2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)

Cryptographic Hashing in P4 Data Planes

Dominik Scholz*, Andreas Oeldemann†, Fabien Geyer*, Sebastian Gallenmüller*,
Henning Stubbe*, Thomas Wild†, Andreas Herkersdorf†, Georg Carle*

Technical University of Munich, Germany
*{scholz,fgeyer,gallenmu,stubbe,carle}@net.in.tum.de
†{andreas.oeldemann,thomas.wild,herkersdorf}@tum.de

***Abstract*—P4 introduces a standardized, universal way for data plane programming. Secure and resilient communication typically involves the processing of payload data and specialized cryptographic hash functions. We observe that current P4 targets lack the support for both. Therefore, applications and protocols, which require message authentication codes or hashing structures that are resilient against attacks such as denial-of-service, cannot be implemented.**

To enable authentication and resilience, we make the case for extending P4 targets with cryptographic hash functions. We propose an extension of the P4 Portable Switch Architecture for cryptographic hashes and discuss our prototype implementations for three different P4 target platforms: CPU, NPU, and FPGA. To assess the practical applicability, we conduct a performance evaluation and analyze the resource consumption. Our prototype implementations show that cryptographic hashing can be integrated efficiently. We cannot identify a single hash function delivering satisfying performance on all investigated platforms. Therefore, we recommend a set of hash functions to optimize target-specific performance.

***Index Terms*—Hash function, Data Plane Programming, Performance Evaluation, P4**

I. Introduction

The rise of paradigms like P4 [1] for programming high-speed packet processing platforms has enabled a shift of networking applications to the data plane. Examples of such applications include heavy-hitter detection [2], [3] and in-network caching for distributed services [4]. Looking at implementations of those applications, hash-based data structures like hash tables, bloom filters, or count–min sketches often serve as a basis for efficiently tracking flows. Currently, P4 only supports few algorithms for hash functions, based on cyclic redundancy check (CRC) or checksum calculations commonly used in network protocols (e.g., IP, TCP checksums), which can operate on the header fields of a packet.

To address more secure and advanced applications in the data plane, a wider set of hash functions with cryptographic properties may be beneficial. Two classes of applications can benefit: First, resilience to hash collisions can be improved for hash-based data structures. High susceptibility to hash collisions can create attack vectors, leading to poor resource usage or denial of services [5]. Second, integrity protection, which is typically implemented using hash-based message authentication codes (HMAC) for instance for digital signatures or challenge-response protocols. These are essential for secure communication not only on the Internet but also in industrial networks.

We argue for the benefits of including cryptographic hash functions in P4 platforms. We present our prototype implementations for three different P4 targets: the t4p4s software platform, the Netronome Agilio NFP-4000 Smart NIC, and the NetFPGA SUME. Using measurements we discuss the impacts on performance and resource consumption of cryptographic hash implementations for these devices.

The rest of this paper is organized as follows: First, we review related work in Section II. We argue for the inclusion of cryptographic hash functions in programmable packet processing platforms in Section III. In Section IV we discuss our approaches to extend three different P4 targets with the functionality to calculate cryptographic hashes over packet data. We conduct an evaluation of our prototype implementations focusing on performance metrics as well as resource consumption in Section V. Section VII concludes our work.

II. Related work

Various work already evaluated the suitability of hash algorithms for network packets. Molina et al. [6] and Henke et al. [7] evaluate several different functions, with a focus on packet sampling. Both works highlight that CRC32 is not recommended due to its linear dependency between hash input and hash value, making it vulnerable to bias and security attacks. They recommend BOB [8] as hash algorithm in non-adversarial scenarios due to its performance and avalanche properties. Regarding hardware implementation of non-cryptographic hash algorithms for networking applications, Hua et al. [9] evaluated 18 different functions. They propose a family of hash functions achieving good properties in terms of hashing at a reduced cost regarding hardware footprint and cost per cycle.

Use of hash functions for networking applications implemented in the P4 data plane can be found in various work. Ghasemi et al. [10] investigate performance diagnostic of TCP with *Dapper* using standard 5-tuple hashes. Zaoxing et al. [2] propose *UnivMon* for network flow monitoring based on a sketch data structure where multiple pairwise-independent hash functions are used. Cidon et al. [11] propose *AppSwitch*, a cache for key-value storage using hashes of keys. Sivaraman et al. [3] introduce *HashPipe*, a heavy-hitter detection using a pipeline of hash tables, which retain counters

978-1-7281-4387-3/19/$31.00

for heavy flows while being memory efficient. Finally, Kucera et al. [12] also address heavy-hitter detection using *Elastic Trie*, a novel trie-based data structure. The mentioned works either do not detail the hash algorithm used, or make use of CRC32 as hash function, making them potentially vulnerable to security attacks. Only Ghasemi et al. [10] explicitly describe the strategy used to deal with hash collisions. They use a hash chaining technique combining the hashed value and the TCP sequence numbers.

The IEEE 802.1AE (MACsec) standard provides data confidentiality and integrity through a security tag and a message authentication code (MAC) on the data link layer. These properties are especially interesting for industrial use cases, including automotive [13], [14] and aeronautical applications [15]. Hauser et al. [16] propose P4-MACsec for the automation of MACsec deployment by shifting the MACsec implementation entirely to the data plane of P4 targets. They implement prototypes for the BMv2 model and the NetFPGA, but the solution for the latter was not feasible due to the same problems regarding externs we encountered (see Section IV-C). Hauser et al. [17] propose a similar approach for IPsec, but their prototype implementation for the NetFPGA has the same restrictions. For the ASIC prototype, all cryptographic processing is performed by a CPU-based controller, as the ASIC neither offers cryptographic algorithms nor adding them as externs. This limits performance and functionality [17].

III. Motivation

Hash functions play a key role in various network applications being fundamental for modern network communication. As more and more functionality is being moved to programmable data planes, supporting hash functions with strong cryptographic properties will be a key enabler for various networking use cases.

A. Working with hashes in P4

The Portable Switch Architecture of $P4_{16}$ supports five different functions, which may serve as hash functions: four variants of CRC and the 16 bit one's complement used for IP, TCP, and UDP checksum calculation. While these may serve as a good hash function in networking applications [18], they do not provide cryptographic properties.

In $P4_{16}$, hash algorithms can be accessed via standard function calls of external libraries, so-called externs. For instance, the P4 switch model `v1model.p4` external library offers the generic `hash` function. Its parameters include the hash algorithm to use, as well as a list of parsed header or metadata fields to be used as input. The P4 target platform may support additional hashing algorithms as externs.

While P4 does not directly offer primitives for working with data structures such as hash tables or Bloom filters, P4 primitives can be used in combination with P4 registers to emulate those data structures.

B. Applications with security properties

Various attacks have been proposed on poorly implemented hash-based data structures. For instance, hash tables can degenerate to linked lists with maliciously chosen input, leading to high CPU usage in network security monitors [19]. It is therefore recommended to either use cryptographically-strong random number generators or keyed pseudo-random functions instead of CRC [19], [20].

Due to the use of relatively small messages in packet processing, the choice of a hash function for efficient processing is not straightforward. While the SHA-2 family of hash functions is a strong candidate regarding cryptographic features and security, these functions were not designed with good performance for small inputs. Popular candidates are the SipHash family of hash functions used in various programming languages and software [21], or the BLAKE2 family [22], both designed for good performance for small inputs.

Cryptographic hash algorithms are found in various network protocols with different uses [23]. Extending P4 and its hardware platforms with cryptographic algorithms enables offloading of secure applications to the data plane. A use case of interest are MACs, where packet content is checked for data integrity and authenticity. Hash-based MACs (HMACs) are often used for this and can be found in various protocols such as IPsec, TLS or IEEE 802.1AE (MACsec).

Digital signatures or challenge-response protocols, using token or cookie mechanisms, are used to either prove the possession of an authentication token or to encode state that is being exchanged. One such example are TCP SYN cookies [24], which are calculated for each incoming TCP SYN packet during an ongoing attack and, therefore, have to be efficiently generated and verified.

IV. Hashing Extern Implementation

We have extended three different P4 target platforms with externs calculating cryptographic hashes. Each platform has its own way how P4 externs can be added.

A. CPU: t4p4s

t4p4s [25] is a P4 compiler, which generates platform-independent C code. Target-specific code can be linked with additional libraries, in our case the Dataplane Development Kit (DPDK version 17.08). This allows the P4 program to be executed in user space on a CPU-based software system. We use the name t4p4s synonymously for both the $P4_{14}$ compiler and the DPDK-based P4 target.

As t4p4s only supports the TCP/IP checksum calculation as hash algorithm, we extended it with an SSE4.2-accelerated non-cryptographic *CRC32* function. Furthermore, we added the following open-source implementations of (pseudo-) cryptographic hash functions as P4 externs: the original version of *SipHash-2-4* [21]; *Poly1305-AES* [26] based on [27]; the original verion of *BLAKE2b* [22]; and *HMAC-SHA256* and *HMAC-SHA512* based on OpenSSl (v1.1.0). The output length in bit of each of the hash functions is listed in Table I. We

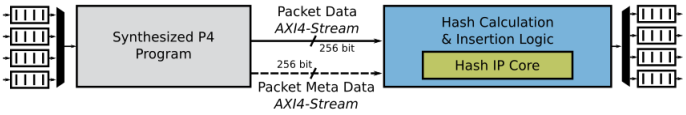

Figure 1: Integration of hash calculation and insertion

refer to related work regarding their cryptographic properties and cryptanalysis.

B. NPU: NFP-4000 SmartNIC

The 10G Netronome Flow Processor (NFP)-4000 Agilio SmartNIC [28] is a Network Processing Unit (NPU) that relies on a 32 bit many-core architecture with up to 60 freely programmable flow processing cores. A P4 compiler is offered by Netronome, which compiles P4 code for the NPU (SDK v6.0.4). While none of the supported hash functions has cryptographic properties, the SmartNIC allows implementing P4 externs in Micro-C, a variation of C used to program the processing cores. Externs are inlined into the compiled P4 program. In addition to the existing P4 hashes for CRC32 and Checksum, we have implemented the SipHash-2-4 function in Micro-C, calculating a hash for the payload of the Ethernet frame. The NFP-4000 features a hardware crypto security accelerator supporting SHA1 and SHA2, however, the accelerator was not available on our NPU, therefore we opted for the CPU-optimized SipHash instead.

C. FPGA: NetFPGA SUME

P4->NetFPGA [29] provides an open-source hardware design for the NetFPGA SUME board, which instantiates $P4_{16}$ programs compiled via Xilinx SDNet (we used version 2018.1). We selected the open-source RTL implementations of a *SipHash-2-4*[1] (64 bit output) and a *SHA3-512*[2] (512 bit output) IP core for integration into our prototype design.

Integrating the hash IP cores seamlessly as P4 externs via interfaces defined by the P4->NetFPGA implementation is not possible. The current design does not implement a streaming interface for extern data in- and output, where data is fragmented into multiple subsequent words. In- and output data is passed among the P4 program and externs as a single data word via a fixed number of parallel wires, requiring thousands of wires for maximum-sized Ethernet frames. We found that the current version of the SDNet compiler is only able to handle input widths of up to approx. 600 B. However, even for an input width of 64 B we were unable to obtain timing closure due to resource congestion.

As an alternative, we have changed the P4 switch model of the P4->NetFPGA design by integrating the hash calculation in the egress path after the synthesized P4 program (Figure 1). Per-packet metadata written by the P4 program instructs the hash module whether and where to insert the hash. As hashes are calculated after packets traverse the P4 program, packet modifications or forwarding decisions relying on hash calculations cannot be implemented in P4. Relocating the hash IP core into the ingress path would allow hashes to be passed to the P4 program via metadata. Another alternative is to further enhance the P4 switch model of the P4->NetFPGA by placing a second P4 pipeline after the hashing module.

Finally, our implementation would benefit from a traffic manager to selectively steer traffic around the IP core to avoid blocking of packets, which do not require hash calculation.

[1] SipHash IP Core: https://github.com/secworks/siphash

[2] SHA3-512 (KECCAK) IP Core: https://github.com/freecores/sha3

D. Limitations

Our extern implementations for the NFP-4000 and NetFPGA SUME do not use key material or an HMAC scheme required to generate a message authentication code, but only calculate a single cryptographic hash. This is done for simplicity and to focus on evaluating the performance of the basic cryptographic operation, which could be applied for use cases other than HMAC calculations. However, this functionality could be added, for instance by providing the key material as part of P4 metadata on a per-packet basis.

V. Prototype evaluation

Our measurement setup consists of two servers connected via a 10 Gbit/s Ethernet link. One server acts as a load generator and sends packets to the device under test (DuT), which runs an L2 forwarding P4 program that additionally calculates hashes based on the complete Ethernet frames. The server acting as the DuT is equipped with an Intel Xeon CPU E5-2620 v3 (Broadwell) at 2.40 GHz and either an Intel X540 network card, Netronome NFP-4000 SmartNIC, or the NetFPGA SUME. For measurements performed for the CPU target, all traffic is pinned to one CPU core.

A. Metrics for hash functions

Several metrics depending on the requirements of the application and capabilities of the (hardware) platform are of relevance when choosing a hash function. In high-performance applications, the performance of the hash function in terms of latency and processing time (e.g. clock cycles) is an important characteristic. When implementing the hash function, its memory footprint and, when implemented in hardware, resources of the hardware required, e.g. logic elements and registers for an FPGA, have to be considered.

Cryptographic properties of a hash function may be limited to a defined length (5-tuple vs. payload) and/or type of input data (entropy of passwords vs. random data). Furthermore, the function's collision resistance has to be taken into consideration. Finally, different applications may have different constraints regarding the length of the produced output hash. While a short HMAC included in an Ethernet frame causes only minor packet overhead, it can negatively impact its effectiveness.

B. Hash function micro-benchmarks

CPU system We investigate the individual latency of the hash algorithm implementations. Each hash function was executed multiple times for input data lengths ranging from

Hash algorithm	Cycles per B	Fixed cycles per packet	Cycles for 64 B	Output length (bit)
CRC32	0.32	0.00	10.79	32
Checksum	0.44	0.00	30.06	16
SipHash-2-4	1.06	56.40	121.10	64
Poly1305-AES	1.69	83.71	170.38	128
BLAKE2b	3.14	35.85	232.77	8-512
HMAC-SHA512	3.70	1454.51	1578.14	512
HMAC-SHA256	5.57	959.69	1462.13	256

Table I: Hash function latency on CPU DuT

2 B to 1500 B on the CPU DuT. For each run, we counted the number of CPU cycles using the timestamp counter (TSC). To model the latency behavior of each hash function, we then performed a linear regression based on the input data lengths and the number of cycles consumed by each algorithm.

The number of fixed CPU cycles per packet reported in Table I highlights that some algorithms are better suited to process small input data such as network packets than others. Especially when comparing HMAC-SHA256 and HMAC-SHA512 to the other cryptographic functions, an increase of per-packet fixed cycles by a factor of up to 40 can be seen. To process 14.88 Mpps for 10GbE with 64 B packets on a single CPU core clocked at 2.40 GHz, the processing for each packet must be completed in 161 CPU cycles. Only the non- or pseudo-cryptographic functions not using an HMAC mode satisfy this requirement. SipHash-2-4 shows the most promising results, being optimized for hashing on 64 bit CPU architectures.

FPGA We evaluate the hash IP cores in our NetFPGA SUME implementation through RTL simulations. Table II and Figure 2 present observed latency and throughput. SHA3-512 hash values are calculated on 72 B data blocks, SipHash-2-4 operates on smaller 8 B blocks. If the input data must be padded to fill the content of the last block, calculation efficiency (i.e. B/clock cycle) decreases and throughput drops. However, this is barely visible for SipHash-2-4 calculation due to the small block size. While the theoretical maximum throughput of the SHA3-512 IP core is 48 bit/clock cycle for infinitely long input data, we observe a maximum rate of 46.14 bit/clock cycle for packets between 64 B and 1518 B. The maximum throughput of the SipHash-2-4 IP core for these packet lengths is 21.02 bit/clock cycle, falling slightly below the theoretical maximum of 21.33 bit/clock cycle. Although the throughput of the SHA3-512 calculation is significantly higher, we were unable to operate the integrated IP core at clock frequencies exceeding 165 MHz. While the SipHash-2-4 logic can be operated at 200 MHz, matching the frequency of the P4->NetFPGA pipeline, we had to place the SHA3-512 IP core in a separate clock domain.

C. Hashing complete packets

For the use case of communication integrity and authentication, we evaluate the hashing of complete packets. For each platform, we perform a baseline measurement, where the P4 program is a simple L2 forwarder without performing any hashing operations.

Hash algorithm	Block Size in B	Cycles per Block	Fixed Cycles per Packet	Clock Frequency
SHA3-512	72	12	10	165 MHz
SipHash-2-4	8	3	8	200 MHz

Table II: Hash function latency on FPGA DuT

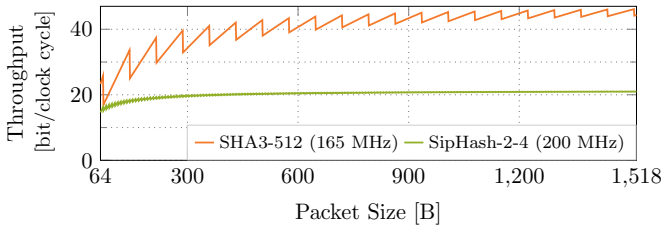

Figure 2: Throughput of hashing FPGA IP cores

Throughput Results for maximum throughput are presented in Table III. Independent of packet size, all three platforms reach 10 Gbit/s in the baseline scenario, with the exception for minimum-sized packets on the CPU target. Adding the calculation of hashes reduces the maximum performance such that no platform can reach line rate for packets with minimum size. In our evaluation, the best results are achieved by the Netronome card. Experiments with Checksum and CRC32 as hash algorithms showed that the card can hash packets at line rate regardless of packet size (not shown in Table III). Using SipHash-2-4 about 75 % of line rate for minimum-sized packets can be achieved. Despite high throughput for packet sizes up to 900 B, performance degrades rapidly for larger packets. This behavior can be explained by the SmartNIC's RAM architecture [30]. Our experiments showed that buffers residing in a fast memory region are only used for packets smaller than 900 B for payload processing. For larger packets, slower shared RAM has to be accessed, causing a drop in throughput to approx. $10^{-6}\%$ line rate.

The NetFPGA SUME platform achieves an almost constant

Algorithm	64 B	96 B	128 B	512 B	1024 B	1500 B
t4p4s						
Baseline	95.03	100	100	100	100	100
SipHash-2-4	36.09	46.01	54.73	100	99.17	100
HMAC-SHA512	8.47	11.69	11.11	24.26	31.67	37.80
NFP-4000						
Baseline	100	100	100	100	100	100
SipHash-2-4	75.60	80.71	91.61	99.15	10^{-6}	10^{-6}
NetFPGA SUME						
Baseline	100	100	100	100	100	100
SipHash-2-4	42.00	42.18	42.29	42.56	42.61	42.52
SHA3-512	48.21	42.53	54.27	65.02	71.78	76.00

Table III: Achievable throughput for hashing frames of different sizes in percent, relative to 10 GbE line rate

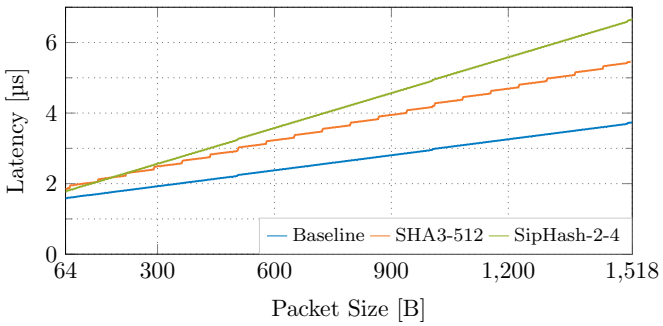

Figure 3: Median latency for NetFPGA (2.5 Gbit/s)

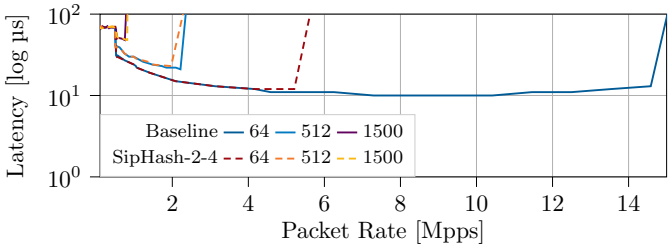

Figure 4: Median latency for CPU system

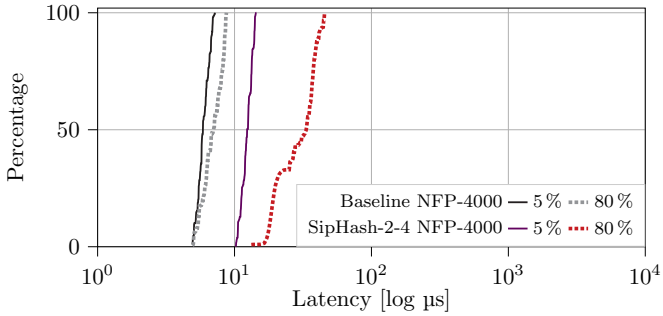

Figure 5: Latency distribution at 5 % and 80 % of respective maximum throughput using 64 B packets

	LUTs		Registers		BRAM	
	Abs.	%	Abs.	%	Abs. [kB]	%
Baseline	64,533	14.90	109,783	12.67	16,362	30.92
SipHash-2-4	66,380	15.32	114,282	13.19	17,460	32.99
SHA3-512	73,449	16.95	118,689	13.70	17,460	32.99

Table IV: Resource utilization for the NetFPGA SUME

performance of approx. 42 % line rate using SipHash-2-4. The SHA3-512 IP core is clocked slower, but its higher per-cycle throughput results in superior performance for all packet sizes. It reaches 76 % line rate for 1500 B packets. The non-monotonic increase of throughput is caused by the block-based hash calculation. While our prototype is limited to open-source hash implementations, we note that higher throughput could be achieved with commercial IP cores.

The worst performance is shown by the CPU target. Compared to the baseline, for SipHash-2-4 the throughput is more than halved for small packet sizes, roughly matching the calculated latency shown in Table I. Only for packets larger than 390 B line rate is reached. Due to the large number of fixed cycles per packet, SHA512 when used in HMAC mode processes less than 10% line rate for minimum-sized packets and even for large packets is unable to reach line rate.

Latency Figure 3 shows our latency measurements for the NetFPGA SUME. As expected, latency increases linearly with packet size with slight discontinuities due to the block-based hash calculation. We found that for each packet size the measured values do not differ by more than 100 ns.

For t4p4s latency is influenced by the packet rate (see Figure 4, results for HMAC-SHA512 omitted due to low maximum packet rates) as packets are sent either when a burst size of 32 is reached, or after a timeout. This causes increased latency for packet rates below 0.5 Mpps as the batch is not filled quickly enough, instead waiting for the timeout. The latency is independent of the algorithm used, however, increases with packet size (l) due to the increased serialization delay: $t_s(l) = ((l - 64) \cdot \mathrm{b_{size}})/(t_{s10\mathrm{GbE}})$ in relation to 64 B packets, with $\mathrm{b_{size}} = 32$ and $t_{s10\mathrm{GbE}} = 1.25ns$. Overall the latency is between 10 µs and 80 µs, however, outliers, which regularly occur when using the DPDK, exist (see Figure 5).

The NPU demonstrates stable behavior below 10 µs with no outliers for the baseline scenario. Performing the SipHash-2-4 operation shifts the latency distribution to the right up to 30 µs and increases the long tail.

Resource Consumption Packet processing in general is parallelizable, scaling well using multi-queue NICs and multi-core CPUs. Thus, the hardware of CPU-based systems can be tailored to meet an application's resource requirements.

Apart from the described performance issues, we did not encounter resource restrictions for the Netronome card as the P4 program is of small size even when adding the SipHash implementation. For other applications, the program may be too large such that the generated firmware image can no longer be loaded onto the card.

Finally, Table IV lists the resource consumption (LUTs, registers, BRAM) of the FPGA-based implementations. Adding hashing functionalities increases resource consumption only moderately by no more than approx. 2 %.

VI. Limitations

The performance of the evaluated platforms depends on the chosen hash function and their implementation. For this work, we selected open-source implementations, because we are primarily interested in the general feasibility of using cryptographic hash functions in programmable data planes. While we have shown that this is possible, more sophisticated hash function implementations (e.g. commercial FPGA IP cores) in combination with an optimized integration into the P4 program (e.g. parallelization, pipelining) could reduce implementation artifacts, improving the performance and resource utilization, but would require further monetary costs and engineering effort.

VII. Conclusion

Our review of the current use of hash functions in P4 applications reveals two insights. First, a prevalent use of CRC, making applications vulnerable to potential attacks targeting hash collisions. Second, protocols and applications requiring cryptographic hashes for authentication or integrity cannot be described using P4. Therefore, the implementation of cryptographic hash functions would increase the applicability of P4 to a wider range of use cases.

We describe prototype implementations integrating cryptographic hashing algorithms in three different P4 target platforms – CPU, NPU, and FPGA. Our analysis shows that the CPU target is easily extensible, but has the highest worse-case latency of up to several milliseconds. The tested NPU offers the highest throughput, but cannot process packets larger than 900 B efficiently. The FPGA-based target offers the lowest latency with small variance. However, the hashing IP core currently cannot be integrated using native P4 features, limiting the programmability and requiring a change of the P4 switch model.

Our measurements show hashing performance to be highly target, algorithm, and use case specific. Therefore, we cannot recommend a one-size-fits-all solution. We rather suggest that P4 targets should implement hash functions – operating on header and payload data – from a family of algorithms, which should be recommended by the P4 specification. These recommendations should include cryptographic hashes and take into account the unique characteristics of platforms such as CPU, NPU, FPGA, or even future ASICs.

Acknowledgment

This work was supported by the German Research Foundation (project ModANet under grant no. CA595/11-1) and by the German-French Academy for the Industry of the Future. The authors would like to thank Fabian Pusch for contributions to our FPGA prototype, Philipp Hagenlocher for the integration and evaluation of various hash functions in t4p4s, as well as the anonymous reviewers for their valuable feedback.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, Jul. 2014.

[2] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016.

[3] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-Hitter Detection Entirely in the Data Plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: ACM, 2017.

[4] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing Key-Value Stores with Fast In-Network Caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017.

[5] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "Vulnerability of Network Mechanisms to Sophisticated DDoS Attacks," *IEEE Transactions on Computers*, vol. 62, no. 5, May 2013.

[6] M. Molina, S. Niccolini, and N. Duffield, "A Comparative Experimental Study of Hash Functions Applied to Packet Sampling," in *International Teletraffic Congress (ITC-19)*, 2005.

[7] C. Henke, C. Schmoll, and T. Zseby, "Empirical Evaluation of Hash Functions for Multipoint Measurements," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, Jul. 2008.

[8] R. Jenkins, "A Hash Function for Hash Table Lookup," 1997. [Online]. Available: http://www.burtleburtle.net/bob/hash/doobs.html

[9] N. Hua, E. Norige, S. Kumar, and B. Lynch, "Non-crypto Hardware Hash Functions for High Performance Networking ASICs," in *Proceedings of the 2011 ACM/IEEE 7th Symposium on Architectures for Networking and Communications Systems*, Oct. 2011.

[10] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data Plane Performance Diagnosis of TCP," in *Proceedings of the Symposium on SDN Research (SOSR'17)*. ACM Press, 2017.

[11] E. Cidon, S. Choi, S. Katti, and N. McKeown, "AppSwitch: Application-layer Load Balancing Within a Software Switch," in *Proceedings of the First Asia-Pacific Workshop on Networking*, ser. APNet'17. New York, NY, USA: ACM, 2017.

[12] J. Kučera, D. A. Popescu, G. Antichi, J. Kořenek, and A. W. Moore, "Seek and Push: Detecting Large Traffic Aggregates in the Dataplane," 2018.

[13] J.-H. Choi, S.-G. Min, and Y.-H. Han, "MACsec Extension over Software-Defined Networks for in-Vehicle Secure Communication," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2018.

[14] B. Carnevale, L. Fanucci, S. Bisase, and H. Hunjan, "Macsec-based security for automotive ethernet backbones," *Journal of Circuits, Systems and Computers*, vol. 27, no. 05, 2018.

[15] E. Heidinger, C. Heller, A. Klein, and S. Schneele, "Quality of service IP cabin infrastructure," in *29th Digital Avionics Systems Conference*. IEEE, 2010.

[16] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection with MACsec in P4-SDN," *arXiv preprint arXiv:1904.07088*, 2019.

[17] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-IPsec: Implementation of IPsec Gateways in P4 with SDN Control for Host-to-Site Scenarios," *arXiv preprint arXiv:1907.03593*, 2019.

[18] Z. Cao, Z. Wang, and E. Zegura, "Performance of Hashing-Based Schemes for Internet Load Balancing," in *IEEE INFOCOM 2000*, 2000.

[19] S. A. Crosby and D. S. Wallach, "Denial of Service via Algorithmic Complexity Attacks," in *USENIX Security Symposium*, 2003.

[20] S. Goldberg and J. Rexford, "Security Vulnerabilities and Solutions for Packet Sampling," in *Proceedings of the 2007 IEEE Sarnoff Symposium*, Apr. 2007.

[21] J.-P. Aumasson and D. J. Bernstein, "SipHash: A Fast Short-Input PRF," in *Progress in Cryptology - INDOCRYPT 2012*, S. Galbraith and M. Nandi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[22] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, Smaller, Fast as MD5," in *Applied Cryptography and Network Security*. Springer Berlin Heidelberg, 2013.

[23] P. Hoffman and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols," RFC Editor, RFC 4270, Nov. 2005.

[24] W. M. Eddy, "TCP SYN flooding attacks and common mitigations," RFC 4987, 2007.

[25] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, and M. Tejfel, "High Speed Packet Forwarding Compiled from Protocol Independent Data Plane Specifications," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016.

[26] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption*, H. Gilbert and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[27] Austin Appleby, "Poly1305 git repository," 2016. [Online]. Available: https://github.com/floodyberry/poly1305-donna

[28] "NFP-4000 Theory of Operation," Netronome Systems Inc., Tech. Rep., 01 2016, last accessed: 2019-06-17. [Online]. Available: https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf

[29] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4->NetFPGA Workflow for Line-Rate Packet Processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019.

[30] S. Wray, "The Joy of Micro-C," 2014, https://open-nfp.org/media/documents/the-joy-of-micro-c_fcjSfra.pdf.