Accelerating PUF-based UAV Authentication Protocols Using Programmable Switch

Ranjitha K
IIT Hyderabad
Email : cs21resch01002@iith.ac.in

Divya Pathak
IIT Hyderabad
Email : cs21mtech12009@iith.ac.in

Tejasvi Alladi
Carleton University, Canada
Email : talladi.carleton@gmail.com

Antony Franklin
IIT Hyderabad
Email : antony.franklin@cse.iith.ac.in

Praveen Tammana
IIT Hyderabad
Email : praveent@cse.iith.ac.in

***Abstract*—Many UAV technology use cases (e.g., traffic management) has ultra-low latency and strong security requirements. But achieving both simultaneously is challenging. In this work, we consider UAV device authentication as a use case and develop a fast and secure UAV device authentication system. Our key idea is to leverage highly secure Physically Unclonable Functions (PUFs) and high-speed programmable packet-processing data planes, and develop a practically deployable PUF-based authentication protocol for UAVs that is (a) robust to various security attacks, and (b) enables UAV authentication at network speed. In this work, we demonstrate the feasibility of our idea by offloading the authentication protocol to a Tofino-based high-speed programmable switch. Our preliminary experiments show that protocol offloading would reduce authentication latency significantly (approx.** 100%**).**

I. Introduction

Unmanned Aerial Vehicle (UAV) technology is being deployed around the world at a rapid pace with use cases such as delivery services, flying base stations, remote sensing, surveillance, and smart city applications. Many of these services require ultra-low latency and secure communication. A category of UAV devices (*e.g.*, small aerial drones) are lightweight, unmanned, and deployed in the open air, therefore (1) they are vulnerable to various attacks [1] that aim to steal sensitive data, such as tampering, spoofing, man-in-the-middle, cloning, etc; and (2) they have constraints on compute and energy resources.

To prevent such attacks, device authentication plays a key role in avoiding unauthorized devices being part of the network. Existing crypto-based authentication techniques provide necessary security, but they are proven to be inefficient in terms of computing requirements and energy consumption. These techniques are compute-intensive, thus require many CPU cycles, and are energy inefficient [2]. Moreover, oftentimes secret information (*e.g.*, private keys) is stored in the device on battery-backed storage. If an adversary gets access to the device, the secret information would get exposed and thus compromise the UAV network. With these limitations, the key question that we like to investigate in this work is: *How to authenticate resource-constrained UAV devices using techniques that are computationally light-weight, fast, and resistant to data tampering and spoofing attacks?*

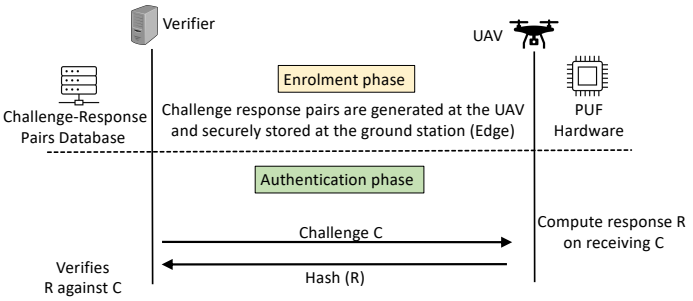


Fig. 1. **PUF-based UAV Authentication Protocol**

One potential approach is to leverage Physical Unclonable Functions (PUFs) [3], [4] for device identification and authentication purposes. PUFs are lightweight and unclonable, and there have been many PUF-based authentication protocols [5]–[11] that are robust to data tampering and spoofing attacks. Fig. 1 summarizes the key idea behind these protocols. Though the existing works mainly focus on the security analysis (*e.g.*, security proofs) of proposed protocols, they do not provide insights on other important performance parameters such as latency and throughput.

More specifically, consider that the authentication functionality (or PUF verifier logic) is running on a general-purpose CPU server at a ground control station. Before a UAV can access resources (*i.e.*, data, compute, network), (a) it sends an authentication request to the control station, (b) then the station issues a challenge to the UAV followed by a response message from the UAV to the station. All these steps add significant delays before the data at UAV is sent to the intended destination. More specifically, the end-to-end authentication request completion time can take up to milliseconds due to (1) multiple RTTs, and (2) packet copies and I/O interrupts at an edge server, especially while moving packets from the network interface to the hypervisor layer to the application layer to PUF verifier. To summarize, PUF-based authentication introduces additional processing delay in addition to the existing propagation and transmission delay, making it challenging

to achieve ultra-low latency requirements (*i.e.*, < 1000 micro secs) of UAV applications.

In this work, we aim to take a step towards building a secure and fast PUF-based device authentication system by leveraging high-speed programmable data planes (*e.g.*, smartNICs [12], Intel Tofino switch [13]). Recent works [14], [15] show performance benefits by offloading network functions from CPUs to programmable data planes. Inspired by them, in this work we propose to offload PUF-based authentication protocol to a high-speed programmable switch. By doing so, our approach significantly reduces the time to authenticate a device, thereby satisfying ultra-low latency requirements of various UAV application use cases.

To demonstrate the feasibility of our idea, we offloaded the PUF verifier logic of an authentication protocol to Wedge100BF Tofino programmable switch [13]. By doing so, we observe close to 100% improvement in authentication request processing time compared to the time taken by a PUF verifier logic running on a general-purpose CPU server.

II. Background

Physical Unclonable Functions (PUFs). PUFs operate on a challenge-response mechanism, that is, given an input stimulus called a challenge, it generates an output called response. To generate a response for a given challenge, PUF hardware relies on the inherent randomness involved in the manufacturing process of physical structure. Since the variation in the physical structure cannot be replicated, two different PUFs manufactured with the same circuit and fabrication process would give different outputs for the same input, thus *unclonable*. To summarize, similar to the device fingerprint, PUFs can be used as a unique identifier and prevent the adversary from spoofing legitimate devices. Such security benefits and lightweight functioning make PUFs a promising choice for device authentication. Other applications of PUFs are cryptographic key generation [16] and construction of leakage-resilient block ciphers [17].

PUF-based authentication protocol. Using PUFs, one can develop an authentication protocol that runs between UAV and ground control station (GS) where GS acts as a verifier. As shown in Fig. 1, the protocol has two phases: the enrollment phase and the authentication phase. Before enrolling a UAV, challenge-response pairs (CRPs) are generated using PUF hardware on the UAV and the generated CRPs are securely stored in the verifier database. During the authentication phase, the verifier issues a challenge (picked randomly from the stored CRPs) to the UAV, followed by a response generated by the UAV's PUF. The authentication is successful only if the UAV's response matches with the response at the verifier.

Accelerate authentication using P4 and Programmable switch. To keep up with high line rates (*e.g.*, order of Tbps), programmable switches (*e.g.*, Intel Tofino switch [13]) support only a few dozens of arithmetic and comparison operations. Since the PUF verifier logic is simple, it is amenable to implementation on a programmable switch. To do so, we

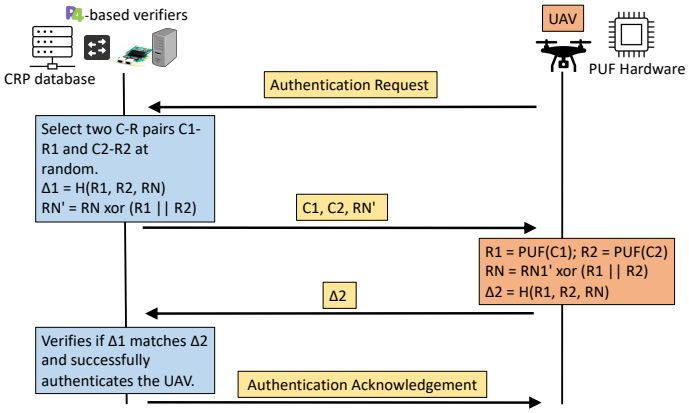


Fig. 2. **PUF-based authentication protocol workflow**

MAT 1

srcMAC	idx 1	action
X	1	copy(CX_1,RX_1)
X	2	copy(CX_2,RX_2)
⋮	⋮	
X	n/2	copy($CX_{n/2}$,$RX_{n/2}$)
Y	1	copy(CY_1,RY_1)
⋮	⋮	
Y	n/2	copy($CY_{n/2}$,$RY_{n/2}$)
⋮	⋮	
M	n/2	copy($CM_{n/2}$,$RM_{n/2}$)

MAT 2

srcMAC	idx 2	action
X	1	copy(CX_1,RX_1)
X	2	copy(CX_2,RX_2)
⋮	⋮	
X	n/2	copy($CX_{n/2}$,$RX_{n/2}$)
Y	1	copy(CY_1,RY_1)
⋮	⋮	
Y	n/2	copy($CY_{n/2}$,$RY_{n/2}$)
⋮	⋮	
M	n/2	copy($CM_{n/2}$,$RM_{n/2}$)

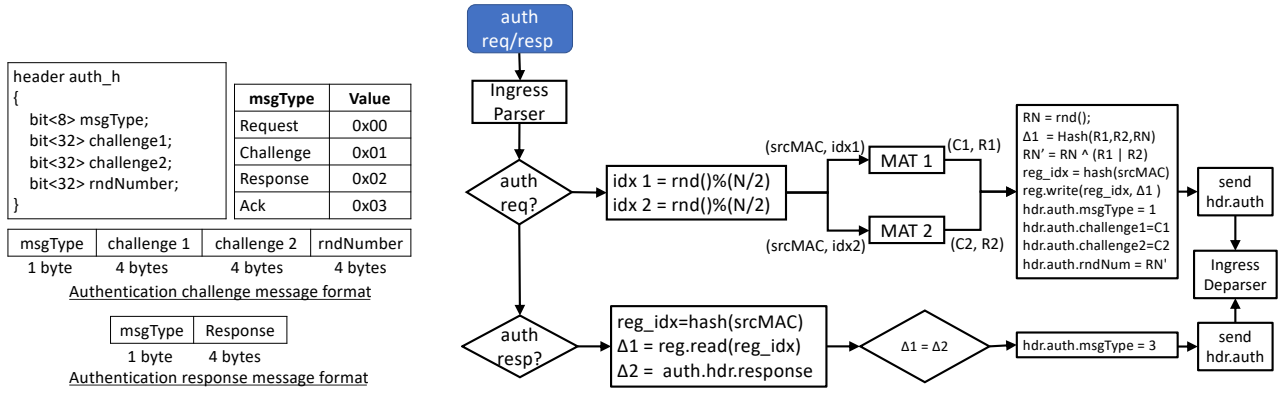
Fig. 3. **Match-Action Tables**

can write programs using P4 [18], a domain-specific network programming language, and specify which packet headers to be recognized in the switch data plane and how the recognized headers should be processed. In short, using P4 and programmable switches, one can implement novel protocols (*e.g.*, PUF verifier logic) entirely in the switch data plane and process authentication requests much faster than general-purpose CPUs.

Programmable switch pipeline. Programmable switches (*e.g.*, Intel's Tofino [13]) follow an abstract switch model [19] called protocol-independent switch architecture (PISA). PISA contains mainly four components: parser, match-action tables, registers, and computation primitives. A parser is a finite state machine using which a P4 programmer can declare which packet headers to be recognized in the switch pipeline. The pipeline also contains a chain of match-action tables where each table can be programmed with a set of rules; each rule matches on a header field and applies associated action (*e.g.*, rewrite header, drop). Registers can be programmed to maintain state (*e.g.*, counters) across packets and the state can be read/written by packets at line rate. Finally, computation primitives can be used to perform arithmetic and logical operations (*e.g.*, additions, bit-shifts, hashing) on packet header fields and per-packet temporary buffer called metadata.

III. Offloading PUF verifier to switch

In this work, we propose to offload PUF verifier logic from the CPU to a programmable switch. By doing so, we can significantly reduce the time to authenticate a UAV device (latency), subsequently improving the authenticate rate



(a) **Authentication protocol message formats** (b) **Authentication request and response flow at the switch**

Fig. 4. **PUF verifier packet-processing in programmable switch pipeline**

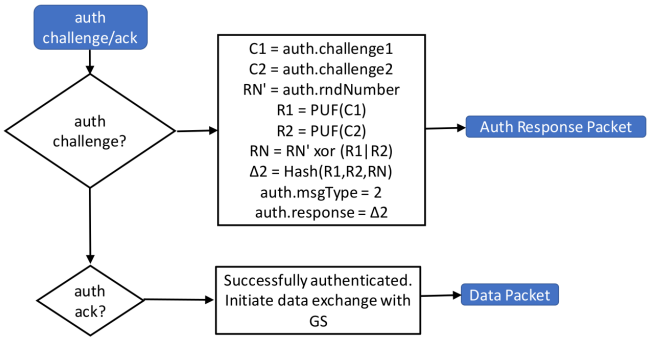


Fig. 5. **Challenge message and Acknowledgement message flow at UAV**

(throughput). In the literature, there are many versions of PUF-based authentication protocol to make the protocol robust to security attacks. Since our focus is to understand the benefits of offloading the PUF verifier, we choose one simple protocol [7] and demonstrate its implementation feasibility on the switch using P4 language. Fig. 2 shows the workflow of authentication protocol that we offload to programmable switch. [7] has more details on security proof of this protocol.

Enrollment Phase. Before deploying a UAV, Challenge-Response Pairs (CRPs) are generated using UAV's PUF hardware. The CRPs of all UAVs managed by a ground control station or an edge cloud is stored in a database. In our approach, the CRPs in the database is programmed as match-action table (MAT) entries, so that during the authentication phase they are accessible in the switch pipeline. More specifically, suppose there are n CRPs per UAV and a total of M UAVs. As shown in Fig. 3, two MATs are programmed with entries such that each entry matches on two header fields: UAV MAC address (srcMAC) to uniquely identify a UAV and an index (idx) range from 1 to $N/2$ (*i.e.*, one table entry for each CRP). We divide per-UAV CRPs (*e.g.*, 1 to N for UAV X) equally across multiple tables; the number of tables is the same as the number of challenges to be sent to the UAV (more details to follow). Note that one can add table entries using control plane software like P4Runtime [20].

Authentication Phase. As shown in Fig. 4(a), we define a protocol header ($auth\_h$) to support four types of protocol messages: Request, Challenge, Response, and Acknowledgement. An authentication message type is identified using *msgType* field. On each packet arrival, the ingress parser extracts the authentication header and then checks whether the packet is a request message or a response message. Fig. 4(b) shows the flow of these two messages in the switch pipeline.

Challenge by the switch. For each request message, the switch identifies UAV's CRPs using MAC address and sends two randomly picked challenges to the UAV. To do so, we generate two random numbers in the range of 0 to $N/2$ and store them in $idx1$ and $idx2$, respectively. The UAV's MAC address and indexes are used as match fields in MATs and copy associated CR pairs (*i.e.*, ($C1$, $R1$) and ($C2$, $R2$)) to packet metadata fields. Next, a hash value ($\triangle 1$) and a random number ($RN^{'}$) is computed from three fields: $R1$, $R2$, and a random number (RN). To remember that a challenge is issued for a particular UAV, $\triangle 1$ is stored in a stateful register indexed by a hash value (reg_idx) obtained by hashing UAV's MAC address (we can also use 5-tuple packet header fields). After that, a challenge packet is prepared with the randomly picked challenges and sent to the UAV.

UAV response. Fig. 5 shows the packet flow for challenge and acknowledgment messages. UAV reads challenges ($C1$, $C2$) in the challenge message sent by the switch, and asks PUF hardware to generate a response for each of these challenges. From the responses ($R1$, $R2$) and $RN^{'}$, a hash value ($\triangle 2$) is generated and a response message with the computed hash value is sent to the switch.

Validate response at the switch. Upon receiving the response, the switch first retrieves the hash value ($\triangle 1$) stored previously in the stateful register and compares $\triangle 1$ with the hash value in the response packet ($\triangle 2$). If both are the same, then the switch will send an acknowledgment message to the UAV to indicate that the UAV is successfully authenticated.

Mapping verifier operations to P4 constructs. To summarize, Fig. 6 shows the mapping of operations in PUF verifier logic to P4 language constructs. Hash functions that are part of the authentication are supported as externs in P4. It is noteworthy that different P4 data planes support

Verifier Construct	P4 Construct
Challenge-response database	P4 match-action table
+, -, &&, ||, xor, >>, <<, >, <, ==, !=	P4 ALUs
If-else condition	P4 if-else
Hash functions	1. SHA variants are available on smartNICs and DPDK 2. CRC variants available on Tofino switch
Psuedo random number generator	Random extern

Fig. 6. **Mapping verifier constructs to P4 language constructs**

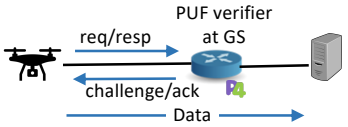


Fig. 7. **Switch verifier setup**

different hash functions offering varying levels of security. For example, Intel's Tofino switch does not support secure hash functions like SHA, instead, it supports insecure CRC hash. Whereas other programmable data planes such as Netronome SmartNICs [12] and Nvidia BlueField [21] do support secure hashing like SHA, but they are SoC-based NICs thus packet-processing latency is higher than programmable switch. A recent work implemented HMAC like secure hash function called sipHash [22] on the Tofino switch. To offer strong security guarantees, in our future work, we plan to integrate PUF verifier with sipHash running on Tofino. To summarize, depending on the underlying data plane used for implementation, our protocol use hash functions that are available on the respective data plane.

IV. Preliminary Evaluation

Experimental setup. To demonstrate the feasibility of our idea, we implement the PUF verifier logic on the Wedge100BF Tofino switch and find the time taken to authenticate a UAV. The testbed comprises one Wedge100BF Tofino switch and one server. Tofino switch has 32 ports each with a maximum speed of 100 Gbps and the server is equipped with 2 Intel Xeon 8-core 3.2 GHz CPU with a dual-port 100 Gbps Netronome smartNICs. Two ports of the switch are connected to the two ports of the server using a QSFP40Gb ethernet copper cable.

Switch verifier. We implement PUF verifier in P4 language and compiled the P4 program using SDE 9.7.0. We deployed the compiled P4 program along with 5 CRPs in the match-action table. The setup is shown in Fig. 7. Next, we simulate UAV requests by opening a UDP socket (written in C++) on the server and send 100 authentication requests one after another to the switch. To find the time taken to finish authentication (includes request, challenge, response, and ack), we collect packet traces and compute the time elapsed between request and corresponding ack.

Server verifier. To understand the offload benefits in terms of latency, we also run the PUF verifier written in C++ on the server and compare the authentication latency with the switch

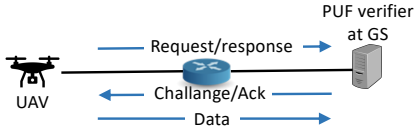


Fig. 8. **Server verifier setup**

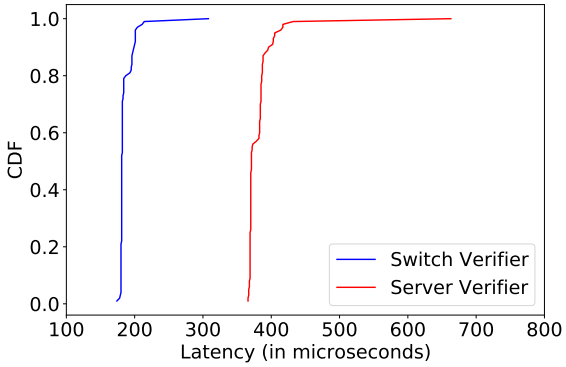


Fig. 9. **CDF of authentication latency**

implementation. For this, on the same server, we create two network namespaces named UAV and verifier, and assigned the namespaces to authentication request generator and CPU-based verifier, respectively. As shown in Fig. 8, requests are sent to the NIC port connected to the Tofino switch, and the switch simply forwards the request to another port on which the CPU verifier is listening for authentication requests. Similar to the switch verifier experiment, we send 100 requests and compute the time taken to finish authentication.

Results. Fig. 9 shows the cumulative distribution function (CDF) of authentication latency of all requests for both implementations. We observe that the authentication latency of the switch verifier is less than 214 microseconds for 99% of the time whereas it is 432 microseconds in the server verifier case. This indicates offloading the PUF verifier to switch pipeline will improve authentication latency by 2 times which is close to 100% improvement.

V. Conclusion and Future Work

In this work, we implement a PUF-based authentication protocol on programmable switch. Our preliminary evaluation shows that by offloading the PUF verifier to switch, we can improve authentication latency significantly, thus enables to meet the ultra-low latency. In our future work, we plan to (1) address challenges that arise while scaling our approach to tens of thousands of UAVs, especially in terms of authentication rate, the maximum number of concurrent requests, and resource overhead; (2) explore secure hashing techniques that can run on Tofino switch; (3) extend PUF-based authentication with key exchange mechanism; (4) design a device-to-device PUF-based authentication protocol where programmable switch acts as a gateway; and (5) offload PUF verifier to other software and hardware programmable data plane targets.

REFERENCES

[1] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," *Internet of Things*, vol. 11, p. 100218, 2020.

[2] M. O. Ozmen and A. A. Yavuz, "Dronecrypt-an efficient cryptographic framework for small aerial drones," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 1–6.

[3] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[4] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A puf taxonomy," *Applied Physics Reviews*, vol. 6, no. 1, p. 011303, 2019.

[5] T. Alladi, G. Bansal, V. Chamola, M. Guizani *et al.*, "Secauthuav: A novel authentication scheme for uav-ground station and uav-uav communication," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15 068–15 077, 2020.

[6] V. Pal, B. S. Acharya, S. Shrivastav, S. Saha, A. Joglekar, and B. Amrutur, "Puf based secure framework for hardware and software security of drones," in *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2020, pp. 01–06.

[7] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A puf-based secure communication protocol for iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, pp. 1–25, 2017.

[8] A. Braeken, "Puf based authentication protocol for iot. symmetry 10, 352 (2018)."

[9] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building puf based authentication and key exchange protocol for iot without explicit crps in verifier database," *IEEE transactions on dependable and secure computing*, vol. 16, no. 3, pp. 424–437, 2018.

[10] Y. Yilmaz, S. R. Gunn, and B. Halak, "Lightweight puf-based authentication protocol for iot devices," in *2018 IEEE 3rd international verification and security workshop (IVSW)*. IEEE, 2018, pp. 38–43.

[11] M. Barbareschi, A. De Benedictis, E. La Montagna, A. Mazzeo, and N. Mazzocca, "A puf-based mutual authentication scheme for cloud-edges iot systems," *Future Generation Computer Systems*, vol. 101, pp. 246–261, 2019.

[12] (2016) Netronome Agilio CX SmartNICs. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[13] (2020) Intel Intelligent Fabric Processors. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[14] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: Enabling State-Intensive Network Functions on Programmable Switches," in *ACM SIGCOMM*, 2020.

[15] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, and H. Hu, "FastFE: Accelerating ML-based Traffic Analysis with Programmable Switches," in *ACM SIGCOMM SPIN workshop*, 2020.

[16] R. Maes, A. Van Herrewege, and I. Verbauwhede, "Pufky: A fully functional puf-based cryptographic key generator," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 302–319.

[17] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory leakage-resilient encryption based on physically unclonable functions," in *Towards Hardware-Intrinsic Security*. Springer, 2010, pp. 135–164.

[18] P4_16 Language Specification. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html

[19] P4_16 Language Specification. [Online]. Available: https://p4.org/p4-spec/docs/PSA-v1.1.0.html

[20] P4Runtime. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[21] NVIDIA BlueField DPUs. [Online]. Available: https://www.nvidia.com/en-in/networking/products/data-processing-unit/

[22] S. Yoo and X. Chen, "Secure keyed hashing on programmable switches," ser. ACM SIGCOMM SPIN '21, 2021.