Towards Multiple Pipelines Network Emulation with P7

Fabricio E Rodriguez Cesen*, Francisco Germano Vogt*, Ariel Goes De Castro†,
Christian Esteve Rothenberg*
*University of Campinas (UNICAMP), Brazil
†Federal University of Pampa (UNIPAMPA), Brazil

***Abstract*—Network emulation traditionally relies on software-based solutions. While extremely useful in many scenarios, it suffers from performance fidelity and inherent scalability constraints. With the advent of P4 and programmable switches like Tofino, new opportunities for hardware-based network emulation are emerging. P7 (P4 Programmable Patch Panel) offers a solution for high-fidelity 100G traffic network emulation, including different link characteristics such as latency, jitter, packet loss, and bandwidth, as well as the ability to define custom topologies. However, it currently lacks support for custom P4 code in emulated devices. This is where multiple pipelines network emulation comes in. In this demonstration, we show how to emulate a topology using P7 and incorporate custom P4 code into each emulated node. We allocate a dedicated pipe for user-defined P4 code and allow users to configure tables for each node separately.**

I. Introduction

Network emulation has long been a valuable tool for testing and evaluating network configurations and protocols in a safe and controlled environment. However, existing software-based solutions suffer from limitations in performance fidelity and scalability, which can affect the accuracy and validity of test results. As a result, there is a growing demand for hardware-based network emulators that can provide high-fidelity, high-performance testing capabilities while also offering the flexibility and customizability of software-based solutions.

The availability of programmable switches like Tofino and the programming language P4 has opened up new opportunities for hardware-based network emulation with enhanced flexibility and performance. One such solution is P7 (P4 Programmable Patch Panel) [1], which offers high-fidelity 100G traffic network emulation, including various link characteristics such as latency, jitter, packet loss, and bandwidth, as well as the option to customize network topologies.

However, one limitation of P7 is that each emulated switch is treated as a simple forwarding node without the ability to instantiate a custom P4 code, which limits experimentation in dataplane programmability. To address this limitation, in this demo, we propose integrating user-defined P4 code and P7, enabling users to test their own P4 codes in an emulated topology with all the characteristics of P7[1].

This work was supported by the Innovation Center, Ericsson S.A., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS. This study was partially funded by CAPES, Brazil - Finance Code 001.

II. Background and Related Work

A. P7 (P4 Programmable Patch Panel)

P7 is a network emulator tool for high-fidelity network experimentation on a single P4 switch. With a friendly environment, the users can define different topologies following a script similar to Mininet [2]. P7 allows users to instantiate network elements (i.e., switches), background traffic, and link characteristics. Some of the principal link metrics included in P7 are latency, jitter, packet loss, and bandwidth.

Bandwidth: Define the bandwidth limit (in Mbps) for every single link on the topology. The bandwidth limitation is performed using a port shaping definition. In Tofino Native Architecture (TNA)[2], the physical ports and the recirculation ports can be shaped.

Latency: Define per-link latency in milliseconds. P7 uses recirculation and timestamps to achieve latency by keeping the packet recirculating in the same pipe until it reaches the desired time. A custom header is used to store the original timestamp and verify it at each recirculation.

Packet loss: Define the possibility of dropping a packet in a link. P7 has a random function that discards the packet according to the specified loss percentage. Using a random number, if it is below the threshold, the packet is dropped.

Jitter: Define per-link jitter and a probability (per packet) that the jitter will be applied. P7 uses a random function to increase or decrease latency and apply the variation.

To emulate a topology, P7 adds a custom header to the packet that contains information about the current emulated link, the switch ID, and the original timestamp. Using this information, P7 applies the link characteristics and forwards the packet to the corresponding link or switch. In addition to the header, P7 utilizes recirculations in the same pipe to create the logic determining how the packet transitions from one link to another or enters a switch.

An overview of the P7 architecture is shown in Figure 1. The architecture includes the addition of a custom P4 code in P7 and the corresponding running and compilation processes

[1]Public available at: https://github.com/intrig-unicamp/p7
[2]https://github.com/barefootnetworks/Open-Tofino

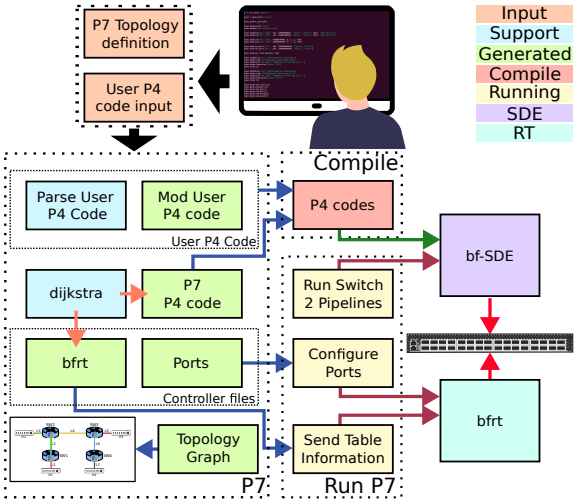

Fig. 1. P7 architecture.

that need to be followed. P7 utilizes the information defined by the user in the main topology script to automatically generate the necessary files required for running P7 on Tofino hardware. The files generated by P7 include the P7 P4 code, tables, and port information. The user is responsible for compiling the P4 code and running it with the corresponding table information sent by Barefoot Runtime (bfrt).

B. Related Work

Recent efforts in P4-based hardware have been utilized to enhance programmability and parallelization by incorporating multiple pipelines. Some works have explored multiple pipelines but are limited in emulating topologies across their solution. Other efforts focused on software-based emulation, including programmable devices, but resources limit them.

P4i/o [3] combines high-level intent policies with a user-friendly language and the ability to install and remove P4 code to the underlying hardware. Similarly, eBPFlow [4] leverages P4 externs to encapsulate user-defined packet processing logic and provides a hierarchical pipeline structure for sharing common processing logic across multiple pipelines. In contrast, Flightplan [5] splits the packet processing pipeline into pieces across multiple devices. Meissa [6] is built on top of Mininet, and Sailfish [7] allows customizing P4 codes and offloading some workloads typically handled by general-purpose servers.

III. Multiple Pipelines Support

The Intel Tofino structure employs a multiple-pipe design (e.g., 2 or 4 pipes). Each internal pipe is identical in structure and contains 16 100G Ethernet ports. When a packet arrives, it enters through the Ingress Parser, where the headers present in the packet are identified. The Ingress Control is responsible for processing the packet and applying the match+action tables. After Ingress processing, the packet enters the Traffic Manager (TM). The TM adds the packet into a packet buffer and enqueues the packet to the desired port's pipe. Finally, the packet passes through Egress processing and is forwarded to the physical interface.

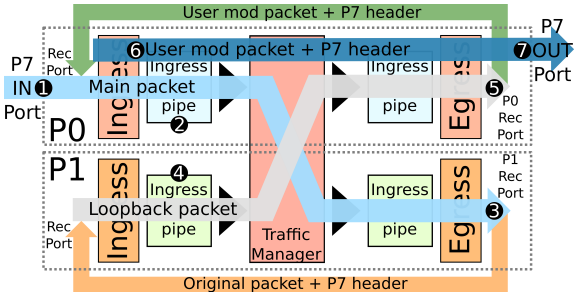

Fig. 2. P7 multiple pipelines approach.

TNA allows us to define and map how the P4 program will be distributed in different pipes. The default configuration when running a single P4 replicates the program along all the available pipes. On the other hand, we can set a custom configuration of pipes and add different programs to the pipes.

With P7, to emulate a custom P4 code in each internal switch, we leverage the possibility of setting a custom pipe distribution model. We propose a solution where a dedicated pipe runs the P7 P4 code, and a separate pipe runs the user-defined P4 code (See Figure 2).

To emulate different devices with user-defined P4 code and the generated topology, we use the TNA recirculation feature. First, the packet enters into P0 ❶ and is processed by the P7 P4 code ❷. Packets to be processed by an emulated switch are sent from the P7 pipe (P0) to the pipe (P1) ❸ running the user-defined P4 code. In TNA, there is a specific recirculation port to forward the packet to a particular pipe. Finally, packets are sent back to P0 to continue P7 processing. This process is repeated for all emulated switches.

To align with P7, the user's P4 code needs to be adapted. We developed a parser that identifies the principal parts of the P4 code and performs modifications. The modifications include the addition of the P7 header, a switch identifier, and a recirculation port. These modifications are necessary to align the user's code with P7 and enable the proposed custom pipe distribution model.

When adding the P7 header, we include the P7 parser processing before the user's parser ❹. A switch identifier is added in all the tables as a Key to match the right switch in the emulation. Finally, when the processing of the packet ends, we modify the forwarding port to the corresponding recirculation port to send the packet back to P0 ❺.

The packet continues with P7 processing in P0 ❻. When one of the metrics needs to be applied in a link, P7 executes the corresponding process. It takes the necessary actions, such as recirculating the packet, limiting the bandwidth, or applying packet loss. If P7 needs to recirculate a packet due to a metric, it must be sent to the same pipe's recirculation port to keep the packet in P0. Finally, the packet is forwarded to the corresponding out port ❼.

To run P7 and the custom P4 code (the integrated processing can be seen in Figure 1), it is necessary to compile both P4 codes and define the corresponding pipe distribution. This can be done by setting a custom target configuration file that includes the compiled files and the assigned pipes. In

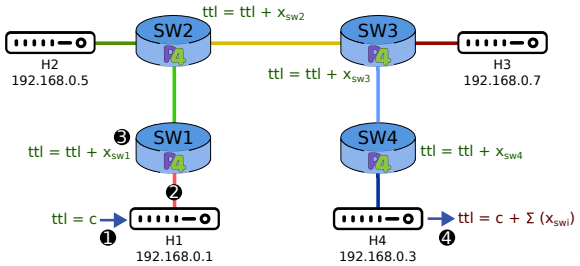

Fig. 3. Demo Topology

addition, we need to take care while setting the recirculation and physical ports since it depends on the specific target.

IV. Demo

To validate the proposed multiple pipeline implementation, we use a custom P4 code that contains different mathematical operations that are applied to the IP field ttl. These operations are defined by a P4 table that contains the operation and its value. This P4 code will perform a specific operation based on the destination IP of the packet and the information filled in a table. To confirm that the packet is passing by all the corresponding nodes in the emulated topology, we can set different values for the operation. We can summarize the end-to-end operation by the equation 1.

$$ttl = c + \sum_{i=1}^{n} (x_{SWi}) \tag{1}$$

Where c is the initial value of the ttl, n is the number of switches, and x_{SWi} is the value of the operation.

Following the topology described in Figure 3. If we send a packet from H1 to H4, the packet will start in ❶ with a defined ttl value c. The packet will enter into the Tofino switch and start the P7 processing. The next step ❷ is in the link between H1 and SW1. The metrics associated with this link will be applied in the P7 pipe (P0). Then, in ❸, the packet is processed in the user-defined P4 code pipe (P1). In this step, the custom P4 will modify the ttl value accordingly to the table information of this switch (SW1) filled in the P7 main script. In this switch, the operation will be $ttl = ttl + x_{SW1}$. After performing all the processing in the custom P4 code, the packet will continue following the topology and applying the process of each link and switch. Finally, when the packet is forwarded to the destination ❹, the result of all the operations will be present in the ttl value of the packet.

During the demo. We will present different use cases, including topologies with link metrics and custom P4 codes. We will run P7 remotely in physical Tofino Hardware connected with different physical servers. The main items to be presented are:

- The calculator use case with various operation values and different source and destination hosts. Also, the results can be confirmed by a local script (software-based) that calculates the operations accordingly to the scenario.
- Attendees will be asked to choose a network topology with different link metrics (e.g., latency, jitter, packet loss, background traffic, bandwidth) and a custom P4 code (e.g., simple l3 forwarding with TTL update). They will see P7 in action, including the auto-generation of files and the complete network environment emulation.
- Real-time visualization of network traffic will contribute to validating on-the-fly performance of the link metric and the emulation capabilities.

V. Conclusions and Future Work

This demo shows the integration of P7 with multiple pipelines, contributing to experimental platforms that support data plane programmability. We presented a solution that allows the user to create a traditional networking topology and advanced programmable networking scenario defining a custom P4 program to run in each node. We offer a user-friendly environment, a cost-effective 100G network emulator, and a hardware-based solution for research and teaching.

P7 is a high-fidelity, programmable testbed that offers repeatable and reproducible research opportunities. Researchers can share P7 topology files and custom P4 codes that can be compiled and deployed, resulting in consistent output regardless of the location Moreover, P7's capability to emulate line-rate networks with custom P4 codes opens up new avenues for experimentation and testing.

We plan to enhance the support setting different custom P4 codes in the instantiated switches. We can send the packet to the corresponding pipe using the same idea of the recirculation port. In addition, we will add support for registers in the custom P4 code since they will be shared across all the emulated devices. Furthermore, we want to formalize the performance & scalability boundaries (e.g., throughput) depending on the memory, buffers, and stages available in the Tofino target.

Future work includes adding In-band Network Telemetry (INT) features for fine-grained statistics of the emulated network (e.g., queue occupancy, device status). INT is an important feature to analyze effectively what is happening inside each of the emulated switches.

Another area of future work is to embed P7 (e.g., adding P7 hardware-in-the-loop or compiling into P4 SmartNICs) into larger testbeds such as NSF Fabric and disaggregated network initiatives (e.g., OpenRAN Brasil) to enrich their experimental toolboxes with tailored line-rate network emulation capabilities.

References

[1] F. Rodriguez *et al.*, "P4 Programmable Patch Panel (P7): An Instant 100g Emulated Network on Your Tofino-Based Pizza Box," in *SIGCOMM '22 Poster and Demo Sessions*. New York, NY, USA: ACM, 2022, p. 4–6.
[2] B. Lantz *et al.*, "A network in a laptop: Rapid prototyping for software-defined networks," in *SIGCOMM*. NY, USA: ACM, 2010.
[3] M. Riftadi and F. Kuipers, "P4i/o: Intent-based networking with p4," in *IEEE NetSoft*, 2019, pp. 438–443.
[4] R. D. G. Pacífico *et al.*, "Application layer packet classifier in hardware," in *IFIP/IEEE IM*, 2021, pp. 515–522.
[5] N. Sultana *et al.*, "Flightplan: Dataplane disaggregation and placement for p4 programs," in *NSDI*, 2021.
[6] N. Zheng *et al.*, "Meissa: Scalable network testing for programmable data planes," in *SIGCOMM*. NY: ACM, 2022, p. 350–364.
[7] T. Pan *et al.*, "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *SIGCOMM*. NY: ACM, 2021.