastructure functions refer to network-level operations such as packet classification, forwarding, encryption, telemetry, load balancing, and congestion control. Traditionally performed in software and across multiple layers of the protocol stack, these operations incur processing delays and overload CPUs when operating at high packet-per-second (pps) rates. Offloading these functions to programmable hardware components, such as Programmable Data Planes (PDP) switches and Smart Network Interface Cards (SmartNICs), enables in-network computation with improved performance [4, 5].

SmartNICs integrate compute resources such as onboard CPUs, FPGAs, or ASICs that support deep packet inspection, header rewriting, and flow control directly on the NIC, without involving the host CPU [6]. Similarly, PDPs written in P4 enable customizable match-action pipelines in switches. Together, these technologies support low-latency, high-throughput processing tailored to application-specific requirements. Despite their capabilities, these technologies remain largely inaccessible in academic settings due to their cost, operational complexity, and support requirements [7]. As a result, many institutions are unable to provide students and researchers with hands-on experience in such emerging architectures.

To address this gap, we present a remotely accessible testbed that combines SmartNICs, P4-programmable switches, and DPDK-enabled targets within a hypervisor-based orchestration system. The platform supports advanced experimentation and instruction by enabling the deployment of infrastructure functions directly in the data plane. It also provides modular training labs and guided exercises suitable for a range of user proficiency levels. The orchestration system is built on NETLAB+ [8], which handles user reservations, dynamic resource allocation, and remote access through a web-based interface; no tools and software (e.g., SSH) are needed on the learner's machine. The only requirement is having Internet access. This approach supports scalable multi-user access and makes complex experiments feasible without requiring physical proximity to specialized hardware.

In this paper, we present a set of use cases enabled by the implemented testbed that demonstrate real-world applications in heavy hitter detection, telemetry, traffic classification, encrypted traffic inspection, and malware detection. These use cases highlight how the platform supports high-performance experiments in programmable networking by offloading infrastructure functions to the data plane. They also illustrate the system's ability to support research and instructional scenarios across various networking topics. The rest of the paper is organized as follows: Section II provides background on SmartNICs and PDPs. Section III describes the system architecture. Section IV presents supported experiments through use cases. Section V concludes the paper and describes future work.

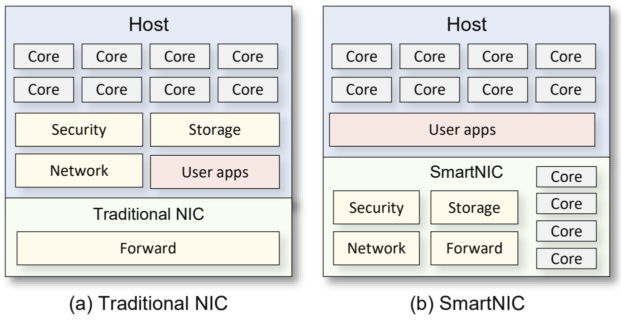

Fig. 1. Comparison between Traditional NICs and SmartNICs. (a) In traditional NIC architectures, infrastructure workloads (e.g., security, storage, network) are executed on the host CPU, increasing system load and latency. (b) SmartNICs integrate programmable compute resources that handle infrastructure functions directly on the NIC, enabling higher performance, reduced host CPU usage, and enhanced support for in-network computing [1].

II. Background

A. Smart Network Interface Cards (SmartNICs)

SmartNICs are advanced network interface devices that extend traditional NIC functionality by integrating programmable processing units. Unlike traditional NICs, which are limited to basic packet transmission and reception, SmartNICs are equipped with specialized hardware such as multi-core CPUs, FPGAs, or ASICs that enable them to offload and accelerate a wide range of infrastructure functions, including packet filtering, encryption, traffic shaping, telemetry, and even portions of application logic. This architectural shift is illustrated in Fig. 1, which compares traditional NICs, where infrastructure functions are executed on the host CPU, with SmartNICs that offload these tasks to onboard processors, thereby enabling lower latency, reduced host load, and enhanced in-network processing capabilities.

This offloading paradigm addresses a growing challenge in modern computing systems, which is the increasing data throughput and complexity of network traffic. With the decline of Moore's Law [2] and the slowdown of Dennard Scaling [3], general-purpose CPUs struggle to keep up with the volume and diversity of network workloads. SmartNICs mitigate this issue by performing compute-intensive or latency-sensitive operations directly on the NIC, thereby reducing the need to traverse the entire operating system stack and minimizing CPU overhead. SmartNICs typically fall into three architectural categories [1]: SoC-based (System-on-Chip), FPGA-based, and ASIC-based. Each category has its advantages and limitations. For instance, an ASIC-based SmartNIC achieves line rate processing, but is limited in programmability. SoC-based SmartNICs are more programmable, but do not reach the line rate when a lot of processing is offloaded to the SmartNIC. These architectures support frameworks such as the Data Plane Development Kit (DPDK) and P4, allowing developers to build custom packet-processing applications.

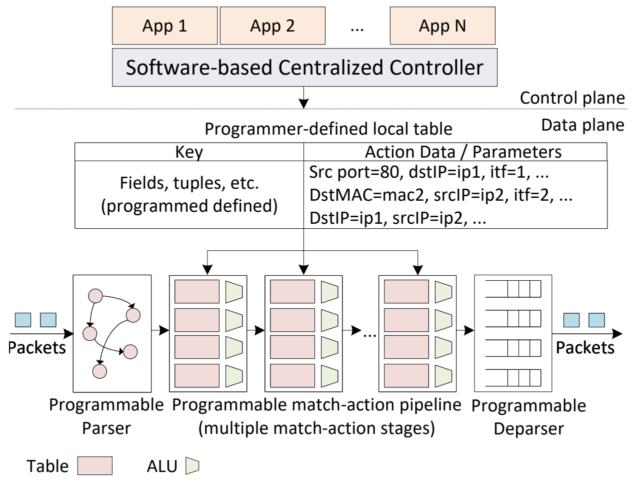
App 1
App 2
...
App N
Software-based Centralized Controller
Control plane
Data plane
Programmer-defined local table
Key
Action Data / Parameters
Fields, tuples, etc. (programmed defined)
Src port=80, dstIP=ip1, itf=1, ...
DstMAC=mac2, srcIP=ip2, itf=2, ...
DstIP=ip1, srcIP=ip2, ...
Packets
Packets
Programmable Parser
Programmable match-action pipeline (multiple match-action stages)
Programmable Deparser
Table
ALU

Fig. 2. Overview of the PISA (Protocol-Independent Switch Architecture) pipeline used in P4-programmable data planes. The architecture includes a programmable parser, a sequence of match-action tables, and a deparser, all controlled by a software-based control plane. Each match-action stage can be programmed to define custom header parsing, table lookups, and packet modifications, enabling high-speed data plane operations [9].

B. Programmable Data Planes (PDPs)

Traditional network devices are built with fixed-function data planes, where packet processing behavior such as forwarding, classification, filtering, and telemetry is pre-defined by the hardware vendor and cannot be modified post-deployment. This rigidity limits innovation, adaptability to new protocols, and the ability to respond to evolving network demands such as dynamic traffic engineering, in-network computing, and real-time telemetry. PDPs address this limitation by enabling direct control over how packets are processed as they traverse network devices [10]. In a PDP, the parsing, matching, and modification of packets are no longer hardwired. Instead, they can be defined and updated by the operator or developer, without modifying the underlying hardware. This programmability allows networks to become more agile, responsive, and tailored to specific applications such as traffic classification [9, 11], DDoS mitigation [12], and granular performance monitoring [13, 14].

P4 (Programming Protocol-Independent Packet Processors) has emerged as the most prominent data plane programming language. P4 provides a high-level, platform-agnostic syntax for defining how packets are parsed, matched, and acted upon. It implements abstractions such as customizable headers, parsers, control blocks, and match-action tables, enabling developers to express both standard and custom packet-processing behaviors. Fig. 2 shows the Protocol-Independent Switch Architecture (PISA), which is a programming model for P4. The PISA pipeline consists of a programmable parser, a sequence of match-action tables, and a programmable deparser. Incoming packets are first parsed into defined headers, which are then processed through one or more match-action stages that perform basic arithmetic operations, header modification, or telemetry collection. Finally, packets are reassem-

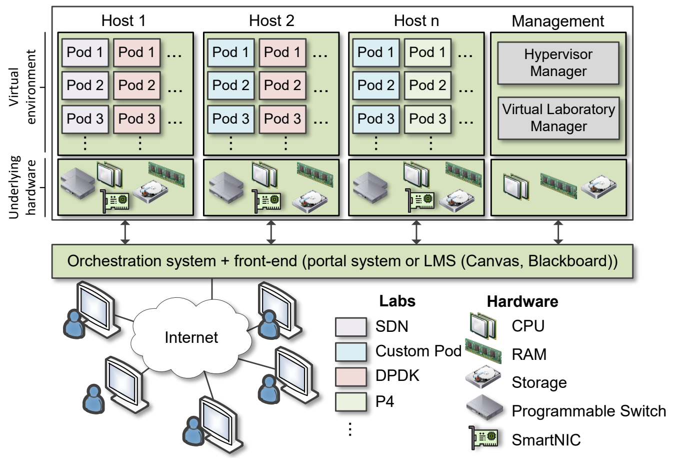

Fig. 3. System architecture. The platform leverages a hypervisor-based orchestration layer to dynamically manage hardware resources—including CPUs, RAM, storage, SmartNICs, and programmable switches—across multiple host machines. Each host runs isolated laboratory pods that support experiments in SDN, P4, custom pods, DPDK, and others. A centralized orchestration system interfaces with a portal or LMS to schedule, provision, and manage virtual labs in a scalable and user-accessible manner.

bled and forwarded via the deparser. The pipeline behavior is defined entirely in P4, which allows users to express how packets are handled at each stage based on arbitrary header fields.

The use of P4 enables infrastructure functions, such as in-band telemetry, custom tunneling protocols, traffic filtering, and load balancing, to be executed directly in the data plane.

III. System Architecture and Implmentation

The implemented system operates as a private cloud, dynamically allocating and managing computing resources to facilitate scalable and interactive virtual laboratories. These resources are distributed across multiple data center hosts to optimize performance and enhance the user experience during hands-on sessions. Fig. 3 shows how the resources are organized in each datacenter host. Each data center host is equipped with robust computing resources, including servers featuring 32 to 40 CPU cores, approximately 1TB of RAM, and roughly 2.5TB of storage capacity. Typically, each data center consists of 4 to 10 servers, dedicated to hosting virtual laboratory pods and the management infrastructure.

The system is implemented on bare-metal hypervisors. Each virtual laboratory pod comprises virtual machines (VMs), virtual switches, legacy routers, PDPs, and SmartNICs. The PDPs are Intel Tofino switches, while the SmartNICs used in the testbed are NVIDIA's BlueField-2 and BlueField-3, and Intel's Infrastructure Processing Unit (IPU) E2100. These elements enable complex, realistic networking scenarios that often require multiple interconnected virtualized components. The system relies on a hypervisor-based orchestration framework utilizing the NETLAB+ appliance. NETLAB+ automates the provisioning and management of virtual lab environments. It handles user reservations, dynamically allocates resources,

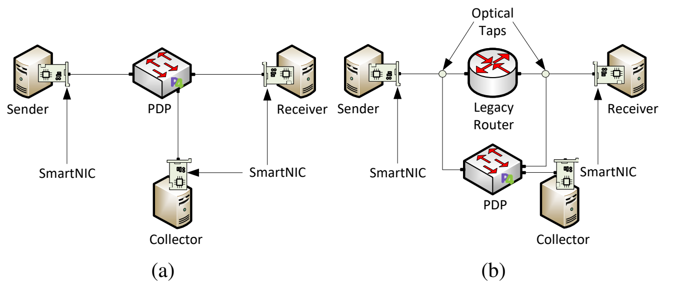

Fig. 4. Generic experimental topology. It includes a traffic sender, a legacy router, and both (a) inline and (b) passive (non-inline) PDPs. Optical taps mirror traffic to a SmartNIC-enabled collector for out-of-band processing. A receiver node completes the end-to-end flow. The sender and the receiver are SmartNIC-enabled hosts.

and schedules lab sessions to ensure seamless accessibility for educational and research purposes.

Users interact with the virtual laboratories through a secure, intuitive web-based interface. Secure HTTPS connections enable access without additional software. These connections traverse the organization's firewall and are managed by an on-premises proxy server, which facilitates Virtual Network Computing (VNC) access to dynamically provisioned lab pods. A reservation and scheduling workflow is implemented via NETLAB+: users log in, select time slots, and gain access to fully provisioned lab environments at the scheduled time. The system supports scheduled and on-demand sessions, with resource allocation optimized across distributed hosts.

IV. Use Cases

To demonstrate the type of experiments supported in the implemented testbed, we present a series of use cases that highlight how the platform supports advanced experimentation in PDPs and SmartNICs. These examples reflect real-world scenarios and research directions recently explored within our academic environment, including security enforcement, traffic monitoring, and telemetry. Each use case leverages the testbed's ability to deploy programmable logic at line rate, enabling high-performance packet processing and low-latency decision-making.

Fig. 4(a) illustrates one of the topologies used across multiple experiments in the testbed. It includes a sender, a receiver, and an inline PDP. Fig. 4(b) shows another topology used for experimenters. This topology implements a non-programmable router (e.g., Juniper MX-204), a PDP, and passive taps. The PDP operates as a measurement instrument, receiving mirrored traffic through optical taps, which supports passive monitoring and deferred analysis. A SmartNIC-based collector is used to aggregate telemetry or execute filtering rules offloaded from the host. The sender and the receiver are SmartNIC-enabled hosts in both topologies. In experiments where multiple senders and receivers are required, the user can start a software switch (e.g., Open vSwitch, Linux Bridge) that connects virtual interfaces from the senders, and the interfaces of the SmartNIC.

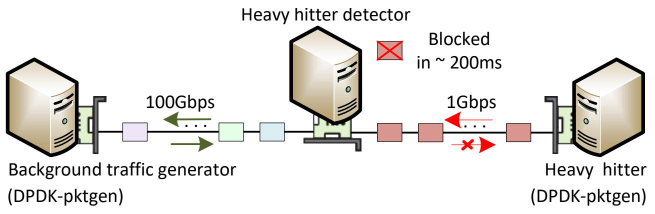
Heavy hitter detector
Blocked in ~ 200ms
100Gbps
1Gbps
Background traffic generator (DPDK-pktgen)
Heavy hitter (DPDK-pktgen)

Fig. 5. Experimental topology for in-band telemetry and heavy-hitter detection. Background traffic and heavy flows are generated using DPDK-pktgen. A SmartNIC-based heavy hitter detector monitors incoming traffic at 100 Gbps, identifies large flows using sketch-based algorithms, and drops or marks them before they saturate the downstream 1 Gbps link [15].

A. Use Case 1: Heavy-Hitter Detection

Network operators require fine-grained visibility into traffic behavior to detect anomalies, enforce policies, and optimize performance. Traditional monitoring tools often rely on external collectors or CPU-based sampling, which introduce latency and lack real-time responsiveness. In this use case [15], the testbed is used to implement a heavy-hitter detection solution using P4-DPDK. P4-DPDK is a technology that bridges the P4 language with the performance benefits of DPDK. DPDK runs on multi-core CPUs on the host, and bypasses the kernel stack, enabling lower latency and high throughput.

Fig. 5 shows the experimental topology. The experiment deploys a match-action pipeline where the Count-min Sketch (CMS) algorithm is implemented. The CMS allows estimating the counts with high scale. Heavy hitters are then identified based on the frequency and volume thresholds. The CMS uses multiple hash functions. A hash function takes as input the 5-tuple, which is used to identify a flow. The heavy hitter is detected by comparing the number of packets received from the flow recorded in the CMS (calculated by taking the minimum across the counts) against a predefined threshold.

Fig. 6 shows the loss rate under varying packet sizes (64 to 1500 bytes) and line rates from 1 Gbps to 100 Gbps where the proposed system is compared with Suricata DPDK, an open-source Network Security Monitor (NSM). The results demonstrate a performance advantage for the proposed system.

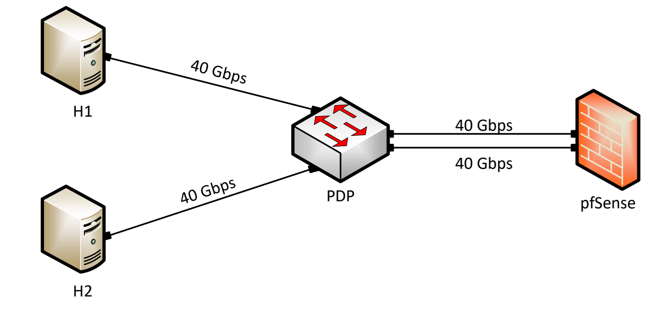

Fig. 7. Experimental topology for DNS deep packet inspection and domain-based filtering. DNS traffic is generated between two hosts and analyzed by both a P4-programmable switch running P4DDPI and a pfSense firewall to compare in-network and CPU-based filtering approaches. [16].

Suricata-DPDK experiences drastic packet loss, particularly with smaller packet sizes and higher line rates. On the other hand, the proposed system maintains negligible loss in nearly all configurations, especially when larger packets are used.

B. Use Case 2: Deep Packet Inspection and Domain-Based Filtering in P4 Switches

Traditional firewall systems rely heavily on CPU-based packet inspection, which becomes a bottleneck in high-throughput environments. In this use case, the testbed is used to deploy a P4-based Deep Packet Inspection (P4DDPI) system designed to extract and filter domain names directly in the data plane of programmable switches. The goal is to block malicious domains at line rate without involving control-plane logic or external firewalls.

Fig. 7 shows the experimental topology implemented in the testbed, where DNS traffic is routed through a programmable switch running the P4DDPI pipeline. The topology follows the generic topology shown in Fig. 4(a). Instead of a collector, a pfSense firewall is installed. The firewall is used for comparison purposes against the PDP. Hosts H1 and H2 generate DNS and background traffic at 40 Gbps across the switch and the firewall. The P4 switch parses domain labels using packet

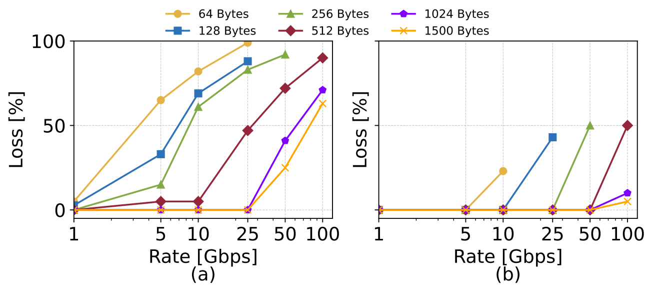

Fig. 6. Comparison of packet loss rates between (a) Suricata-DPDK and (b) the proposed P4-DPDK system under varying packet sizes and traffic rates. The proposed system consistently outperforms Suricata, achieving minimal packet loss even at 100 Gbps with larger packets, while Suricata suffers significant loss across most configurations [15].

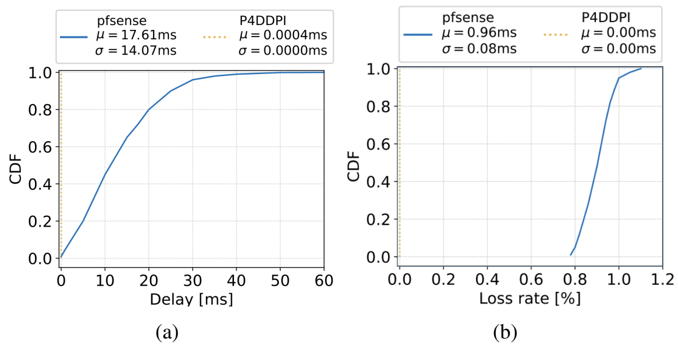

Fig. 8. Experimental results. (a) CDF of DNS query delay under P4DDPI and pfSense. P4DDPI achieves sub-microsecond delay due to data plane processing, while pfSense shows significant latency due to CPU load and software-based inspection. (b) CDF of packet loss under increasing DNS traffic rates. P4DDPI maintains 0% packet loss across all scenarios, while pfSense shows loss rates approaching 1% as DNS traffic increases [16].

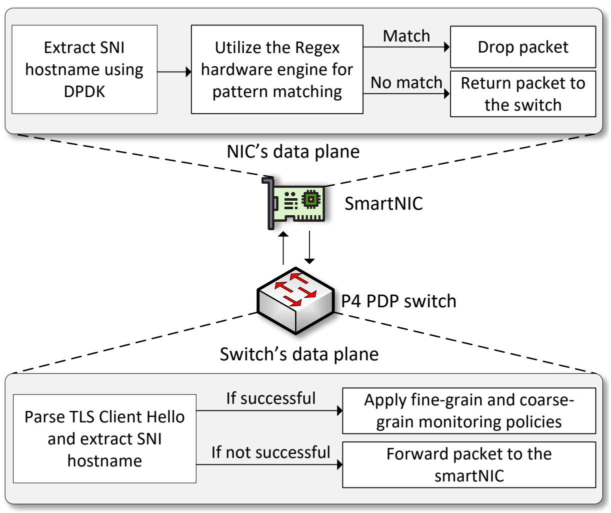
Extract SNI hostname using DPDK
Utilize the Regex hardware engine for pattern matching
Match
Drop packet
No match
Return packet to the switch
NIC's data plane
SmartNIC
P4 PDP switch
Switch's data plane
Parse TLS Client Hello and extract SNI hostname
If successful
Apply fine-grain and coarse-grain monitoring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

Fig. 9. Experimental topology for TLS SNI inspection using hybrid P4-DPDK pipeline. The P4 switch parses TLS Client Hello messages to extract SNI hostnames and apply filtering rules. Packets not fully parsable are sent to a SmartNIC running a DPDK application, which uses a Regex engine to enforce additional security policies [17].

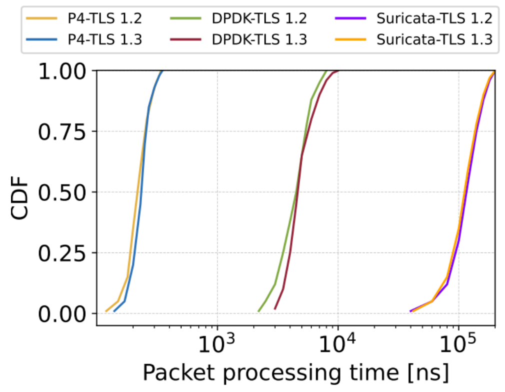

Fig. 10. CDF of TLS packet processing times for TLS 1.2 and TLS 1.3 using P4 switch, DPDK SmartNIC, and Suricata. The hybrid P4-DPDK system significantly outperforms Suricata, processing most packets in under 1 μs (P4) or 7 μs (SmartNIC), while Suricata averages over 90 μs [17].

recirculation, computes label hashes, and applies filtering rules stored in match-action tables to drop or allow traffic based on the resolved IP address of the queried domain.

The experiment compares the proposed system with pfSense in terms of delay and packet loss. Fig. 8(a) presents the cumulative distribution function (CDF) of DNS query delays. The results show that the proposed system maintains sub-microsecond latency, while pfSense introduces an average delay of over 17 ms, with a significant number of queries experiencing delays above 20 ms. Fig. 8(b) shows the CDF of packet loss, where pfSense experiences up to 1% loss under heavy DNS loads, while P4DDPI incurs no observable packet drops.

C. *Use Case 3: TLS SNI Inspection Using Hybrid P4 and DPDK Pipelines*

As HTTPS adoption approaches near ubiquity, monitoring encrypted traffic has become both critical and challenging. A widely used approach for classifying HTTPS traffic involves extracting the Server Name Indication (SNI) field from the TLS handshake. Traditionally, SNI-based inspection relies on general-purpose CPUs and kernel-level DPI, which introduces latency and scalability limitations under high traffic loads.

This use case presents a hybrid architecture that performs TLS SNI extraction and filtering directly in the data plane using a P4-programmable switch and a SmartNIC-accelerated DPDK application. Fig. 9 illustrates the system architecture. TLS Client Hello packets are first parsed by the P4 switch, which extracts SNI hostnames and enforces fine and coarse-grain security policies. Packets that cannot be parsed due to variable-length headers are forwarded to a DPDK application running on a SmartNIC. This application uses the SmartNIC's built-in Regex engine to perform high-speed pattern matching, allowing complex policy enforcement with minimal delay.

To evaluate performance, the system is compared against Suricata, a CPU-bound intrusion detection system. Fig. 10 presents the CDF of TLS packet processing times for TLS 1.2 and TLS 1.3. The proposed system achieves near-line-rate performance: the P4 switch processes packets in ~700–900 ns, and the SmartNIC-DPDK application handles complex packets in ~7 μs. In contrast, Suricata averages ~92–95 μs per packet, demonstrating 1–2 orders of magnitude improvement.

D. *Use Case 4: Real-Time Flow Monitoring and Malware Detection with P4 and RDMA*

Accurate and real-time monitoring of network flows is a fundamental requirement for applications such as anomaly detection, traffic engineering, and malware identification. Traditional monitoring tools such as NetFlow, sFlow, and Zeek struggle with packet loss and coarse visibility under high data rates. This use case demonstrates how the testbed enables scalable and lossless per-flow statistics collection using a P4-programmable switch and a SmartNIC-enabled collector via Remote Direct Memory Access (RDMA).

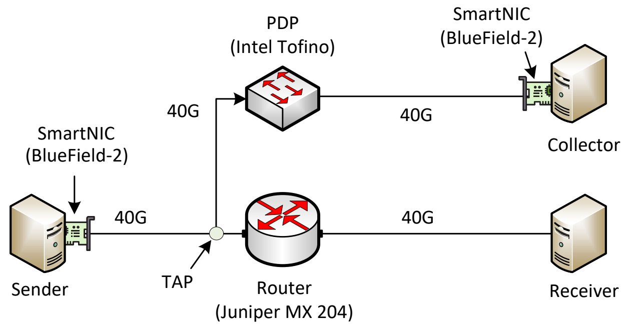

Fig. 11. Experimental topology for real-time flow monitoring. Traffic is mirrored from a router to an Intel Tofino P4 switch, which processes the packets and sends flow-level metrics to a SmartNIC-enabled collector using RDMA over 40 Gbps links [13].

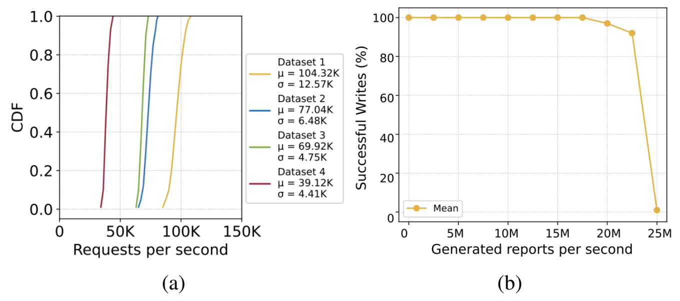

Fig. 12. Experimental results. (a) CDF of generated reports per second across four MAWI traces. The system handled between 39K and 104K reports/sec in real-world traffic scenarios. (b) System scalability: percentage of successful RDMA memory writes as a function of report rate. The system maintains near 100% success up to ∼22 million reports/sec, ensuring lossless high-speed telemetry [13].

As shown in Fig. 11, the system consists of a traffic sender connected to a router and a P4-enabled Intel Tofino switch. The topology follows the generic one shown in Fig. 4(b). The P4 switch processes mirrored traffic and calculates metrics such as round-trip time, inter-arrival time, packet count, and flow size. Instead of using traditional host-based transmission, the P4 switch sends flow reports using RDMA over Converged Ethernet version 2 (RoCEv2), which is a protocol that allows direct writes to the collector's memory without CPU involvement. The collector uses parallel worker threads to update flow metrics in real time.

Fig. 12(a) shows the number of reports per second that are sent when real backbone traces are replayed. Depending on the dataset, the number varies between 39K and 104K reports/sec. Fig. 12(b) demonstrates the system's scalability, achieving over 20 million reports/sec with over 99% successful memory writes, far exceeding the rates required for backbone monitoring. These results confirm the system's ability to support high-throughput malware detection and dynamic flow monitoring. Unlike network security monitors such as Zeek, which experience packet loss as the traffic rates increase, this architecture offers lossless monitoring, real-time processing, and accurate metric collection at scale.

V. Conclusion and Future Work

This paper presented the design and implementation of a testbed for experimentation and instruction in programmable networking. The platform integrates SmartNICs, P4-programmable switches, and DPDK within a virtualized orchestration system, enabling realistic, scalable experiments in areas such as telemetry, traffic classification, and encrypted traffic inspection. Results across multiple use cases confirm the effectiveness of offloading infrastructure functions to the data plane, reducing latency, lowering CPU load, and improving throughput. In addition to supporting research, the testbed provides guided training through NETLAB+, offering remote access and lab scheduling for P4 and SmartNIC experimentation. This makes advanced network experimentation accessible to institutions with limited physical resources. Future work includes integrating the testbed with national-scale platforms such as FABRIC [18] to enable broader access, multi-site collaboration, and resource federation for large-scale programmable networking research.

Acknowledgment

The authors acknowledge the support of the National Science Foundation (NSF) under award numbers 2346726 and 2403360. They also thank the FABRIC [18] team for granting access to the platform used in the preliminary experiments.

References

[1] E. Kfoury, S. Choueiri, A. Mazloum, A. AlSabeh, J. Gomez, and J. Crichigno, "A comprehensive survey on SmartNICs: Architectures, development models, applications, and research directions," *IEEE Access*, 2024.

[2] G. Moore, "Progress in digital integrated electronics," in *Electron devices meeting*, vol. 21, 1975.

[3] R. Dennard, F. Gaensslen, H. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of solid-state circuits*, vol. 9, 1974.

[4] Google, "Encryption in transit." [Online]. Available: https://tinyurl.com/yp93tj9u, Accessed on 06-18-2025.

[5] Microsoft, "Azure SmartNIC." [Online]. Available: https://tinyurl.com/2wwzuykz, Accessed on 06-18-2025.

[6] A. AlSabeh, A. Mazloum, E. Kfoury, J. Crichigno, and H. Berry, "Enabling line-rate TLS SNI inspection in P4 programmable data planes," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, 2025.

[7] J. Gomez, E. Kfoury, J. Crichigno, and G. Srivastava, "A survey on network simulators, emulators, and testbeds used for research and education," *Computer Networks*, vol. 237, 2023.

[8] Network Development Group (NDG), "NETLAB+." [Online]. Available: https://tinyurl.com/2yn88e85, Accessed on 06-18-2025.

[9] J. Gomez, E. Kfoury, J. Crichigno, and G. Srivastava, "A survey on TCP enhancements using P4-programmable devices," *Computer Networks*, 2022.

[10] E. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, 2021.

[11] J. Gomez, E. Kfoury, A. Mazloum, and J. Crichigno, "Improving flow fairness in non-programmable networks using P4-programmable data planes," *Computer Networks*, 2025.

[12] A. AlSabeh, J. Khoury, E. Kfoury, J. Crichigno, and E. Bou-Harb, "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment," *Computer Networks*, 2022.

[13] E. Kfoury, A. Mazloum, J. Gomez, A. AlSabeh, and J. Crichigno, "Real-time flow statistics collection using RDMA and P4 programmable data planes," *IEEE International Conference on Communications (ICC)*, 2025.

[14] A. Mazloum, J. Gomez, E. Kfoury, and J. Crichigno, "Enhancing perfSONAR measurement capabilities using P4 programmable data planes," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023.

[15] S. Choueiri, A. Mazloum, E. Kfoury, and J. Crichigno, "Scalable heavy hitter detection: A DPDK-based software approach with P4 integration," *IEEE Global Communications Conference (GLOBECOM)*, 2024.

[16] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4DDPI: Securing P4-programmable data plane networks via DNS deep packet inspection," *Network and Distributed System Security (NDSS)*, 2022.

[17] A. Mazloum, A. AlSabeh, E. Kfoury, and J. Crichigno, "Domain name security inspection at line rate: TLS SNI extraction in the data plane using P4 and DPDK," *IEEE International Conference on Communications (ICC)*, 2025.

[18] I. Baldin, A. Nikolich, J. Griffioen, I. Monga, K. Wang, T. Lehman, and P. Ruth, "FABRIC: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, 2019.