Rust Barefoot Runtime (RBFRT): Fast Runtime Control for the Intel Tofino

Etienne Zink*, Moritz Flüchter*, Steffen Lindner*, Fabian Ihle*, Michael Menth*
*University of Tübingen, Chair of Communication Networks
Email: {etienne.zink, moritz.fluechter, steffen.lindner, fabian.ihle, menth}@uni-tuebingen.de

***Abstract*—Data plane programming enables the programmability of network devices with domain-specific programming languages, like P4. One commonly used P4-programmable hardware target is the Intel Tofino™ switching ASIC. The runtime behavior of an implemented P4 program on Tofino™ can be configured with shell scripts or a Python library from Barefoot provided with the Tofino™. Both are limited in their capabilities and usability. In this paper, we introduce the Rust Barefoot Runtime (RBFRT), a Rust-based control plane library. The RBFRT provides a fast and memory-safe interface to configure the Intel Tofino™. We showed that the RBFRT achieves a higher insertion rate for MAT entries and has a shorter response time compared to the Python library.**

***Index Terms*—SDN, Control Plane Library, Rust, Tofino™**

I. Introduction

P4 [1] is a domain-specific programming language for data plane programming currently used in academia and industry. It provides an architecture abstraction that allows developers to define forwarding logic independent of the hardware. The Intel Tofino™ switching ASIC is one commonly used P4-programmable hardware. It can be configured with shell scripts or a provided Python control plane library. Both approaches have significant shortcomings. The shell scripts can only write static table entries and cannot react to events. Python controllers react to events but are based on a slow programming language. Especially in productive environments, the control plane has to be fast and reliable.

Therefore, we present the Rust Barefoot Runtime (RBFRT), a Rust-based control plane library for the Intel Tofino™. Rust is an up-to-date, fast, and memory-safe programming language [2]. The RBFRT uses the same protocol for communication with the Tofino™ as the provided Python library. RBFRT facilitates the development of faster and more reliable control planes for the configuration of Tofino™. Moreover, RBFRT features batch configuration which is not supported by the Python library and leads to significantly larger insertion rates for table entries.

II. Configuring P4 Data Planes

P4 programs define the data plane, e.g., packet processing logic, for P4-programmable switches, called targets. Targets can be software-based, like the bmv2 [3], or hardware-based like the Intel Tofino™ switching ASIC [4]. In P4, the packet processing logic is implemented in tables, called match-action tables (MATs), and applied through packet header and metadata matching. Further, the target can define other P4 entities like registers. While the structure of MATs and P4 entities is defined in the data plane, they are configured by the control plane. A controller is a program that implements this configuration logic.

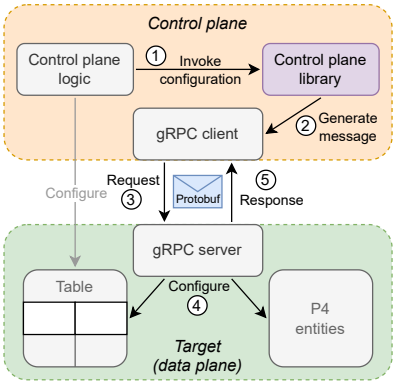


Fig. 1. *Configuration of a P4 target by a control plane. The control plane logic invokes the control plane library to generate a Protobuf message from the provided data. Then, the control plane library sends the Protobuf message with a gRCP client library. The target's gRPC server receives the message, and the respective P4 entities, like tables, are configured.*

Figure 1 shows how the control plane configures a target. The *control plane logic* defines how the data plane needs to be configured, e.g., what entries to insert. The communication between the control plane and target is implemented with Protobuf [5] messages and gRPC [6] requests. Therefore, the control plane implements a gRPC client and the target a gRPC server. The control plane logic invokes a *control plane library* ① to configure the target. The control plane library generates Protobuf messages ② and sends them to the target with the gRPC client ③. The target's gRPC server receives the request and configures MATs and other P4 entities accordingly ④. Afterward, the gRPC server responds with an acknowledgment ⑤ of whether the configuration was successful. The control plane can configure the target and read data from it. The gRPC server's response returns the data when reading from the target.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/3-1. The authors alone are responsible for the content of the paper.

The Protobuf messages are defined in specifications like the *P4Runtime* [7] or *Barefoot Runtime (BFRT)* [8]. We call these specifications *runtimes* in the following. The definition of the runtime messages is independent of the target and P4 program. To facilitate the controller development, *control plane libraries* are provided. Control plane libraries generate the Protobuf messages for a specific runtime based on provided data. Because of the target independence, these control plane libraries can be used to configure all targets implementing the respective runtime.

The Intel Tofino™ is configured through BFRT messages. Barefoot implemented a Python-based control plane library to facilitate the configuration of Tofino™. The library's usability is limited because of its lack of documentation and the effort required to use it. Further, the library is not maintained anymore. APS Networks introduced its own Python control plane library for better documentation and ease of use. Further, this library is no longer developed, little in use, and the GitHub repository [9] is not well documented. Therefore, we do not consider the library from APS Networks further in this paper. In addition, both libraries are Python-based, limiting the resulting controller's speed. Another option to configure the Tofino™ is to execute commands directly in its shell or to write and execute simple scripts. Both require much manual effort, are limited in extensibility, and do not react to events. Below the line, the current tools to configure the Intel Tofino™ are all limited in usability and resulting controller's speed.

III. The Rust Barefoot Runtime

We present the Rust Barefoot Runtime (RBFRT), which is a Rust-based control plane library implementing the BFRT and is available on GitHub [10]. It is developed to overcome the limitations of the currently available tools to configure the Intel Tofino™. RBFRT provides new functions that simplify the configuration process compared to the existing Python library. The programming language Rust is used because of its fast execution, memory efficiency and safety, and static type system. As a result, RBFRT facilitates the development of faster and more reliable controllers. Controller implemented with the RBFRT are already used by P4TG in [11], [12], an MNA prototype in [13], and an extension to BIER-TE in [14].

With RBFRT, one or multiple configuration entries are passed to a function call and then sent within a single gRPC request. We call this *batch configuration*. In contrast, the existing Python libraries from Barefoot and APS support only a single configuration entry per gRPC request. Batch configuration reduces overhead in the presence of many timely related configuration entries. The objective is a higher configuration rate, leading to a faster control of the target.

IV. Evaluation

In this section, two control plane libraries are evaluated: the Barefoot Python library provided with the Intel Tofino™ and the Rust Barefoot Runtime. First, we describe the testbed and the experiment. Afterward, we analyze the results and summarize improvements.

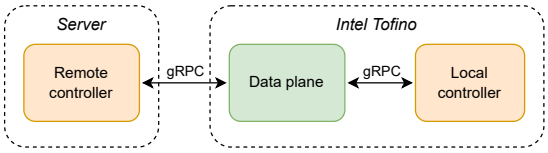


Fig. 2. *Experiment setup with a remote and a local controller. Both communicate via gRPC with the data plane API.*

A. Testbed

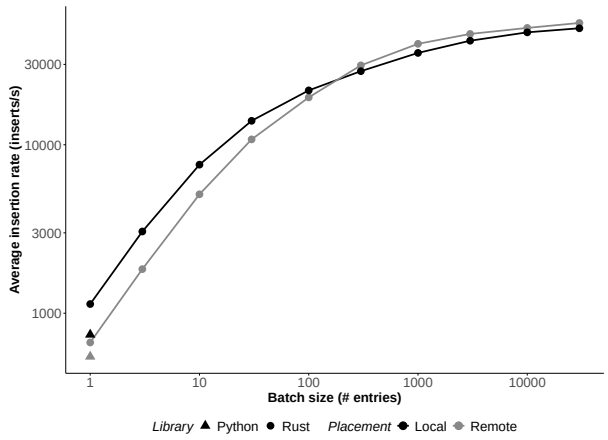
Two control paradigms are distinguished: local and remote control. Figure 2 illustrates the testbed for both control paradigms. The local controller runs directly on the CPU of the Tofino™. The remote controller runs on a server with Ubuntu 22.04.1, an Intel(R) Xeon(R) Gold 6134 CPU @ 3.20 GHz, four cores, 16 GB of RAM, and a Mellanox(R) ConnectX-5 NIC. It is connected to the Tofino™ with a single $100\,\mathrm{Gb/s}$ link. The local controller has limited performance, while the remote controller has a larger delay for the gRPC requests.

B. Insertion of MAT Entries

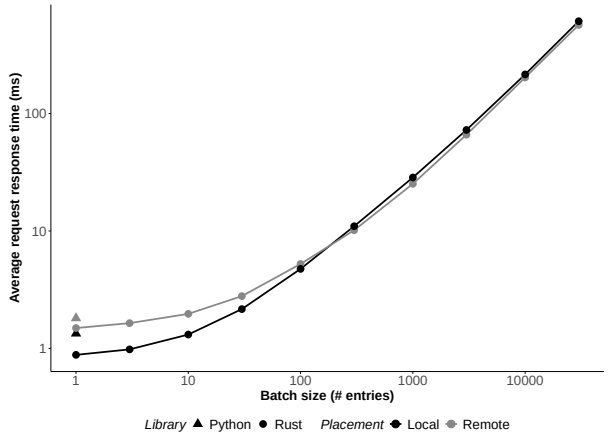
This experiment evaluates the insertion rate and response time for inserting MAT entries with different batch sizes. The insertion rate indicates the number of entries a single controller can insert into a MAT over time. The response time includes the duration for a controller to create and transmit a request until it receives a successful response from the target. The data plane on the Intel Tofino™ runs a simple, self-programmed firewall in the experiment. It consists of one MAT that performs an exact match on the source and destination IP address of an incoming packet. In each evaluation run, the controller inserts $3 \cdot 10^4$ entries into the MAT with different batch sizes, i.e., entries per request. The batch sizes are $\{a \cdot 10^i \mid a \in \{1,3\} \wedge i \in \{0,1,2,3,4\}\}$. Single-entry configuration corresponds to a batch size of one, and a batch size of $3 \cdot 10^4$ configures all entries in a single request. Smaller batch sizes are relevant for productive environments while larger batch sizes are relevant to review the scalability of the control plane library. The controller creates and sends a batch of configuration entries to the target and waits until the configuration is acknowledged. Then the next batch is created and sent. The cumulative time to insert all entries, i.e., creation of the first request until the response of the last request, is measured. The insertion rate and response time are calculated based on this cumulative time. For any batch size, 100 runs were conducted. Confidence intervals with an overall significance level of 1% were calculated. As the half-widths of all confidence intervals are smaller than 1% of the measured average, we omit them in the following figures.

C. Results of MAT Insertions

Figure 3(a) shows the average insertion rate with different batch sizes. The local and remote Python controller with single-entry requests achieve a rate of $746\,\mathrm{entries/s}$ and $553\,\mathrm{entries/s}$, respectively. The RBFRT controller with single-entry insertion achieves $1134\,\mathrm{entries/s}$ and $671\,\mathrm{entries/s}$,



(a) *Average insertion rate.*



(b) *Average request response time.*

Fig. 3. *Evaluation of inserting* $3 \cdot 10^4$ *entries into the MAT of the firewall data plane with different batch sizes, control plane paradigms, and control plane libraries.*

which is an increase of 48% and 21% compared to Python. Further, the insertion rate of the RBFRT increases significantly with the batch size. This increase results from a lower overhead per entry in larger batches. With a batch size of $3 \cdot 10^4$ entries, the insertion rate reaches $52\,832\,\mathrm{entries/s}$ for the local and $49\,136\,\mathrm{entries/s}$ for the remote controller. Thus, when configuring the Intel Tofino™ with a single controller, the insertion rate is limited by controller performance and network delay, not by the Tofino™ itself. Therefore, larger batch sizes lead to a more efficient insertion of MAT entries.

We further observe in Figure 3(a) that the insertion rate with small batches is larger for a local than for a remote controller while it is vice-versa for large batches. This can be explained as follows. The remote controller runs on more powerful hardware, therefore its insertion rate is better for large batches. In contrast, small batches induce more network delay per entry, so a better insertion rate is obtained with the lower network delay of the local controller.

Figure 3(b) shows the average response time for requests with different batch sizes. Like the insertion rate also the response time increases with larger batch size. The average response time for the local and remote Python controller is $1.34\,\mathrm{ms}$ and $1.8\,\mathrm{ms}$, respectively. The corresponding values for the Rust controller (for a single entry) are $0.88\,\mathrm{ms}$ and $1.49\,\mathrm{ms}$, which is a reduction of 34% and 17% compared to Python. Therefore, an appropriate batch size is recommended so that the insertion rate can be high while the response time for a single request is still low enough. Given an acceptable response time, a maximum batch size can be derived from Figure 3(b). Figure 3(a) can be used to deduce the achievable insertion rate for that batch size.

V. Conclusion

In this paper, we introduced RBFRT, a Rust-based control plane library for the Intel Tofino™. Due to the programming language Rust, the library is fast and memory-safe. Our experiments showed that RBFRT leads to shorter response times for MAT entry insertion than the Barefoot Python library, which is shipped with Tofino™. Moreover, RBFRT supports batch configurations, which greatly increase achievable insertion rates compared to single-entry insertions offered by the Python library. While RBFRT also supports port configuration and data reading, performance regarding these operations still needs to be evaluated. Further, for better comparison, batch configurations may be implemented and then evaluated for Barefoot's Python library.

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," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, 2014.

[2] The Rust Foundation, "Rust Programming Language." https://www.rust-lang.org/. *Last accessed on 13.02.2025.*

[3] P4 Language Consortium, "GitHub: Behavioural Model (bmv2)." https://github.com/p4lang/behavioral-model. *Last accessed on 13.02.2025.*

[4] Intel, "Intel Tofino." https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html. *Last accessed on 13.02.2025.*

[5] Google, "Protocol Buffers Documentation." https://protobuf.dev. *Last accessed on 13.02.2025.*

[6] The Linux Foundation, "gRPC - A High Performance, Open Source Universal RPC Framework." https://grpc.io. *Last accessed on 10.01.2025.*

[7] The P4.org API Working Group, "P4Runtime Specification." https://p4.org/p4-spec/docs/p4runtime-spec-working-draft-html-version.html. *Last accessed on 13.02.2025.*

[8] Intel, "GitHub: Open-Tofino." https://github.com/barefootnetworks/Open-Tofino. *Last accessed on 13.02.2025.*

[9] APS Networks, "GitHub: Barefoot Runtime Helper (bfrt-helper)." https://github.com/APS-Networks/bfrt-helper. *Last accessed on 13.02.2025.*

[10] S. Lindner and F. Ihle, "GitHub: Rust BF Runtime Interface (rbfrt)." https://github.com/uni-tue-kn/rbfrt. *Last accessed on 13.02.2025.*

[11] S. Lindner, M. Häberle, and M. Menth, "P4TG: 1 Tb/s Traffic Generation for Ethernet/IP Networks," *IEEE Access*, vol. 11, pp. 17525–17535, Feb. 2023.

[12] F. Ihle, E. Zink, S. Lindner, and M. Menth, "Enhancements to P4TG: Performance, Protocols, and Automation," 2025. arXiv Preprint.

[13] F. Ihle and M. Menth, "MPLS Network Actions: Technological Overview and P4-Based Implementation on a High-Speed Switching ASIC," 2024. arXiv Preprint.

[14] M. Flüchter, S. Lindner, F. Ihle, T. Eckert, and M. Menth, "Extensions to BIER Tree Engineering (BIER-TE) for Large Multicast Domains and 1:1 Protection: Concept, Implementation and Performance," 2024. arXiv Preprint.