Towards Time-Sensitive Networking Traffic Generation with PIPO-TG

Filipo Gabert Costa[1], **Francisco Germano Vogt**[1], **Fabricio Rodriguez**[1], **Marcelo Luizelli**[2], **Christian Esteve Rothenberg**[1]

[1] University of Campinas (UNICAMP)
[2]Federal University of Pampa (Unipampa)

f289951@g.unicamp.br, f234632@dac.unicamp.br, frodri@dca.fee.unicamp.br, marceloluizelli@unipampa.edu.br ,chesteve@unicamp.br

Abstract. *Realistic traffic simulation is essential for evaluating network performance, security, and efficiency. However, modern network environments demand testers with high performance, scalability, precision, flexibility, and cost-effectiveness, which current solutions struggle to achieve simultaneously. PIPO-TG has been proposed to overcome these constraints as a Tofino-based traffic generator tailored for high-performance parametrizable traffic experiments. In this work, we extend PIPO-TG capabilities to support an even more challenging scenario, a delay distribution on a nanosecond scale. We reproduce time-sensitive networking (TSN) delay traces measured in a commercial TSN bridge. We demonstrate the flexibility, accuracy, and performance of PIPO-TG, making it an ideal tool for network testing in network research and experimentation.*

1. Introduction

Proper traffic generation is key to evaluating network performance, efficiency, and security in both research and technical applications. The ability to generate controlled and realistic network traffic has become crucial, particularly with recent advancements in programmable networks [Liu et al. 2022]. These tools enable the controlled generation of realistic traffic and the emulation of real-world scenarios, providing valuable insights into network system behavior. Moreover, traffic generation plays a pivotal role in capacity planning, enabling the prediction and scaling of network capacity to meet current and future traffic demands, thereby ensuring consistent network performance. Lastly, the ability to generate controlled and realistic traffic supports scenario-based simulations, facilitating decision-making and network architecture planning across a range of environments.

Software-based traffic generators [Emmerich et al. 2015, TRex 2023] offer valuable network evaluation and testing alternatives. These platforms, often developed using frameworks like Data Plane Development Kit (DPDK) and netmap, are known for their cost-effectiveness and flexibility. Moreover, software-based traffic generators are often open-source and freely available, making them accessible to a wide range of users. However, these solutions have limitations, such as their accuracy, precision, and scalability, reaching a rate of around 10 Gbps per server core, therefore having difficulties in reaching a throughput in the range of Tbps like hardware-based traffic generators. On the other hand, despite its high performance and accuracy, hardware-based traffic generators [Zhou et al. 2019, Lindner et al. 2023] still suffer from their flexibility and inability

to replicate real scenarios (e.g., stateful connections), in addition to some not being available to the community [Zhou et al. 2019].

To address these limitations, we recently proposed PIPO-TG [Costa et al. 2024], a Tofino-based traffic generator designed for high-performance parametrizable traffic experiments. PIPO-TG is capable of generating traffic replicating various scenarios, such as congestion, microbursts, and distributed denial-of-service (DDoS) attacks. In this demonstration, we present new capabilities of PIPO-TG to support even more challenging scenarios. We extend PIPO-TG to reproduce real-world delay measurements from time-sensitive networks (TSN). We use the real delay traces shared by the DETERMINISTIC-6G project [DETERMINISTIC6G 2023]. The traces contain delay variations measured in a real TSN scenario in the range of nanoseconds. Therefore, a fine granularity tool for their reproduction and experimentation is required. Our results show that hardware-based PIPO-TG is able to reproduce these extremely low delay proposed scenarios, reaching delays in the range of $500ns$, significantly contributing to the replication of real scenarios through the generation of synthetic traffic. During the demonstration, participants will be able to observe the application of PIPO-TG not only to these use cases but also to alternative ones. Additionally, they will be able to visualize the results through our new graphical interface, which has been developed especially for demonstration purposes.

2. Related work

This section discusses related works that focus principally on traffic generators. Existing endeavors are implemented either software-based or hardware-based.

Traditional open-source software tools such as Iperf3 [Mortimer 2018], Netperf [Jones 1996], Netcat [*Hobbit* 1995], and Httperf [Mosberger and Jin 1998] are commonly utilized for bandwidth measurement, offering support for various protocols and modes for traffic generation and measurement. However, their accuracy is often unreliable, posing challenges in replicating tests across different scenarios. Despite attempts in the literature to address and test these issues [Kundel et al. 2020], the results have fallen short of expectations [Botta et al. 2010].

In more specific solutions like TRex [TRex 2023], an open-source traffic generator, operations encompass both stateless and stateful modes, simplifying traffic generation across layers L3 to L7. TRex achieves remarkable speeds of up to 200 Gbps within a single server, rendering it ideal for high-performance testing scenarios. Moreover, its fully-featured scalable TCP/UDP integration underscores TRex's scalability, reinforcing its suitability for demanding testing environments. Similarly, MoonGen [Emmerich et al. 2015] achieves speeds of up to 10 Gbps for minimal-sized frames within a single core, scaling up to 120 Gbps for multiple cores. Moreover, it offers exceptional adaptability by allowing users to tailor the packet generation logic using Lua scripts under their control. Additionally, MoonGen leverages hardware features to unlock the possibilities of commodity NICs, further enhancing its capabilities.

Additionally, the Tofino Switch is employed by some traffic generators HyperTester [Zhou et al. 2019] represents a hybrid approach, combining software and hardware for traffic replication. It operates on a single Tofino switch, utilizing the Network Testing API (NTAPI) to define triggers for packet manipulation and statistic collection. HyperTester achieves line-rate packet generation, reaching speeds of 400 Gbps on a hardware

testbed while maintaining precise rate control. Similarly, P4TG [Lindner et al. 2023], another Tofino-based tool, can generate up to 1 Tbps of Ethernet traffic distributed across ten ports at 100 Gbps each. It supports packet customization and provides comprehensive measurements, including L1 and L2 transmission and receive rates, packet loss, round trip time, and inter-arrival times (IATs). Notably, P4TG demonstrates stable IATs for 64-byte frames in constant bit-rate traffic at 100 Gbps, outperforming other traffic generators.

P4TG relies on the same hardware unit for traffic generation, while HyperTester utilizes the Tofino for traffic replication, suggesting potential similarities in performance and accuracy. Unlike P4TG, HyperTester enables user-defined protocols. Additionally, HyperTester relies on the CPU for packet generation, whereas P4TG leverages the Tofino internal traffic generation unit. In this work, we use PIPO-TG, a traffic generator that utilizes the Tofino traffic generation unit without external hardware. Enables users to employ custom protocols without packet restrictions. Furthermore, PIPO-TG offers versatility in traffic generation by allowing users to create various workload models, including random bursts and throughput fluctuations. Additionally, PIPO-TG supports user-defined Protocol-independent Packet Processors (P4) code, enhancing its flexibility and usability.

3. PIPO-TG: Parameterizable High-Performance Traffic Generation

PIPO-TG presents a robust traffic generation solution, leveraging Tofino's traffic generation capabilities in conjunction with Python and P4 processing to achieve speeds of up to 100 Gbps per port, all customizable to user preferences. This tool offers a user-friendly scripting interface akin to Scapy [Biondi 2011], allowing users to define intricate traffic patterns, including protocols, packet size distributions, and throughput parameters. Moreover, PIPO-TG supports concurrent execution of additional P4 code, enabling comprehensive testing scenarios on a single P4 switch without requiring external servers for traffic generation.

3.1. Architecture

The architecture of PIPO-TG, present in Figure 1, provides an overview of the modules, components, and their connections.

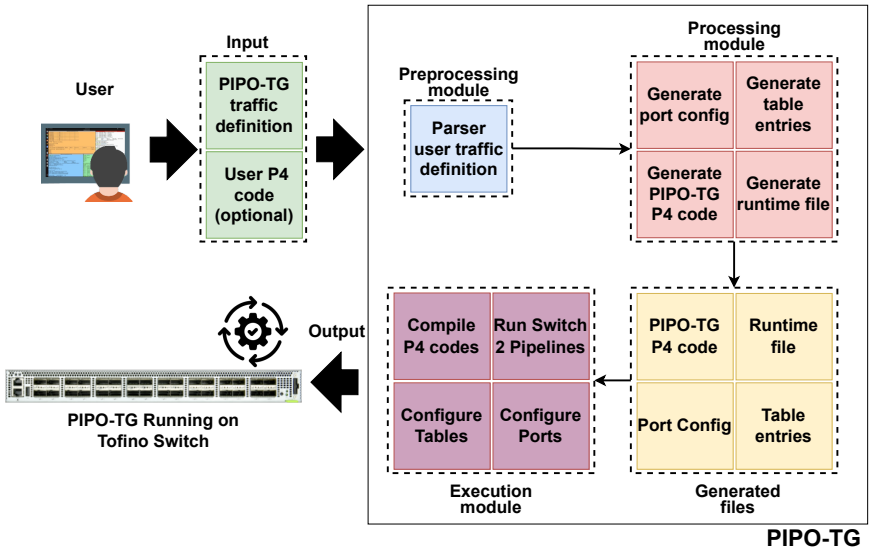


Figura 1. PIPO-TG architecture

Input: PIPO-TG receives traffic pattern definitions and, optionally, user-provided P4 code. To define the generated traffic, users write a straightforward Python script outlining traffic patterns and configuration parameters. The script includes specifications like output port, throughput, common or custom protocols, and additional configuration details like pipeline generation port, port bandwidth, and desired traffic limitations.

PIPO-TG: The fundamental part of our tool is where data processing occurs, and the corresponding files are generated and executed.

Preprocessing Module: The preprocessing module is responsible for parsing, analyzing, and preparing input data for further processing. Within this phase, PIPO-TG ensures that user-defined traffic adheres to Tofino restrictions.

Processing Module: Utilizing the data structures configured, the processing module executes the necessary actions to process input data and generate essential configuration files. This includes utilizing user-defined traffic patterns to generate PIPO-TG P4 code, table entry scripts, port configurations, and execution and interaction scripts.

Generated Files: The generated files encompass the essential configuration files required for running the traffic generator. These files consist of the PIPO-TG P4 code, which is responsible for processing packets generated by the Tofino TG unit based on specified traffic patterns. Additionally, they include a Python script that adds necessary table entries to activate the traffic generator, configure meters and define required packet streams.

Execution Module: The execution module orchestrates the execution of PIPO-TG, encompassing tasks such as P4 code compilation, switch initialization, port configuration, addition of table entries, and initiation of traffic generation. Its output includes the generated traffic forwarded either to the user's P4 code or the defined physical port.

The traffic generation process initiates with users configuring traffic parameters. Subsequently, PIPO-TG generates traffic utilizing the Tofino traffic generation unit, customizes it with the PIPO-TG P4 code, and ultimately directs it to the user's P4 code or designated port. This tool facilitates arbitrary traffic pattern support through an intuitive, high-level software-based programming interface, streamlining the design and operation of hardware-based traffic generators.

3.2. Main Features

PIPO-TG offers some characteristics and features for traffic generation. These features allow users to generate different types of traffic and simulate realistic network scenarios.

Packet Scheme: At its core, PIPO-TG excels in generating basic Ethernet packets destined for specified output ports. By default, it produces 64B Ethernet packets at 100 Gbps, seamlessly forwarding them to the configured port.

Throughput Definition: Users wield the power to designate their desired traffic transfer rate in Mbps, with the flexibility to set rates up to 100 Gbps per port. Moreover, users can allocate up to 10 ports to receive identical traffic, achieving a throughput of up to 1 Tbps.

Common Protocols: PIPO-TG empowers users to craft packets with common protocols like Ethernet, IP, TCP, and UDP. Users can define specific field values (e.g., fixed IP source and destination) or incorporate random value variations (e.g., 100 random IPs). Additionally, users can specify distributions, such as allocating 10.

Custom Protocols: In a departure from conventional approaches limited to traditional protocols, PIPO-TG enables the definition of custom protocols or headers for generated packets. Users enjoy the freedom to create any custom protocol, complete with field definitions and distributions.

Packet Distribution and Definition: Users have granular control over packet size, specifying fixed sizes (e.g., 64B, 128B) or desired distributions (e.g., 20% 64B, 20% 128B).

Workload Analysis: Instead of fixed transfer rates, PIPO-TG facilitates the definition of parameters like minimum and maximum points alongside time distributions. This feature enables users to replicate various real traffic examples, such as traffic based on a sinusoid wave and a mathematical model of the sine function [de Almeida et al. 2023], resulting in a periodic load behavior.

User P4 Code: Leveraging multiple pipeline support, users can execute custom P4 code within PIPO-TG, enabling seamless reception of generated traffic.

4. 6G/TSN Use Case

Due to the number of features provided and the flexibility in varying parameters, PIPO-TG is capable of reproducing a large set of traffic scenarios. Even so, in this work, we further extend the traffic generation capabilities of PIPO-TG to support a new use case: delay distribution in 6G/TSN networks.

This is highly challenging because while the time-varying features present in the original PIPO-TG implementation work on the scale of seconds, 5G/TSN networks have requirements in the nanosecond scale. Therefore, we had to adapt our way of generating traffic and our mechanism for controlling packets sent to a lower granularity and thus get closer to the required times. Next, we discuss the scenario considered and the data used, and we provide some results from our experiments.

4.1. Scenario: Reproducing Time-sensitive networking delays

TSN is a set of standards designed for deterministic communication over Ethernet networks. It ensures precise timing and low-latency communication, vital for time-critical applications like industrial automation and automotive systems, by synchronizing devices and guaranteeing reliable data delivery. Ensuring deterministic communication with extremely low delay in the nanosecond range poses significant challenges, and replicating these conditions with high fidelity using real hardware is equally daunting.

In our scenario, we seek to reproduce real TSN communications data, measured and made available by the DETERMINISTIC6G project [DETERMINISTIC6G 2023]. The DETERMINISTIC6G project is an initiative aimed at addressing the emerging challenges of end-to-end deterministic communication enabled by 6G. The project aims to develop a new architecture for 6G systems that provides predictable performance and integrates it end-to-end with TSN and DetNet, thus bridging the gap between wired deterministic communication standards and the current wireless network infrastructure.

We hope that by developing a tool capable of reproducing this traffic pattern in commercial programmable switches, such as Tofino, we can encourage even more research in the area. So, we updated PIPO-TG so that it can understand the histograms

generated by DETERMINISTIC6G measurements. An example of these histograms can be seen below:

```
<histogram>
    <bin low="1500ns">3</bin>
    <bin low="2000ns">13</bin>
    <bin low="2500ns">20</bin>
    <bin low="1000ns">0</bin>
</histogram>
```

Each histogram consists of "bin" elements representing time intervals. Each bin specifies its lower bound and contains a count representing the number of packets falling within that interval. The last bin is used to define the upper bound of the previous bin and always has a count of zero. So, after reading the histograms, we configure PIPO-TG appropriately and reproduce the traffic following the pattern defined by the histogram. Next, we present some results obtained.

4.2. Experiments

Setup. We evaluate the performance of PIPO-TG using a *Tofino Switch* (Edgecore Wedge 100BF-32X) and one server (Intel Xeon E5-2620v2, dual-port 10G Intel X540-AT2 NIC, and 64GB of memory running Ubuntu 20.04) connected via 10G SFP+ interface. We also defined minimum-sized (64B) packets, as described in the measurements.

Methodology. Given the variety of experimental configurations in the published measurements, delays can range from a few hundred nanoseconds to several hundred thousand nanoseconds. Initially, we assess the accuracy of desired and generated delays, ranging from $100ns$ to 1 million nanoseconds. To achieve this, we transmit 300 packets. The first 100 packets increment the desired delay by $100ns$, the next 100 increase it by $1000ns$, and the last 100 increase it by $10,000ns$.

Delay Generation. Figure 2 presents the results. We can see that smaller values and increments exhibit a greater disparity between the desired and generated delays (e.g., the lowest delay value that can be generated is $595ns$), whereas larger values demonstrate closer results. This occurs because Tofino needs a few hundred nanoseconds to process a packet, and recirculations hold the packet until the desired delay. If the packet is close to the desired delay but still has to recirculate once more, the packet is sent with an extra delay in relation to the desired delay due to packet recirculation and reprocessing (usually between 300 and $400ns$). This means that gradual delay increments are only supported in this range, explaining the "staircase" behavior in the generated delays. However, as we can see, for larger values, this time becomes negligible.

Traces Reproduction. Figure 3 presents the results for two different real DETERMINISTIC6G measurements: 3(a) a measurement with nanoseconds in the region of 4k, and 3(b) with delays of up to 17k. As demonstrated previously, PIPO-TG has difficulties with small delay variations (e.g., from 12 to 12 in 3(a)), but with slightly higher values, it generates values very close to ideal.

5. Documentation, Code, and Demonstration

PIPO-TG is an open-source project available at `https://github.com/FilipoGC/PIPO-TG`. The project is ongoing, with frequent feature updates and possible bug fi-

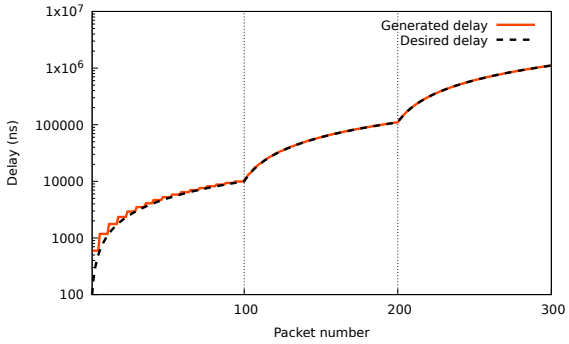
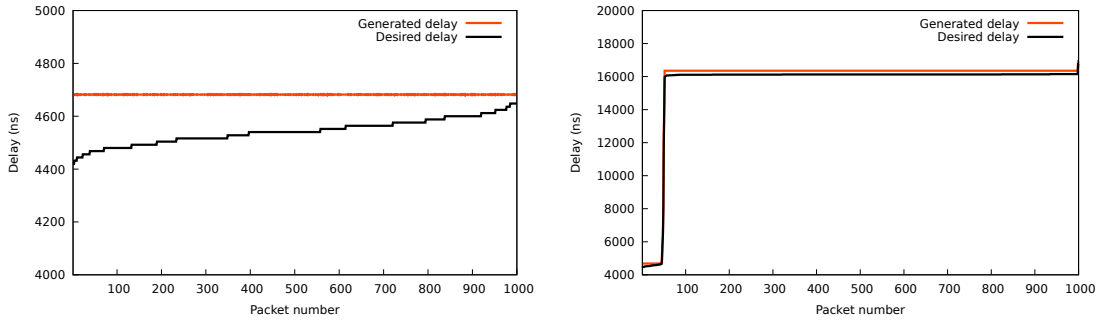


Figura 2. Generating delays from 100 nanoseconds to 1M nanoseconds



(a) Reproduction of trace "$nocross\_vlan\ -\ 10pkt\_per\_sec$" (b) Reproduction of trace "$cross900mbps\ -\ 10pkt\_per\_sec$"

Figura 3. Reproduction of different DETERMINISTIC6G traces

xes. The project is also open to contributors who can contribute with ideas, bug reports, and even new features. The documentation is also available on our github project and provides a complete usage description of all traffic generation features (i.e., packet characteristics, delay distribution). Additionally, we provide a video tutorial showing the configuration and execution of the traffic generator applied for different traffic patterns. The video is available at `https://drive.google.com/drive/folders/1ruF0YIjaaO7xpsjz44nbTOjTAjd7KRmw?usp=sharing`.

The demonstration will focus on reproducing the use case described in Sec. 4, showcasing the potential of PIPO-TG in reproducing TSN scenarios with high fidelity. Furthermore, attendees will be able to visualize how PIPO-TG works in other scenarios, propose different traffic patterns, and exploit the flexibility offered by PIPO-TG.

6. Final Remarks

This work presents advanced features of the PIPO-TG open-source Tofino-based traffic generator. We showcase the improved PIPO-TG's capabilities and demonstrate its efficiency through a new use case: reproducing different TSN delay distributions and creating packets with delays on a nanosecond scale. Future plans for PIPO-TG include supporting stateful connections (e.g., TCP, QUIC) and developing monitoring instrumentation.

Acknowledgments

This work was partially supported by the Innovation Center, Ericsson S.A., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS. Also, this work was partially supported by FAPESP grants 2023/00794-9 and 2021/06981-0. Finally, this study was partially funded by CAPES, Brazil - Finance Code 001

Referências

Biondi, P. (2011). Scapy. Available: `https://scapy.net/` [Acces: April 05, 2024].

Botta, A., Dainotti, A., and Pescapé, A. (2010). Do you trust your software-based traffic generator? *IEEE Communications Magazine*, 48(9):158–165.

Costa, F. G., Vogt, F., Rodriguez, F., de Castro, A. G., Luizelli, M. C., and Rothenberg, C. E. (2024). Pipo-tg: Parameterizable high-performance traffic generation. In *To appear in IEEE/IFIP Network Operations and Management Symposium (NOMS) 2024*.

de Almeida, L. C., da Silva, J. L., Lins, R. P., Maciel Jr, P. D., Pasquini, R., and Verdi, F. L. (2023). Wave-um gerador de cargas múltiplas para experimentação em redes de computadores. In *Anais Estendidos do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 9–16. SBC.

DETERMINISTIC6G (2023). Deterministic6g deliverable 4.1, "deterministic6g detcom simulator framework release 1". Online. Available: `https://deterministic6g.eu/index.php/library-m/deliverables` [Acces: April 05, 2024].

Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., and Carle, G. (2015). Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*, pages 275–287.

Hobbit (1995). Netcat. Available: `https://nc110.sourceforge.io/` [Acces: March 20, 2024].

Jones, R. (1996). Netperf. hewlett-packard. Available: `https://github.com/HewlettPackard/netperf` [Acces: June 26, 2023].

Kundel, R., Rizk, A., and Koldehofe, B. (2020). Microbursts in software and hardware-based traffic load generation. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–2.

Lindner, S., Häberle, M., and Menth, M. (2023). P4tg: 1 tb/s traffic generation for ethernet/ip networks. *IEEE Access*, 11:17525–17535.

Liu, W.-x., Liang, C., Cui, Y., Cai, J., and Luo, J.-m. (2022). Programmable data plane intelligence: advances, opportunities, and challenges. *IEEE Network*.

Mortimer, M. (2018). iperf3 documentation.

Mosberger, D. and Jin, T. (1998). Httperf—a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37.

TRex, T. (2023). Trex realistic traffic generator. Available: `https://trex-tgn.cisco.com/` [Acces: March 20, 2024].

Zhou, Y., Xi, Z., Zhang, D., Wang, Y., Wang, J., Xu, M., and Wu, J. (2019). Hypertester: high-performance network testing driven by programmable switches. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 30–43.