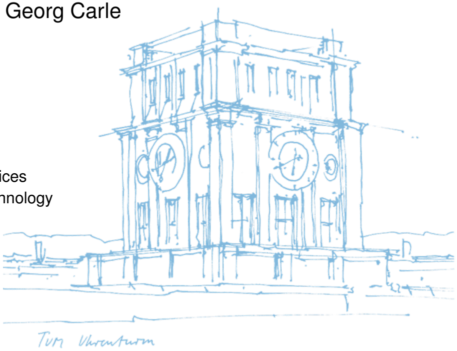
On-the-fly Table Insertions on Programmable Software Data Planes

Manuel Simon, Sebastian Gallenmüller, Georg Carle

Monday 28th October, 2024

CNSM'24

Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich



Introduction

State Keeping in Data Planes

- 6G aims for low-latency but high-resilient communication
- State keeping is essential for many applications
- *Registers* (*arrays*) are unstructured memory areas accessible by indices
    - may be fragmented in memory
    - no matching support
    - limited functionality
- In *tables*, structured state can be accessed by sophisticated key matching
- State is often kept by the control plane, which decreases performance for state-heavy applications
- We implemented state keeping via *tables* directly in the data plane

Introduction

Background

P4

- P4 [2] is a domain-specific language for SDN data planes
- In P4, *registers* are changeable within the data plane, *tables* only by the control plane

→ Updatable table entries would increase performance

  → In **previous work** implemented them for the P4 software target *T4P4S* using an `@__ref` annotation [8]

Introduction

Background

P4

- P4 [2] is a domain-specific language for SDN data planes
- In P4, *registers* are changeable within the data plane, *tables* only by the control plane

→ Updatable table entries would increase performance

  → In **previous work** implemented them for the P4 software target *T4P4S* using an `@__ref` annotation [8]

  → **Here**, we present add-on-miss insertions to tables [7]

Introduction

Background

P4

- P4 [2] is a domain-specific language for SDN data planes
- In P4, *registers* are changeable within the data plane, *tables* only by the control plane

→ Updatable table entries would increase performance

  → In **previous work** implemented them for the P4 software target *T4P4S* using an `@__ref` annotation [8]

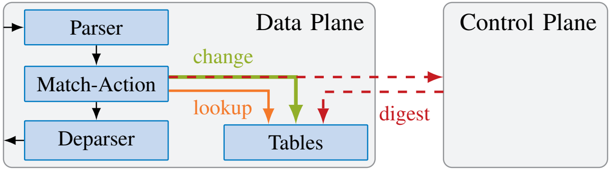
  → **Here**, we present add-on-miss insertions to tables [7]

T4P4S

- *T4P4S* [9] is a hardware-independent transpiler from P4 to C code linked with DPDK developed by ELTE
- The Data Plane Development Kit (DPDK) is an open-source framework enabling fast packet processing in user space
- DPDK performs Receive Side Scaling (RSS) to split traffic among several *lcores*/threads

Table Updates

Digest - Current P4 Way



Current State

- For changes in match-action tables, the data plane has to send a digest to the control plane
  - in *T4P4S*: the controller is a separate process, communication via a socket (low round-trip time (RTT))
- Controller requests data plane to update the table

→ Digest-based approach introduces overhead

Approaches

- **Digest:** introduces a sleep of 1 second or 1 RTT

  ⇒ impractical for frequent updates
- **Add-On-Miss:** direct update in the data plane

  ⇒ avoids the detour over the controller

  ⇒ improves performance

Related Work

- The upcoming Portable NIC Architecture (PNA) [1]
  - brings P4 to the NIC/SmartNIC
  - will allow adding entries on lookup misses
- FlowBlaze [6] allows state updates in programmable data planes relying on registers
- Switcharoo [3] implements a key-value store entirely in the P4 Tofino data plane

- Swing State [5] allows consistent state migration to other P4 nodes
- P4Update [11] implements districtued consistent network updates in P4
- SwiSh [10] implements a distributed state layer to programmable switches

Add-On-Miss – Implementation

- Upcoming P4 Portable NIC Architecture (PNA) defines new table property: `add_on_miss` and new extern for `exact` matches

Add-On-Miss – Implementation

- Upcoming P4 Portable NIC Architecture (PNA) defines new table property: `add_on_miss` and new extern for `exact` matches

```
table forward {
        actions= {forward, add}
        key = {hdr.eth.srcAddr: exact;}
        add_on_miss = true;
        default_action=add;
}
```

```
action forward(bit<48> dstMac) {
        ...
}

action add() {
        bit<48> dstMac = 0xffffffffffff;
        add_entry<forward_params_t>
                ("forward", {dstMac});
}
```

Add-On-Miss – Implementation

- Upcoming P4 Portable NIC Architecture (PNA) defines new table property: `add_on_miss` and new extern for `exact` matches

```
table forward {
        actions= {forward, add}
        key = {hdr.eth.srcAddr: exact;}
        add_on_miss = true;
        default_action=add;
}
```

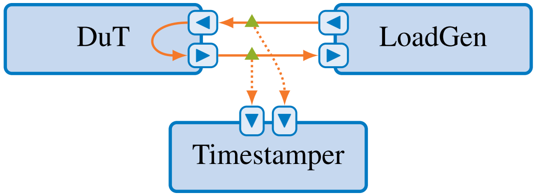
```
action forward(bit<48> dstMac) {
        ...
}

action add() {
        bit<48> dstMac = 0xffffffffffff;
        add_entry<forward_params_t>
                ("forward", {dstMac});
}
```

- For our implementation of these language features in *T4P4S*, we profit from the adaptions to the synchronization mechanism of the tables done in previous work

Evaluation

Setup



DuT

- Intel Xeon D-1518 2.2 GHz, 32 GB RAM
- Latency optimized *T4P4S*
- `add_on_miss` activated

LoadGen

- MoonGen [4] is used to generate traffic
- Contains key and value of new entry
- Packet size 84 B

Timestamper

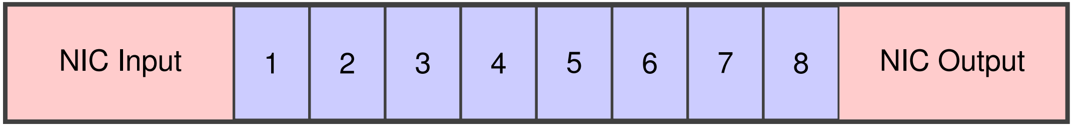
- Packet streams duplicated using optical splitter
- Timestamps each packet incoming packet
- Resolution: 12.5 ns

Evaluation

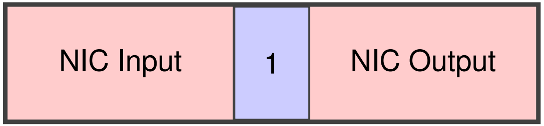
Batched processing

- NIC I/O has nearly constant overhead
- One packet is processed after another

Throughput-optimized $\rightarrow$ larger batch size

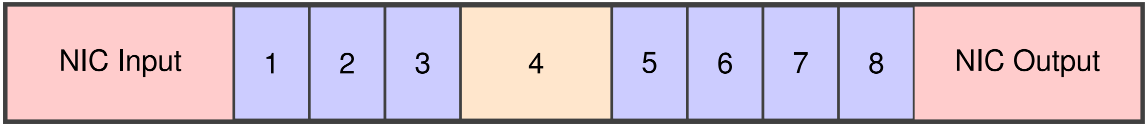


Latency-optimized $\rightarrow$ minimal batch size



Evaluation

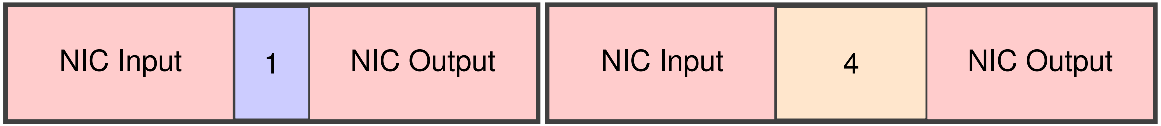
Throughput-optimized → larger batch size



→ Throughput measures average cost per packet

→ Ideal to measure the maximum performance

Latency-optimized → minimal batch size



→ Latency measures single cost for each packet

→ Ideal to measure cost of different operations

Evaluation

Approach

P4 program

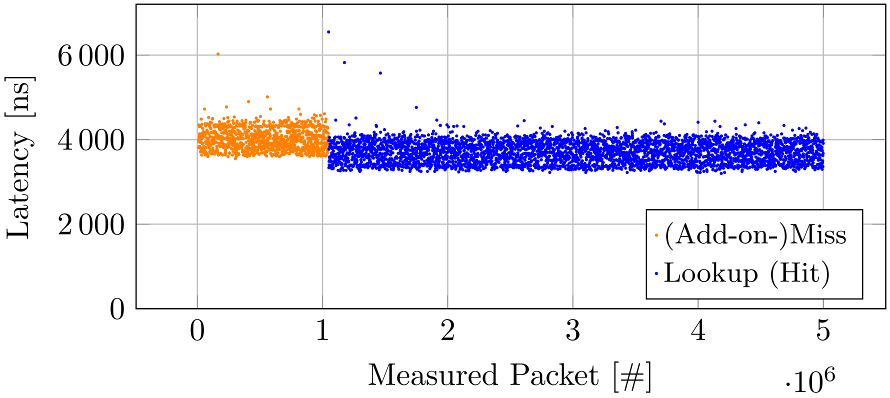
- Each packet contains key and value for a new table entry
- P4 programs contain lookup to this *specific* table
- Forward all packets back

Two phases

- Keys cycle pseudo-randomly through $[0, 2^{20}]$ several times
- *First phase*: only insertions are performed
- *Second phase*: mainly lookups are performed; some insertions are done with different rates

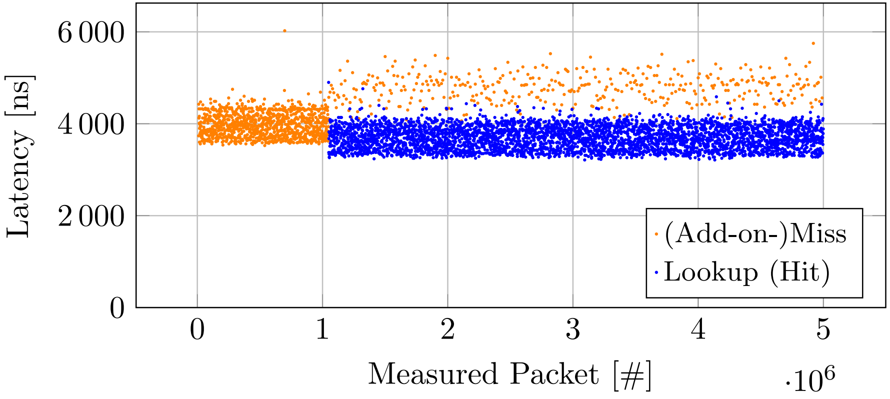
Evaluation

- *First phase*: 2^{20} packets triggering an insertion
- *Second phase*: $\approx 4M$ packets trigger lookup of previously inserted packets



Evaluation

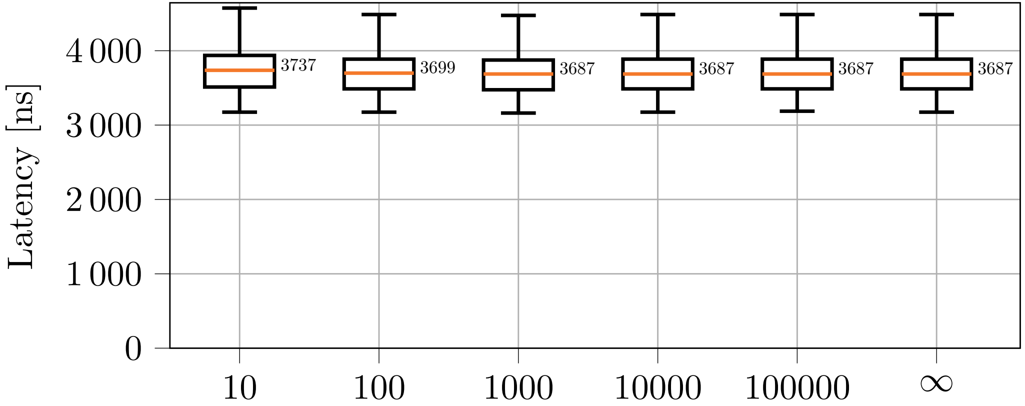
- *First phase*: 2^{20} packets triggering an insertion
- *Second phase*: $\approx 4M$ packets trigger lookup of previously inserted packets
  - But every 10 000-th packet triggers additional insertion



Evaluation

- Different rate of insertions during *second phase*

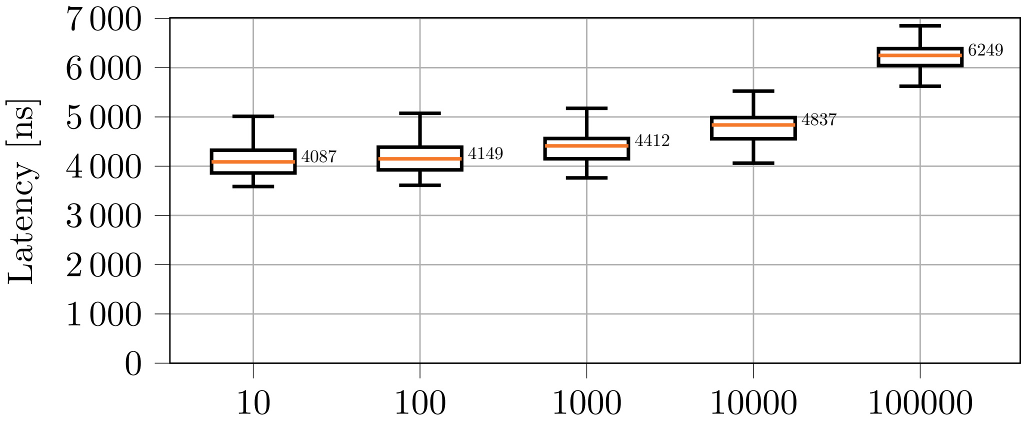
⇒ Median mixed (i.e. insertions & lookups) latency decreases with increasing rate



Evaluation

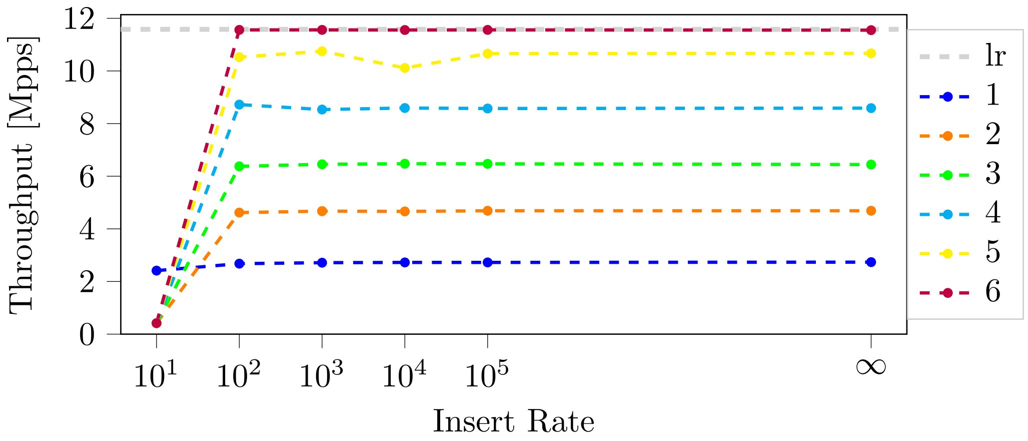
⇒ Insertion latency increases with increasing rate (up to 47 %)

⇒ Worse branch prediction



Evaluation

- Measured in a throughput-optimized version using Intel Xeon E5-2620 v2 2.1 GHz
- For reasonable insert rates, the approach scales linearly



Conclusion

- Adding state to the P4 data plane increases number of possible low-latency applications
  - Updatable Table Entries[1]
  - Add-On-Miss Insertions
- Add-on-Miss insertions enable cheap insertions w.r.t. latency

[1] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, and G. Carle: High-Performance Match-Action Table Updates from within Programmable Software Data Planes, *EuroP4 '21* [8]

Conclusion

- Adding state to the P4 data plane increases number of possible low-latency applications
  - Updatable Table Entries[1]
  - Add-On-Miss Insertions
- Add-on-Miss insertions enable cheap insertions w.r.t. latency

- Is this a step backwards in SDN ?

[1] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, and G. Carle: High-Performance Match-Action Table Updates from within Programmable Software Data Planes, *EuroP4 '21* [8]

Conclusion

- Adding state to the P4 data plane increases number of possible low-latency applications
  - Updatable Table Entries[1]
  - Add-On-Miss Insertions
- Add-on-Miss insertions enable cheap insertions w.r.t. latency

- Is this a step backwards in SDN ?
  - $\Rightarrow$ **No**, local and global state may work hand-in-hand
  - $\Rightarrow$ PNA proposal comes from the P4 community
  - $\Rightarrow$ PNA brings P4 to the NIC of the end-host where state is required anyways

[1] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, and G. Carle: High-Performance Match-Action Table Updates from within Programmable Software Data Planes, *EuroP4 '21* [8]

Conclusion

Contributions

- We implemented Add-on-Miss insertions for T4P4S[2]
- We discussed different optimization strategies w.r.t. to modelling performance
- Systematic analysis of PNA properties (i.e. updates and insertion costs)
- CPU cycle models and costs

On-the-fly Table Insertions on Programmable Software Data Planes

Manuel Simon, Sebastian Gallenmüller, and Georg Carle
Chair of Network Architectures and Services, Technical University of Munich, Germany
{simonm|gallenmu|carle}@net.in.tum.de

***Abstract*—Novel applications require a robust and reliable connection to provide the services for next-generation networks. The complex nature of these algorithms needs fast and efficient stateful processing. Using Software-defined Networking (SDN), new algorithms can be implemented into the network in a platform-independent way. The upcoming Portable NIC Architecture (PNA) for P4, a language to program data planes in SDN, allows inserting new table entries without controller interaction. Thus, it unleashes more performant and stateful applications without the overhead of the controller. We implement and evaluate these so-called 'add-on-miss' insertions introduced by the PNA for a P4 software target. In addition, we discuss the influence of latency and throughput optimizations on software packet processing systems. We determine the impact of these optimization strategies and which performance properties and costs can be measured with each. In our analysis, we model the costs of insertions based on an extensive baseline and compare them to table entry lookups and updates. We analyze the influence of the frequency of insertions and multi-core scenarios. Finally, we demonstrate that the approach scales for realistic scenarios.**

***Index Terms*—SDN, State Management, P4, Add-on-Miss**

I. Introduction

The upcoming 6G standard for communication networks will enable novel and complex applications, ensuring an ultra-low end-to-end latency as well as an ultra-low packet loss rate. Connections with these properties are essential for critical applications in domains such as transport, industry, and medicine. Optimized reliability methods are necessary to achieve these goals. An example of such an approach is hybrid automatic repeat request (HARQ). This algorithm increases the reliability of connections using forward error correction and repetition of non-acknowledged packets. Such complex algorithms must be distributed across different components in a network, either to the network interface card (NIC) or entirely to middleboxes to deal with demanding network applications.

P4 [1] is a platform-independent language to describe the data plane targeting high-performance, vendor-independent packet processing. With the upcoming Portable NIC Architecture (PNA) [2], P4 becomes a language to program both in-network switches and end-host applications. The latter is gaining attention due to efforts to bring P4 into the Linux Kernel [3]. Moreover, Intel announced that the SmartNIC E2000 will support the P4 language [4]. The capability of efficient state management becomes especially important when P4 programs are executed on the end of the communication path. Typical stateful scenarios include TCP flow tracking and the monitoring of connections.

The PNA enables stateful packet processing directly on the data plane. This new feature can speed up existing stateful P4 applications, such as IDS (e.g., *P4ID* [5]), stateful firewalls (e.g., *P4SF* [6]), or flow monitoring (e.g., *NetSeer* [7]). However, the statefulness of the P4 processing pipeline may introduce effects that are absent from the current generation of P4 devices, such as the impact on latency or jitter caused by state updates. The fundamental change in PNA requires a fundamental change to the measurement methodology used to investigate device behavior. Therefore, we establish a novel measurement methodology and apply it to a modified version of the P4 software target T4P4S [8]. This modified version supports *add-on-miss* insertions introduced by the PNA.

Our contributions can be summarized as follows: the definition of a measurement methodology focusing on the effects of stateful packet processing; the implementation of insertions in a software P4 target; the analysis of relevant performance indicators for PNA state updates; and the measurement and analysis for a comparison of costs for table entry lookups, updates, and insertions in a software P4 pipeline.

II. Background & Implementation

a) P4: P4 [1] provides a target-independent way of programming network forwarding devices, relying on compilers for different targets. This concept allows vendor-independent mechanisms and gives sovereignty to the network operator. So-called *externs* can utilize non P4-based extensions.

Several P4 targets exist, which can be classified as hardware and software targets. *Hardware targets* provide the highest performance in terms of throughput and latency. They usually follow a pipeline model with multiple stages executing specific subtasks of the program. Several packets are processed simultaneously but at different stages in the pipeline. This processing approach becomes important considering the consistency of state updates. *Software targets*, on the other side, provide the highest degree of flexibility. While their performance is lower, software targets run on commodity hardware and allow the easy integration of new functionality. Software targets typically follow the *run-to-completion* approach for packet processing. In this approach, different subtasks are handled by the same CPU core to avoid costly transfers of packets between different cores [9]. For our evaluation, we use T4P4S [8], which translates the P4 program to C code linked with DPDK [10], a userspace library for high-performance packet processing.

[2] Implementation available on GitHub `https://github.com/manuel-simon/t4p4s/tree/addonmiss`

Bibliography

[1] P4 portable nic architecture (pna), version 0.5.
accessed: 2023-03-10.

[2] 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.
Comput. Commun. Rev., 44(3):87–95, 2014.

[3] T. Caiazzi, M. Scazzariello, and M. Chiesa.
Millions of low-latency state insertions on ASIC switches.
PACMNET, 1(CoNEXT3), 2023.

[4] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle.
Moongen: A scriptable high-speed packet generator.
In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 275–287, New York, NY, USA, 2015. Association for Computing Machinery.

[5] S. Luo, H. Yu, and L. Vanbever.
Swing State: Consistent Updates for Stateful and Programmable Data Planes.
In *SOSR 2017*. ACM, 2017.

[6] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, et al.
Flowblaze: Stateful packet processing in hardware.
In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 531–548, 2019.

[7] M. Simon, S. Gallenmüller, and G. Carle.
Never Miss Twice - Add-On-Miss Table Updates in Software Data Planes.
In *KuVS Fachgespräch - Würzburg Workshop on Modeling, Analysis and Simulation of Next-Generation Communication Networks 2023 (WueWoWAS'23)*, page 5, 2023.

Bibliography

[8] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, and G. Carle.
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*, pages 102–108. ACM, 2021.

[9] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki.
T4p4s: A target-independent compiler for protocol-independent packet processors.
In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8. IEEE, 2018.

[10] L. Zeno, D. R. Ports, J. Nelson, D. Kim, S. Landau-Feibish, I. Keidar, A. Rinberg, A. Rashelbach, I. De-Paula, and M. Silberstein.
{SwiSh}: Distributed shared state abstractions for programmable switches.
In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 171–191, 2022.

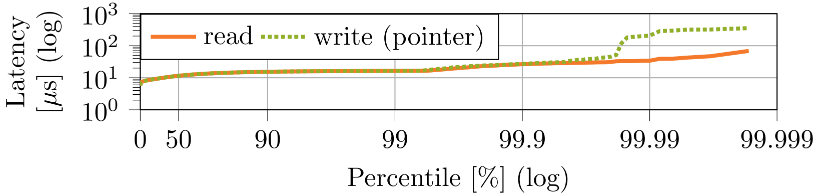
[11] Z. Zhou, M. He, W. Kellerer, A. Blenk, and K.-T. Foerster.
P4Update: fast and locally verifiable consistent network updates in the P4 data plane.
In *CoNEXT 2021*. ACM, 2021.

Additional slides

Previous Work – Changeable Table Entries

- In previous work[3], we implemented updatable table entries
  - `@__ref` annotation to declare parameters as references
- Replaced table architecture for synchronization
- Analyzed different synchronization and storage designs

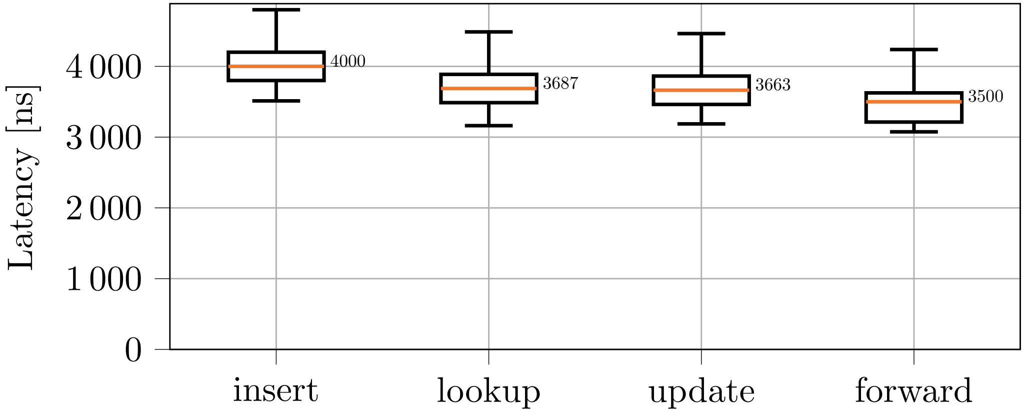
⇒ Table entry updates possible at line-rate



[3] M. Simon, H. Stubbe, D. Scholz, S. Gallenmüller, and G. Carle: High-Performance Match-Action Table Updates from within Programmable Software Data Planes, *EuroP4 '21* [8]

Previous Work – Changeable Table Entries

- Lookups and updates are comparable
- Insertions cost more



Previous Work – Changeable Table Entries

Modelled Costs/CPU-Cycles

	Δ l [ns]	Cycles
Insertion	500	1100
Lookup	187	411
Update	163	358
Resolution	12.5	28

Table 1: Operations

Insertion Rate	Δ l [ns]	Cycles
1	500	1100
10	587	1291
100	649	1428
1000	912	2006
10000	1337	3941
100000	2749	6048

Table 2: Insertions with different rates