Comparative Analysis and Evaluation of P4-Based Network Emulation Testing Environments

Luke Waind
ldwaind@uark.edu
University of Arkansas
Fayetteville, AR, USA

Gong Chen
gchen31@hawk.iit.edu
Illinois Institute of Technology
Chicago, IL, USA

Zheng Hu, Dong Jin
{zhenghu,dongjin}@uark.edu
University of Arkansas
Fayetteville, AR, USA

ABSTRACT

P4, an emerging technology enabling flexible and programmable data plane processing in network devices, has garnered significant attention for revolutionizing in-network operations. Validating P4 programs requires specially designed testing environments to emulate network functionality in hosts and programmable switches. However, the choice of testbed involves weighing various pros and cons. In this paper, we assess four commonly used testbeds, i.e., container-based network emulation on the virtual machine, native Linux, and native Linux with Virtual Time, as well as physical hardware, to provide comparisons and offer guidelines for developers in selecting the most suitable P4 testbed for their needs.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation**; • **Networks** → *Network performance evaluation*; • **Computer systems organization** → Parallel architectures.

ACM Reference Format:
Luke Waind, Gong Chen, and Zheng Hu, Dong Jin. 2024. Comparative Analysis and Evaluation of P4-Based Network Emulation Testing Environments. In *38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS '24), June 24–26, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3615979.3662157

1 INTRODUCTION

Programming Protocol-independent Packet Processors (P4) is a rapidly growing technology seeing widespread adoption in modern networks [5]. Offering high-speed programmability in network data planes, P4 enables swift adaptation to networking challenges without the need for vendors to develop custom hardware solutions. The P4 language provides full customizable control over how a packet is processed.

To mitigate potential complexity and semantic errors in P4 programs, developers require a safe environment for testing and debugging. Running untested code in a real network system could have disastrous consequences. An ideal testbed must accurately emulate complete network functionality to ensure accurate testing and benchmarking.

We aim to assess P4 testing environments, providing insights into their setup processes and comparing their pros and cons by evaluating factors such as functional and temporal fidelity against costs, setup complexity, and developer flexibility. We focus on four prevalent environments and conduct two experiments (linear and throughput) to measure the fidelity of each environment.

SIGSIM PADS '24, June 24–26, 2024, Atlanta, GA, USA

ACM ISBN 979-8-4007-0363-8/24/06
https://doi.org/10.1145/3615979.3662157

2 ENVIRONMENTS FOR P4 TESTBEDS

This section provides detailed descriptions of each environment we evaluated and a comparison of their pros and cons (Table 1).

Virtual Machine Environment is contained in a workstation running VirtualBox [7], set up using resources available in the P4 tutorials repository on GitHub [1]. We maintained default settings for our experiments: 2 CPU cores with 100% utilization and 2 GB of RAM.

Native Linux Environment operates directly on a host machine, utilizing its full computational resources instead of being confined to a virtualized environment. We used the same GitHub repository as the Virtual Machine Environment [1]. This environment runs on a machine with an Intel Core i7-6700 CPU, featuring 4 cores, and 2 threads per core, along with 16 GB of RAM.

Native Linux Environment with VT is similar to the Native Linux Environment but integrates Virtual Time (VT) technology [6]. VT modifies the Linux kernel by dilating the time returned by time-keeping kernel functions by a constant factor called the Time Dilation Factor (TDF). This dilation causes network processing to reference a virtual clock instead of wall-clock time. We adjust the TDF to mitigate performance loss caused by CPU overload.

Physical Switch Environment utilizes a custom-designed testbed comprising a physical programmable switch. The programmable switch is an Aurora 610 [3]. We used groups of ports on a single programmable switch to emulate the behavior of multiple switches. While the P4 programs for the software environments were written for Behavioral Model Version 2 (BMv2) [2], the P4 programs for this environment were written for Tofino Native Architecture.

3 EXPERIMENTATION

The two experiments we ran used simple topologies that isolate the environments' delays and bandwidths. Realistic scenarios have more complex topologies to which these results can be applied.

3.1 Linear Experiment

The linear experiment evaluates testbed performance across several links, involving two hosts connected via a linear series of programmable switches. We measured average round-trip time (RTT) using the Linux ping tool and the TCP throughput using a TCP connection with iperf [4].

Table 1: Testing environments comparison.

Environment	Functional Fidelity	Temporal Fidelity Under Low Workload	Temporal Fidelity Under High Workload	OS Independence	Cost	Ease to Install	Flexibility
BMv2+VM	High	Medium	Low	Yes	Low	Easy	High
BMv2+Native Linux	High	Medium	Low	No	Low	Easy	High
BMv2+Native Linux+VT	High	High	High	No	Low	Medium	High
Physical Testbed	Highest	High	High	No	High	Hard	Low

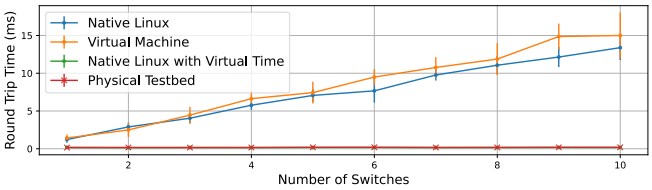

Figure 1: Average RTT through a linear series of switches.

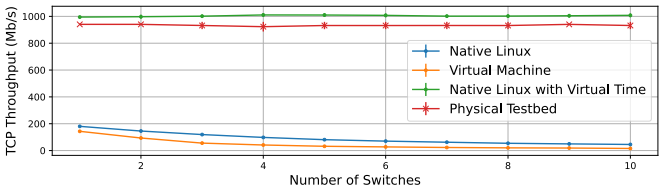

Figure 2: Average bandwidth through a linear chain of switches.

Figure 1 demonstrates that the P4 hardware switch and VT stand out with consistently low delays. In contrast, Native Linux and Virtual Machine environments show anticipated linear delay increases (1.217 and 1.4245 ms per BMv2 switch respectively). This performance gap is due to considerable delays in packet processing and forwarding inherent in P4 software switches. Figure 2 shows that the native Linux and VM environments experience a degradation in performance as switches are added due to computational resource constraints. With 10 switches, these environments operate at less than 30% of the bandwidth they could handle with one switch. VT-enabled Linux is unaffected as we adjust its time dilation to maintain performance; hardware performance remains stable as it can handle higher data volumes. Both environments maintained speeds near 1000 Mbps, the hosts' bandwidth cap.

3.2 Single-Switch Throughput Experiment

The single-switch throughput experiment assesses each environment's ability to achieve specific throughput. Using iperf [4], we establish a TCP flow between H1 and H2, varying the flow's bandwidth and measuring the achieved throughput.

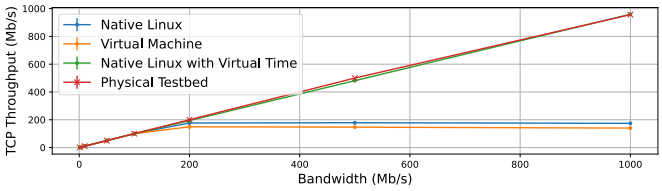

Figure 3: Bandwidth achieved through a single switch

In Figure 3, the throughput for native Linux and virtual machine environments stabilizes at approximately 175 Mbps and 145 Mbps, respectively. Conversely, the other two environments maintain speeds of at least 1000 Mbps. However, the physical hardware environment's maximum throughput is capped at 1000 Mbps, limited by host capabilities.

3.3 Discussions

In Figure 1, the virtual machine and native Linux environments exhibit low temporal fidelity, contrasting with VT and physical hardware setups. These findings caution against using the former for delay-sensitive applications. However, if temporal fidelity is not critical and functional fidelity suffices, these environments offer advantages in cost, flexibility, and installation simplicity.

The native Linux and virtual machine environments exhibit limited bandwidth capacities (Figure 3). This constraint becomes evident in more complex topologies, where bandwidth decreases with the addition of programmable switches (Figure 2). Given these limitations, researchers should assess whether these environments meet their experimental requirements by considering the desired bandwidth and network topology size.

VT should be the next candidate to consider if temporal fidelity is required. It is more difficult to install because it requires root-level access to a machine running a supported Linux kernel version. There is a potential concern using VT if the workload becomes too heavy, as the TDF directly influences runtime. Our experiments reached a maximum TDF of 79.6, highlighting the potential impracticality of higher TDF values due to experimental runtime.

The physical hardware environment provides superior functional and temporal fidelity but comes with a substantial cost and limited flexibility. We recommend using a physical hardware environment when utmost fidelity is essential or when the experiment's workload surpasses all available software-based options.

ACKNOWLEDGMENTS

The authors are grateful for the support of the National Science Foundation under Grant CNS-2247721, CNS-2034870, and EEC-2113903.

REFERENCES

[1] 2015. p4lang/tutorials. https://github.com/p4lang/tutorials
[2] 2023. BMv2. https://github.com/p4lang/behavioral-model
[3] 2024. Aurora 610. https://bm-switch.com/product/48x25g-8x100g-netberg-aurora-610-intel-tofino-sonic-ready/
[4] 2024. iperf. https://iperf.fr/
[5] 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. *SIGCOMM Comput. Commun. Rev.* 44, 3 (jul 2014), 87–95. https://doi.org/10.1145/2656877.2656890
[6] Gong Chen, Zheng Hu, and Dong Jin. 2022. Integrating I/O Time to Virtual Time System for High Fidelity Container-based Network Emulation. In *Proceedings of the 2022 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation.*
[7] Jon Watson. 2008. VirtualBox: Bits and Bytes Masquerading as Machines. *Linux J.* (2008).