Slicing Networks with P4 Hardware and Software Targets

Eric Hauser, Manuel Simon, Henning Stubbe, Sebastian Gallenmüller, Georg Carle
Chair of Network Architectures and Services, Department of Informatics,
Technical University of Munich, Garching near Munich, Germany
{hauser,simonm,stubbe,gallenmu,carle}@net.in.tum.de

ABSTRACT

Running applications over a shared network may lead to unwanted impairments or performance impacts. To avoid these effects, the partitioning of network resources is an integral aspect of effective 5G networks. These virtually partitioned networks or slices allow the provisioning of network resources to guarantee a specific service quality to dedicated virtual networks. Programmable network devices, pushed by new languages such as P4, with their intrinsic flexibility, present themselves as a promising technique to realize slicing. This paper explores three approaches to network slicing and their respective implementation on a P4 soft- and hardware network device. We focus our effort on investigating P4 primitives that do not require the features of a specific P4 device but are available across different P4 targets. Based on our findings, we provide target-specific guidelines minimizing the impact of P4-based slicing for software and hardware targets alike.

CCS CONCEPTS

• **Networks → Network performance analysis**; **Network measurement**; *Programmable networks.*

KEYWORDS

P4, Network Slicing, Network Experiments

ACM Reference Format:
Eric Hauser, Manuel Simon, Henning Stubbe, Sebastian Gallenmüller, Georg Carle. 2022. Slicing Networks with P4 Hardware and Software Targets. In *ACM SIGCOMM 2022 Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases (5G-MeMU '22), August 22, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3538394.3546043

1 INTRODUCTION

Digital cellular networks are characterized by the coexistence of different communication tasks such as signaling, data, or voice. Each of these tasks requires a specific service quality. In 5G, the fundamental property of different service classes is extended to applications utilizing data over cellular networks. Three prominent service classes in 5G networks are the throughput-optimized enhanced mobile broadband (eMBB), the massive machine type communication (mMTC) for networks with many nodes, and ultra-reliable low-latency communication (URLLC) for highly critical applications.

A key concept to realize the different requirements of the previously mentioned service classes is network slicing [13]. Multiple logical networks or slices share the same physical infrastructure in sliced networks. A key challenge for network slicing is the partitioning of resources, e.g., hardware, to minimize the impact of the individual slices on each other. In this work, we investigate the suitability of programmable network devices for realizing sliced networks.

P4 [1] is a domain-specific language for programming different kinds of network devices, also called targets, such as software packet processing applications or switches. In addition, P4 allows the definition of network processing tasks on a low level. As 5G's different service classes have different requirements, a P4-driven network allows low-level optimizations to adapt to the currently running services without changing the hardware. The low-level control over the network enables a degree of flexibility that is unique to P4, making it a highly attractive platform for realizing different network services.

In the following, we identify different ways to utilize P4's capabilities to realize network slicing. The focus of our investigation is the utilization of vanilla P4. We want to avoid relying on specific hardware features available only on selected targets. Based on our findings, we provide measurements of the different slicing approaches on a software and a hardware P4 target. The results identify the benefits and disadvantages of the individual slicing approaches on different P4 targets and provide recommendations for their use.

The paper is structured as follows. Section 2 investigates related concepts and solutions. In Section 3, we propose three different slicing approaches. The implementation of these approaches is detailed in Section 4. Section 5 presents latency and performance measurements of the implemented slicing approaches. The implementations' strengths and weaknesses are discussed in Section 6. Section 7 concludes the paper.

2 RELATED WORK

5G networks and network slicing are highly active research topics, extensively described in literature. In this paper, we want to focus our investigation on slicing approaches of programmable data planes.

Ordonez-Lucena et al. [11] propose general concepts for slicing the 5G network infrastructure. Thus, enabling multiple networks,

5G-MeMU '22, August 22, 2022, Amsterdam, Netherlands
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9393-5/22/08…$15.00
https://doi.org/10.1145/3538394.3546043

here synonymous with slices, on top of a physical network infrastructure. Thereby, technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV) build the foundation of the sliced 5G infrastructure. While akin in terms of general concepts, our work adds the aspect of evaluating concrete approaches and their P4 implementations.

Wang et al. [19] present a hardware design for a multi-tenant programmable processing pipeline. The architecture is applied to two different FPGA-based NIC platforms, demonstrating a near-native performance for the implemented isolation mechanisms. A different multi-tenant architecture for a P4-programmable FPGA target is presented by Stoyanov et al. [17]. They consider the details of isolating tenants and suggest, among others, a privileged tenant responsible for managing access rights of other tenants. Moreover, Ricart-Sanchez et al. [12] propose QoS-aware slicing in the edge-to-core segment of 5G networks. All three works target a common hardware platform: FPGA instead of a software or ASIC target. In addition, they discuss the aspect of multi-tenancy. In contrast to this work, no comparison of different isolation approaches takes place.

Chen et al. [3] introduce the P4-TINS framework allowing network slicing on hardware switches. They rely on hardware-specific priority queues available in the Intel Tofino switching ASIC. TurboNet [2] is a network emulator for the Intel Tofino. While Cao et al. [2] touch on the aspect of realizing isolated links on this ASIC, their solution neither intends nor permits the execution of individual P4 code on the device. Instead, TurboNet focuses on the efficient deployment of arbitrary network topologies. To realize multi-tenancy on hardware ASICs, such as the Intel Tofino, Wang et al. [20] discuss how multiple programs can be run on a single piece of hardware. Apart from showing the feasibility, they intended to start a discussion on the vision for multi-tenancy. In an early work, Hancock et al. [7] describe HyPer4, a virtualization solution for P4-programmable devices. In contrast to our work, HyPer4 relies on target-specific hardware features to implement network slicing with P4. As the two previously mentioned publications, this work targets the Intel Tofino and its ASIC. This results in programs relying on non-vanilla P4 functionality specific to this target. In our paper, we include an ASIC-based target in our comparison. However, we eliminate target-specific extensions to allow for more general observations.

Finally, from a P4 language perspective, Soni et al. [16] discuss challenges and possible solutions when combining arbitrary programs. They suggest a framework that eases combining programs and running them on a single P4 target. While the main idea of such an abstraction would assist in combining programs of sliced networks, possible slicing approaches discussed here remain the same. Arguably, the abstraction would distract during the evaluation and comparison provided by this work.

P4 slicing is an active field of research investigated from various angles. Approaches consider isolation on a software or hardware level, with studies exploring different targets such as software, FPGA, or ASIC. However, the mentioned approaches use target-specific features supported by platforms to realize slicing. Relying on target-specific features restricts users to a specific device, such as Intel Tofino, or class of devices, such as FPGAs. Because of this restriction, we claim that target-specific slicing approaches need to be well justified. For us, this means that these target-specific slicing approaches must offer significant advantages compared to regular P4 targets. We restrict our investigation to vanilla P4 capabilities, i.e., features available on any P4-capable device. Our measurements provide a baseline performance for vanilla P4 slicing approaches across different targets. This baseline can act as a performance goal for the previously described target-specific slicing approaches.

3 SLICING WITH P4

P4 is an open-source programming language aimed at solving the heterogeneity in SDN by introducing programmability in a vendor-, protocol-, and target-independent way. The underlying model of P4 considers three parts: *parser*, *match-action pipeline*, and *deparser*. The *parser* processes incoming packets. Afterward, packets traverse the *match-action pipeline* consisting of one or more *control blocks*. Within these *control blocks*, matches to header fields and *actions* are performed. *Tables* provide the values against which header fields are matched and the associated action and their parameters. *Actions* allow arbitrary packet modifications and special instructions such as dropping the packet. The control plane can update table entries dynamically without changing the P4 program. Additionally, current research aims to provide updates to the tables directly from the data plane [15]. Modifying table entries directly reduces the update delay compared to the traditional method involving the control plane. Finally, the *deparser* reassembles packets before transmission.

The common goal of network slicing is the separation of the traffic for different logical networks. In this work, we propose three approaches to sharing the resources of a single physical P4 programmable device between multiple logical networks: *table*, *program*, and *hardware slicing*. Each method relies on specific P4 constructs to partition the logical networks.

Table slicing uses separate entries in P4 tables to discriminate between logical networks. Every entry in this P4 table is associated with a specific logical network. The table and its structure remain the same for all logical networks. This unmodifiable table structure determines the investigated header fields and the performed matches. The content of the P4 table can be changed without modification of the P4 code or the hardware simplifying the implementation of table slicing. P4 allows the modification of table entries at runtime. This way, entries for the different logical networks can be easily added, removed, or changed during operation. The table slicing method may increase the number of table entries, as entries are added for each logical network. Therefore, table slicing requires additional space for the table, which may lead to an increased energy consumption or impact the performance depending on the data structures used for the tables.

Program slicing allows providing complete processing tasks for each logical network. Specific actions can be defined and mapped to specific logical networks. In addition, individual tables using their specific structure can be created for each logical network. Individual actions and tables allow the implementation of highly individual functionality. In extreme cases, the specific processing tasks of the different logical networks share neither actions nor table structures. When combining logical networks that require different functionality, the P4 program must be adapted. Typically,

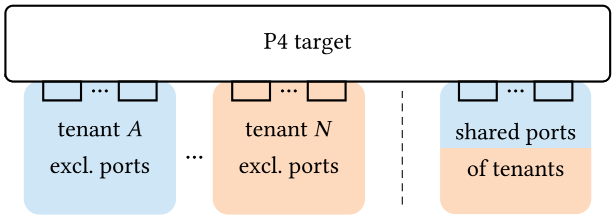


Figure 1: Flexible port allocation concept

this requires merging two different P4 programs into one. The combined P4 program needs to reflect the different processing paths of the separate logical networks. As a result, the combination of different processing paths with their specific complexity may impact the observed performance. Packets that require complex processing may delay the processing of other packets requiring less complex processing.

Hardware slicing achieves separation between logical networks on the hardware level. Additional hardware resources on the P4 target are required to realize hardware slicing. The additional resources are used to realize P4 processing pipes that rely on exclusive hardware resources. Therefore, the individual programs of the logical networks have exclusive access to the sliced hardware. Such exclusive hardware access allows the processing of packets with minimal impact on the other logical networks. Thus, the performance impact is low even for programs with highly different complexity. A disadvantage of providing the additional hardware resources is high costs. In addition, the hardware resources must be available; however, not all P4 targets may implement extra hardware resources to realize hardware slicing.

4 IMPLEMENTATION

This section presents our reference implementation for the three slicing approaches, i.e., table, program, and hardware slicing. Prior to that, we introduce a reference scenario that we use to illustrate our assumptions.

To foster the reproducibility of our results, we published our implementation: *https://github.com/tumi8/memu-5g22-p4-slicing*

4.1 Scenario Description

The concept of network slices in 5G enables diversified services sharing one physical network. Therefore, the physical network operator rents independent logical networks, or network slices, to the tenants. In our proposed implementation scenario, we associate one logical network with one specific tenant. Thus, tenants are independent and have different applications to be executed on their network slice. For all three approaches, we isolate the tenants by strictly separating their traffic. To achieve this separation, we split the data plane's available ports into exclusive and shared ports. Figure 1 depicts this flexible port allocation concept. Separation is implemented using Virtual LANs (VLANs) according to IEEE 802.1Q [8]. Thereby, we assign an individual VLAN identifier (VID) to every tenant. Exclusive ports, i.e., access ports, are solely bound to one tenant. Shared ports, i.e., trunk ports, can be used by multiple tenants simultaneously.

In our implementation, the assignment of incoming packets to the respective tenant on the exclusive ports is directly based on the receiving port number. In contrast, for the shared ports, the VID is used to distinguish the packets of different tenants. Packets arriving at unallocated ports or with an unknown VID are dropped. A possibly existing VLAN header is removed for all outgoing packets on a tenant's exclusive port. On the contrary, a VLAN header is inserted for the shared ports. In this case, the assigned VID depends on the initially mapped tenant. Direct communication between different VLANs is prohibited; however, packets may be exchanged outside the VLAN domain.

4.2 Table Slicing

Table slicing allows tenants to write table entries to a predefined table structure. Therefore, the table structure is fixed and cannot be individually adjusted by the tenants. If a packet matches a tenant's table entry, the respective tenant can select actions from a pool provided by the P4 target operator. We propose the following example actions for our implementation: forwarding to one of the valid shared or exclusive output ports, dropping packets, and modifying the header fields of packets. Custom actions that are not part of the pool of predefined actions are not supported. Our implementation focuses on enabling support for widely used protocols such as IPv4, IPv6, UDP, and TCP. The proposed table structure has a predefined set of keys to match the table entries. The predefined table keys follow the standard 5-tuple model. Therefore, the key fields are fixed to the network layer's source and destination addresses, the used protocol on the transport layer, and the transport layer's source and destination ports. Other table keys or actions can support further protocols depending on the actual use case.

4.3 Program Slicing

In contrast to table slicing, program slicing offers more customizability to the tenants. Instead of adding entries to a predefined table structure, tenants have complete control over a single control block. Within a control block, arbitrary P4 operations can be executed. This includes custom actions and regular if-else statements. Tenants can also define their own tables. As long as the resources, e.g., special memory for fast table lookups, of the respective P4 target are not exhausted, there are no limits on table entries or the number of tables in total. However, because the tenants only have access to their control block, parser and deparser structures are shared across all tenants. To allow sharing parsers and deparsers among tenants, our implementation provides a common parser supporting various standard protocols. This default parser is applied to each packet processed by our P4 implementation. The tenants can access the headers of the parsed packets from within the tenant-defined control blocks and use the available information for processing. After the control block processing has ended, the initially parsed headers are deparsed by the provided deparser implementation. Our deparser implementation supports the same standard protocols as the initially defined parser.

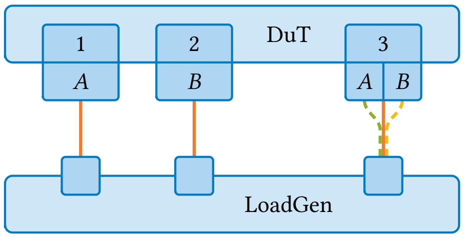


Figure 2: Measurement setup

4.4 Hardware Slicing

Hardware slicing aims to lift all limitations of the table and program slicing approaches. Instead of control blocks for every tenant, tenants can use all features of the P4 programming language. Therefore, the tenants define their programs individually, including parsers, control blocks, and deparsers. The required tenant-specific parsers, for instance, enable support for new protocols. Hardware slicing is implemented differently depending on the desired P4 target. Multiple P4 programs can run simultaneously using separate CPU cores on software-based targets. In contrast, hardware targets like specialized ASIC switches typically feature multiple independent processing pipes. In the default case, all pipes execute the same P4 program. However, some targets support the assignment of different programs to each pipe. The actual implementation depends on the used P4 target. We only support one specialized ASIC switch with four parallel pipes in our implementation. Therefore, the total number of tenants participating in the hardware slicing implementation is limited to four. A combination of the hardware slicing approach and the table or program slicing approaches is possible to support a larger number of tenants.

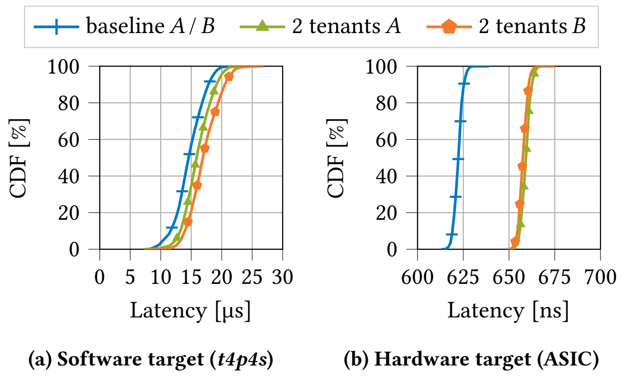
5 EVALUATION

This section presents latency measurements of the proposed network slicing approaches. We compare multi-tenant examples with baseline measurements to identify the performance impacts of the different slicing approaches. Thereby, the baseline measurement represents a single-tenant setup but keeps all introduced features like exclusive and shared ports.

5.1 Measurement Setup

We compare the performance of two different P4 targets for the evaluation: *(i)* a hardware target based on a switching ASIC and *(ii)* the software target *t4p4s* [18]. T4p4s is based on the Data Plane Development Kit (DPDK) [10], a high-performance packet processing framework. We use *t4p4s* based on commit `a3a54e37` with custom latency optimizations. Despite fundamental differences, we aim to keep the measurement setups for both targets as similar as possible. Furthermore, we keep the overall setup as simple as possible to eliminate any additional influences. Therefore, we use only one isolated CPU core for the evaluation of the software target. We use the *plain orchestrating system (pos)* [6] methodology to create fully automated, reproducible experiments and experiment evaluations. The overall measurement setup is depicted in Figure 2. The setup consists of two devices, the P4 target as Device under Test (DuT) and a load generator (LoadGen). The setup supports two active tenants, A and B, utilizing two exclusive ports (1, 2) and a shared port (3). All three ports of the P4 target are connected to the load generator via separate physical links. As described in Section 3, we use VIDs 100 and 200 to separate traffic on the shared port for tenants A and B, respectively. We use *MoonGen* [5] as packet generator sending 84 B sized packets to the DuT. In addition to the packets, MoonGen periodically samples the end-to-end latency with additional hardware-timestamped packets. The used measurement hardware allows the collection of timestamps with nanosecond resolution [9]. All packets are sent to the shared port 3 of the DuT. Depending on the targeted tenant, the packets are dropped or forwarded from port 3 to ports 1 or 2 of the DuT. The chosen forwarding direction simplifies the overall measurement setup because different tenants can be addressed by adjusting the packet's VID.

For the load generator and the software target, we use servers equipped with *AMD Epyc 7542* (32 cores, base clock 2.9 GHz, max. turbo clock 3.4 GHz), 512 GB RAM, dual-port 100 Gbit/s *Intel E810* NIC, and quad-port 25 Gbit/s *Intel E810* NIC. The hardware target is a high-performance switching ASIC with four separate processing pipes supporting up to 100 Gbit/s per port. We use 100 Gbit/s links between the hardware target and the load generator and 10 Gbit/s links between the software target and load generator. To avoid overloading the software target, we limit the packet rate to 2.1 Mpps. This limitation derives from the comparably expensive unboxing of the VLAN header, which requires copying different parts of the packet. As the hardware target offers higher throughput, the measurement is done at 110 Mpps. A further packet rate increase is not possible because we are limited by the maximum transmit packet rate of the used *Intel E810* NIC.



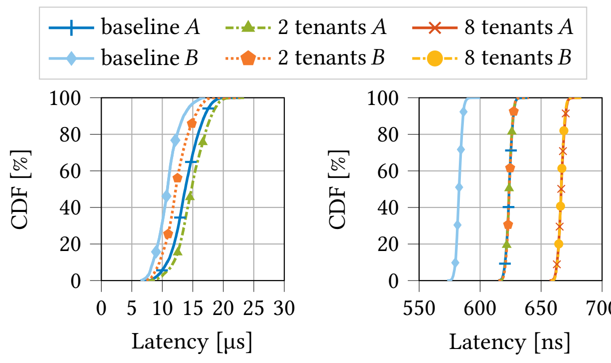
(a) Software target (*t4p4s*) **(b) Hardware target (ASIC)**

Figure 3: Latency CDFs for table slicing

5.2 Table Slicing

The measurements in Figures 3a and 3b compare the latency distributions of the table slicing approach for the software and hardware target, respectively. Observed latencies are displayed as cumulative distribution functions (CDFs) for each of the different measurement runs. For both targets, we compare a one-tenant baseline with a two-tenant scenario.

Regarding the *software target*, a single tenant uses 7500 entries each for the IPv4 and IPv6 tables, thus raising the total number of entries to 15 000. Accordingly, in the two-tenant scenario, tenants A



(a) Software target (*t4p4s*) **(b) Hardware target (ASIC)**

Figure 4: Latency CDFs for program slicing

		tenant entries		stages	
setup	total entries	*A*	*B*	*A*	*B*
baseline	15 000	15 000	0	4	2
2 tenants	15 000	15 000	0	4	4
8 tenants	40 000	15 000	0	8	8

Table 1: Required stages on the hardware target

and B have 15 000 entries each, resulting in a total of 30 000 entries for this table-sliced program. The two-tenant latency distribution is close to the baseline (*blue*), with a median latency of 14.6 µs. The median for tenant A (*green*) is increased by approx. 1.3 µs; the median for tenant B (*orange*) by approx. 2.3 µs. The latency increase between the baseline and tenant A is caused by the overhead of the table slicing approach. In general, the number of tenants' table entries has only a minor impact on the expected latency per tenant because *t4p4s* uses a dynamically-sized hash table. This statement holds as long as all entries fit into the software target's available cache [14, 15]. However, we observe a higher latency for tenant B compared to A. The likely reason for this is the sequential evaluation of VIDs on the software target. Together with the discussion about table sizes before, it implies that a single tenant does not have a significant disadvantage when using table slicing, as long as the entries of all tenants fit into the cache.

Investigating the *hardware target*, we raise the number of entries for each table to 40 000, resulting in 80 000 entries for the single-tenant and in 160 000 entries for the two-tenant scenario, respectively. The latency distributions of both tenants in the sliced scenario are highly similar, indicating that the latencies of all tenants are directly related to the total number of table entries. This behavior can be explained by the number of active stages per processing pipe. Every pipe is divided into sub-units called stages. Thereby, every stage has a fixed amount of available hardware resources, e.g., SRAM or TCAM memory. The hardware target activates no more than the minimum number of required stages. The more table entries are used, the more stages have to be activated on the processing pipes. As a result, there is a direct relation between the total number of entries and the expected total latency of the table-sliced program. For this behavior, we introduce the term *inter-tenant interference*. With inter-tenant interference, we consolidate all effects when the actions of a single tenant affect the performance of other tenants. Regarding the table slicing approach, we observe a minimal inter-tenant interference for the software target that is neglectable. In contrast, a noticeable inter-tenant interference occurs on the hardware target.

5.3 Program Slicing

The measurements shown in Figure 4a compare the latency distributions of the program slicing approach for the software target and in Figure 4b for the hardware target, respectively. To demonstrate the customizability of the individual control blocks, we show the two tenants implementing different applications. Tenant A runs an access control list (ACL) to filter incoming packets while tenant B implements a simple forwarder. Using this setup, tenant A allocates tables and therefore requires more hardware resources than tenant B. For both tenant applications, we conduct a baseline measurement indicating the single-tenant latency, for tenant A (*blue*) and tenant B (*light blue*), respectively. Due to the reduced hardware effort of the forwarder, tenant B achieves a median latency of 10.8 µs on the software target and 583 ns on the hardware target. In contrast, tenant A's ACL achieves a median latency of 13.8 µs and 624 ns, respectively. Next, we measure the latency when tenant A's (*dash-dotted green*) and tenant B's (*dotted orange*) applications run simultaneously using the program slicing approach.

On the *software target*, we observe a median latency increase between the baseline and the sliced version for tenant B (forwarder) of 1.4 µs. Furthermore, the latency distribution for tenant A (ACL) is shifted by approx. 1.5 µs. This slight increase for both targets is caused by the general overhead of the program slicing approach. However, in general, both two-tenant latency CDFs are close to their respective baseline latency CDF. An explanation for this behavior is the software target's "run-to-completion" paradigm. In this paradigm, one packet after another is processed, to minimize expensive in-memory moves [4]. Therefore, the runtime for each packet is flexible. After selecting the associated control block for the tenant, only this block has to be executed. As a result, the tenant's individual latencies follow the respective baseline latencies plus a small slicing overhead. Therefore, only a minimal inter-tenant-interference is visible for the software target.

The measured latencies of the *hardware target* show a different result. Tenant B's forwarding latency moved from its baseline (median 583 ns) to tenant A's baseline latency (median 624 ns). This behavior can be explained by the hardware target's ASIC architecture. If the complexity of a P4 program increases, e.g., total lines of code, conditional statements, or more table entries, the hardware target activates more processing stages. With every additional stage, the latency increases by a specific offset. Moreover, all packets must traverse through all activated stages regardless of the executed tenant application. Due to tenant A's increased hardware usage, which activates more stages, tenant B's latency increases equally. To further illustrate this behavior, we implement an eight-tenant setup on the hardware target. Additionally, this exemplary

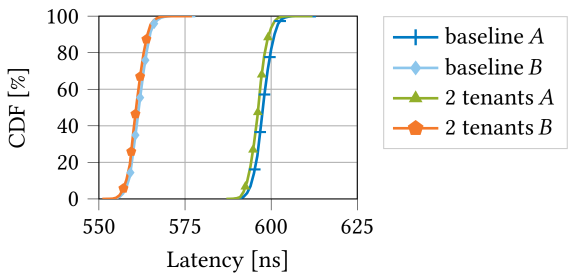


Figure 5: Latency CDFs for hardware slicing (ASIC)

eight-tenant setup serves as demonstration for the overall scalability of all presented slicing approaches. In this eight-tenant setup, tenant B's forwarder (*dash-dotted yellow*) further suffers from the hardware usage of the other seven tenants, which all run an ACL. Moreover, even tenant A (*red*) suffers from the required hardware resources of the new tenants C to H. The overall median latency in the eight-customer scenario increases to 667 ns. Table 1 emphasizes the relation between the allocated number of table entries and the required stages. Even though tenant A does not require more table entries than in the two-tenant setup, the total latency increases because of the other tenants' table entries. In general, the hardware target's expected total latency for the program slicing approach derives from the combined hardware resource requirements of all tenants. Therefore, we again observe a noticeable inter-tenant interference in terms of latency for the hardware target.

5.4 Hardware Slicing

The third approach, hardware slicing, promises more separation and closer hardware access to the tenants. By now, hardware slicing is only implemented in the ASIC switching *hardware target*. We keep the same applications for tenants A and B as in the previous program slicing measurements. The latency distributions are depicted in Figure 5. In contrast to the table and program slicing measurements on the hardware target, tenant B's (*orange*) latency is not influenced by tenant A (*green*). The latency distribution of the baseline scenario for both tenants A (*blue*) and B (*light blue*) are close to their sliced versions. Because every tenant has an exclusively assigned pipe, a tenant's increased hardware usage does not affect the other tenants. As a result, every tenant has an exclusive slice of the available physical hardware. Therefore, any inter-tenant interference is completely eliminated when using the hardware slicing approach.

6 DISCUSSION

The three approaches offer a different amount of flexibility and sovereignty as depicted in Figure 6. The table slicing approach features the least flexibility having a fixed program logic. Program slicing offers high flexibility by allowing different logic for different tenants. While table and program slicing are compatible with all targets, the hardware slicing approach is currently only supported on particular targets since it does not rely on vanilla P4 mechanisms.

When it comes to performance, software and hardware targets differ. While *hardware targets* generally have a better performance,

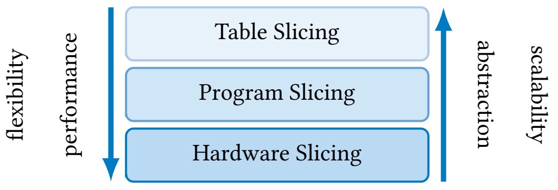


Figure 6: Comparison of slicing approaches

their pipes support only a fixed number of stages. Therefore, they impose the cost of slicing on all tenants when using table or program slicing. The sliced performance is worse than the performance of the most complex single-tenant baseline. High-complexity tenants may introduce performance penalties for low-complexity tenants. This is not the case for the hardware slicing. In *software targets*, performance implications are less severe. Latency is typically increased, but one tenant is not significantly penalized by the complexity of the others.

7 CONCLUSION

Network slicing is an important technique to realize the different service classes of 5G networks sharing the same underlying network equipment. An essential aspect of network slicing is ensuring the isolation of slices, especially when it comes to the quality of provided services. Programmable data planes, enabled by the programming language P4, are a hot topic facilitating the move of functionality towards tenants. In this paper, we present, discuss, and evaluate three variants of network slicing on two well-known P4 targets: table, program, and hardware slicing, on a performance-oriented ASIC-based hardware target and a flexibility-focused DPDK-based software target. Our results suggest that P4-programmable targets are capable of network slicing in all proposed variants. However, properly isolating tenants to avoid unintended performance influences is a challenging endeavor. Depending on the target and the slicing approach, we observed different aspects of inter-tenant interference. These interferences are eliminated only for the hardware slicing approach. However, this hardware-oriented approach is currently only available for specific hardware targets. In future work, we plan to bring this approach to the software target. The required parallelism can be implemented on multi-core CPUs by dedicating individual cores to specific clients.

The possibility of enabling tenants to deploy custom logic on network devices offers attractive application opportunities. Moreover, with careful planning, the expected performance changes can be embedded in appropriate service guarantees.

ACKNOWLEDGMENTS

This project has received funding from the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project 6G Future Lab Bavaria. Additionally, we received funding by the German Federal Ministry of Education and Research (BMBF) under the project 6G-life (grant number 16KISK001K). Furthermore, we would like to thank the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: programming protocol-independent packet processors. *Comput. Commun. Rev.* 44, 3 (2014), 87–95. https://doi.org/10.1145/2656877.2656890

[2] Jiamin Cao, Ying Liu, Yu Zhou, Lin He, and Mingwei Xu. 2022. TurboNet: Faithfully Emulating Networks With Programmable Switches. *IEEE/ACM Transactions on Networking* (2022), 1–15. https://doi.org/10.1109/TNET.2022.3142126

[3] Yan-Wei Chen, Chi-Yu Li, Chien-Chao Tseng, and Min-Zhi Hu. 2022. P4-TINS: P4-driven Traffic Isolation for Network Slicing with Bandwidth Guarantee and Management. *IEEE Transactions on Network and Service Management* (2022), 1–1. https://doi.org/10.1109/TNSM.2022.3159232

[4] Mihai Dobrescu, Norbert Egi, Katerina J. Argyraki, Byung-Gon Chun, Kevin R. Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. 2009. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, Jeanna Neefe Matthews and Thomas E. Anderson (Eds.). ACM, 15–28. https://doi.org/10.1145/1629575.1629578

[5] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring (Eds.). ACM, 275–287. https://doi.org/10.1145/2815675.2815692

[6] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. 2021. The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments. In *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021.* ACM, 259–266. https://doi.org/10.1145/3485983.3494841

[7] David Hancock and Jacobus van der Merwe. 2016. HyPer4: Using P4 to Virtualize the Programmable Data Plane. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies* (Irvine, California, USA) *(CoNEXT '16)*. Association for Computing Machinery, New York, NY, USA, 35–49. https://doi.org/10.1145/2999572.2999607

[8] IEEE. 2018. Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), 1–1993. https://doi.org/10.1109/IEEESTD.2018.8403927

[9] Intel 2021. *Intel Ethernet Controller E810 Datasheet.* Intel. Rev. 2.3.

[10] Intel. 2022. Intel DPDK: Data Plane Development Kit. http://dpdk.org Last accessed: 2022-06-30.

[11] Jose A. Ordonez-Lucena, Pablo Ameigeiras, Diego R. López, Juan J. Ramos-Muñoz, Javier Lorca, and Jesús Folgueira. 2017. Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges. *IEEE Commun. Mag.* 55, 5 (2017), 80–87. https://doi.org/10.1109/MCOM.2017.1600935

[12] Ruben Ricart-Sanchez, Pedro Malagón, Jose M. Alcaraz-Calero, and Qi Wang. 2019. P4-NetFPGA-based network slicing solution for 5G MEC architectures. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2019, Cambridge, United Kingdom, September 24-25, 2019.* IEEE, 1–2. https://doi.org/10.1109/ANCS.2019.8901889

[13] Peter Rost, Christian Mannweiler, Diomidis S. Michalopoulos, Cinzia Sartori, Vincenzo Sciancalepore, Nishanth Sastry, Oliver Holland, Shreya Tayade, Bin Han, Dario Bega, Danish Aziz, and Hajo Bakker. 2017. Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks. *IEEE Commun. Mag.* 55, 5 (2017), 72–79. https://doi.org/10.1109/MCOM.2017.1600920

[14] Dominik Scholz, Henning Stubbe, Sebastian Gallenmüller, and Georg Carle. 2020. Key Properties of Programmable Data Plane Targets. In *Teletraffic Congress (ITC 32), 2020 32nd International.* Osaka, Japan.

[15] Manuel Simon, Henning Stubbe, Dominik Scholz, Sebastian Gallenmüller, and Georg Carle. 2021. High-Performance Match-Action Table Updates from within Programmable Software Data Planes. In *ANCS '21: Symposium on Architectures for Networking and Communications Systems, Layfette, IN, USA, December 13 - 16, 2021.* ACM, 102–108. https://doi.org/10.1145/3493425.3502759

[16] Hardik Soni, Myriana Rifai, Praveen Kumar, Ryan Doenges, and Nate Foster. 2020. Composing Dataplane Programs with μP4. In *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*, Henning Schulzrinne and Vishal Misra (Eds.). ACM, 329–343. https://doi.org/10.1145/3387514.3405872

[17] Radostin Stoyanov and Noa Zilberman. 2020. MTPSA: Multi-Tenant Programmable Switches. In *EuroP4@CoNEXT 2020: Proceedings of the 3rd P4 Workshop in Europe, Barcelona, Spain, December 1, 2020.* ACM, 43–48. https://doi.org/10.1145/3426744.3431329

[18] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. 2018. T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors. In *IEEE 19th International Conference on High Performance Switching and Routing, HPSR 2018, Bucharest, Romania, June 18-20, 2018.* IEEE, 1–8. https://doi.org/10.1109/HPSR.2018.8850752

[19] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. 2022. Isolation Mechanisms for High-Speed Packet-Processing Pipelines. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22).* USENIX Association, Renton, WA, 1289–1305. https://www.usenix.org/conference/nsdi22/presentation/wang-tao

[20] Tao Wang, Hang Zhu, Fabian Ruffy, Xin Jin, Anirudh Sivaraman, Dan R. K. Ports, and Aurojit Panda. 2020. Multitenancy for Fast and Programmable Networks in the Cloud. In *12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020*, Amar Phanishayee and Ryan Stutsman (Eds.). USENIX Association. https://www.usenix.org/conference/hotcloud20/presentation/wang