PIPO-TG: Parameterizable High-Performance Traffic Generation

Filipo G. Costa[+], Francisco G. Vogt[*], Fabricio Rodríguez Cesen[*], Ariel Góes de Castro[*]
Marcelo Caggiani Luizelli[+], Christian Esteve Rothenberg[*]
[+]Federal University of Pampa (UNIPAMPA), Brazil
[*]University of Campinas (UNICAMP), Brazil

***Abstract*—In recent years, the increasing demand for network resources due to real-time applications and data-intensive activities has posed challenges in managing and optimizing network performance. To assess network performance, security, and efficiency, traffic generation plays a crucial role. We introduce PIPO-TG, a Tofino-based traffic generation for high-performance experiments. The primary objective of PIPO-TG is to generate realistic and diverse traffic patterns, enabling researchers to evaluate network performance under varying conditions providing customizable packet forwarding with P4 programmable data planes. Our main contributions include user-defined packet header customization and open-source code for reproducibility. These efforts foster collaboration within the research community to advance traffic generation techniques. We show that PIPO-TG only requires a few lines of code to simulate heterogeneous network scenarios (e.g., traffic bursts and DDoS attacks) while maintaining hardware performance and flexibility.**

I. Introduction

Traffic generation tools have been widely used to assess network performance, efficiency, and security in research and practical applications. The capability to generate controlled and realistic network traffic has become paramount for recent advances in programmable networks [1]. By enabling the controlled generation of realistic network traffic and emulating real-world scenarios, these tools empower researchers and practitioners with valuable insights into network system behavior. These insights, derived from the experiments using traffic generation tools, contribute to enhancing network technologies, making them an essential asset for those seeking to navigate the complexities of current networks.

Despite consistent efforts made by the research community, existing traffic generation solutions still pose some limitations. Software-based traffic generation solutions [2]–[4] offer a cost-effective and versatile means of simulating network traffic. These solutions can be executed on top of commodity servers and provide high flexibility for traffic generation, user-friendly interfaces, and intuitive controls, simplifying the process of generating traffic. However, these generators have performance limitations and struggle to reach line-rate values or generate a few hundred Gbps. Hardware-based traffic generation solutions [5], [6] are built on top of specialized hardware components and are a common choice for high-performance experiments. Although they can reach line rate performance metrics, they are usually challenging to use, inflexible to changes (e.g., creation of new protocols), and costly.

More recently, some research efforts [7], [8] have emerged to design easy-to-use solutions to generate network traffic with high performance and without dedicated hardware. These solutions are based on generating traffic with Tofino hardware, a P4-based switch capable of processing hundreds of Gbps per port, and also perform internal traffic generation. However, these strategies still have limitations in their traffic generation. While HyperTester [7] needs an auxiliary CPU to create its packets and is not completely open-source, P4TG [8] does not support the definition of customizable protocols and is limited to only a few different network flows. Additionally, both solutions do not support throughput variations (e.g., bursts and varying workloads) and do not support running a user-defined P4 code side-by-side with Tofino traffic generation.

Addressing these limitations while maintaining the benefits offered by these solutions is not a trivial task. Working with Tofino and the P4 language to maintain high performance and traffic generation accuracy includes many challenges. Tofino's native traffic generation unit does not support resolving these limitations; thus, we must address them through the P4 code. However, P4 has its restrictions, such as instruction limits, and does not support complex comparisons, floating-point operations, and loops.

In this work, we present PIPO-TG, a parameterizable and high-performance traffic generation solution. PIPO-TG is built on top of the Tofino Native Architecture (TNA) [9] and relies on P4 language to describe and customize packet generations on the TNA architecture. PIPO-TG can generate network traffic up to 1 Tbps line rate, ensuring accurate performance evaluations without introducing bottlenecks or distortions. PIPO-TG introduces support for arbitrary traffic patterns using a high-level software-based programming interface that makes the design and operation of a hardware-based traffic generator. For example, by using the PIPO-TG programming interface, we can easily define a variety of network workloads and forward them to a user P4 code running on the same switch.

PIPO-TG extends Tofino traffic generation capabilities and provides features never seen in other Tofino-based traffic generators. In Figure 1, we illustrate the entire traffic genera-

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. This work was partially funded by National Council for Scientific and Technological Development (CNPq 404027/2021-0), Foundation for Research of the State of Sao Paulo (FAPESP 2020/05115-4, 2021/06981-0, 2021/00199-8, 2020/05183-0).

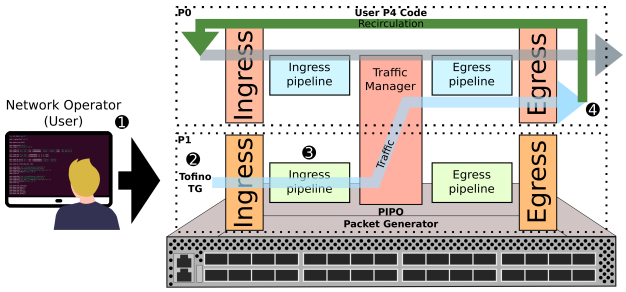


Fig. 1: Overview of PIPO-TG traffic generation in TNA.

tion process: ❶ users set the traffic parameters, ❷ PIPO-TG generates traffic utilizing the Tofino traffic generation unit, ❸ tailors it using the PIPO-TG P4 code, and ❹ subsequently routes it to the user's P4 code or defined port.

Our main contributions are summarized as follows:

- A hardware-based traffic generator for line-rate traffic;
- A high-level interface for easy traffic generation
- A parameterizable traffic generator that extends Tofino capabilities, allowing users to generate customizable traffic with different packet distributions and rates.
- Open source artifacts for the sake of reproducibility.[1]

The remainder of this paper is organized as follows. Section II discusses our background topics while Section III presents the related works. In Section IV, we introduce the PIPO-TG architecture, features, and implementation. Section V presents and discusses an evaluation of the proposed approach in three use cases. Last, in Section VI, we conclude the paper with final remarks and perspectives for future work.

II. Background

In this section, we provide a comprehensive background overview covering network traffic generation and the Tofino Native Architecture. It starts by delving into the fundamentals associated with conventional traffic generation and then progresses to explore concepts such as the P4/TNA architecture and Tofino traffic generation. Finally, we highlight the technologies underpinning our traffic generator.

A. Traffic generation

In the realm of computer networking, traffic generation refers to the creation of artificial traffic that mimics the behavior of real-world network traffic for testing and evaluating network devices, protocols, and applications. Traffic generation is crucial in developing, testing, and validating modern network applications, especially in data centers, cloud computing, and software-defined networking contexts.

In this context, traffic generators are designed for injecting controlled packets into the network, providing high flexibility. This flexibility allows network engineers to replicate traffic behavior and simulate specific events and desired traffic patterns. Whether mimicking the heavy data transfers of a file transfer protocol, the intermittent bursts of a VoIP call, or the continuous flow of web traffic, these traffic generators offer the versatility to recreate a wide array of network scenarios. This capability empowers engineers to thoroughly test and optimize network configurations, ensuring that networks can adapt and perform optimally under various conditions and meet the demands of real-world usage.

Traffic generators can be categorized into two distinct groups: software-based and hardware-based traffic generators. This classification is determined by the methods employed for traffic generation. Software-based generators primarily rely on software applications to simulate and generate network traffic, whereas hardware-based generators use specialized hardware components to accomplish this task. Several critical factors come into play when deciding whether to opt for a hardware-based or software-based traffic generator. Table I presents the benefits of choosing traditional software or hardware traffic generation techniques and compares them with the help of PIPO-TG. Below, we discuss each of these benefits.

Usability. The usability determines the ease of use for a traffic generator. Hardware-based solutions are usually not user-friendly since users must configure low-level features to obtain the desired behavior. On the other hand, software-based solutions provide a transparent platform to the user, where he only cares about declaring the desired behavior. PIPO-TG balances both approaches with the simplicity of stating approaches in software and hardware performance.

Accuracy. Accuracy in traffic generation refers to how closely the generated traffic patterns match real-world network behavior. Software-based generators may be limited in replicating complex or nuanced traffic patterns. In contrast, hardware-based generators often offer higher accuracy by using specialized hardware components to emulate real traffic precisely.

Flexibility. Flexibility assesses how easily a traffic generator can be configured and adapted to different network environments and testing scenarios. Software-based traffic generators are more flexible, allowing for versatile configuration and adaptability to various network setups. Hardware-based solutions may have limitations in flexibility because they are built around specific hardware components.

Performance. Performance refers to the ability of a traffic generator to handle and generate network traffic effectively and efficiently, principally in terms of traffic volume. In general, Hardware-based traffic generators can deliver superior performance, especially when dealing with high traffic loads (e.g., hundreds of Gbps) or complex scenarios, thanks to their dedicated hardware. Software-based generators might perform less for demanding network testing tasks due to their reliance on general-purpose computing resources.

Features. Features encompass the capabilities and functionalities offered by a traffic generator. Due to their dedicated hardware components, hardware-based generators often provide advanced features and capabilities. Software-based generators can vary widely regarding available features, depending on the specific software used and its capabilities.

In this context, PIPO-TG tries to take advantage of the best of both worlds, using Tofino switch's hardware traffic generation capabilities combined with Python and P4 applications. It

[1]https://github.com/FilipoGC/PIPO-TG/

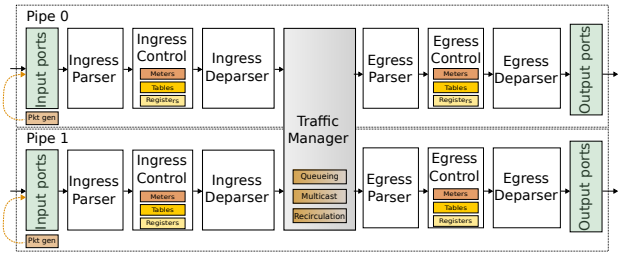


Fig. 2: TNA architecture.

extends the Tofino traffic generation possibilities, providing a user-friendly and flexible interface while maintaining accuracy and performance.

B. P4 and Tofino Native Architecture

P4 is a network programming language that specifies packet processing behaviors and network forwarding in a protocol-agnostic manner at a high level of abstraction. P4-enabled devices, called targets, can be reprogrammed on the fly to adapt to evolving network requirements without expensive hardware upgrades or vendor-specific software. However, each P4 target follows a specific architecture containing its components, units, externs, and limitations. The Intel Tofino switch is an example of a P4 target following the TNA architecture.

The TNA offers network operators a suite of programming tools and abstractions for fine-tuning the behavior of Tofino ASICs. It empowers the creation of efficient, adaptable data plane pipelines tailored to precise network needs. TNA integrates a compiler and run-time system, converting P4 programs into optimized low-level instructions for Tofino ASIC execution, leveraging the hardware's capabilities to enhance performance and efficiency.

Additionally, TNA adopts an extensively parallel and programmable pipeline architecture, facilitating the concurrent execution of even four P4 codes across parallel pipelines. These pipelines are optimized for efficient packet processing tasks and are structured with ingress and egress units, encompassing a parser, a deparser, and dedicated processing stages. Figure 2 presents a two-pipeline TNA architecture where we can see the ingress and egress units commented and the traffic manager responsible for tasks such as traffic scheduling, buffering, and management.

C. Tofino traffic generation

In addition, the Tofino switch can generate packets internally. As discussed previously, Tofino can contain 2 or 4 pipelines, each with its internal traffic generation unit (see Figure 2). Each of these units can be configured individually and support the definition of eight different packet streams. These streams have a set of parameters (e.g., headers, pkt size, inter-packet gap). They can be activated by four different triggers: port down, packet recirculation, one-time, and periodic. With its triggers and streams, each pipeline can generate up to 100 Gbps of traffic and forward it to a desired output port.

Despite providing a powerful traffic generation unit, Tofino's native traffic generator has limitations. These limitations include the maximum number of flows that can be generated (only 8 per pipeline), lack of support for custom headers, and, principally, the configuration complexity, requiring the configuration of dozens of parameters to generate a simple IP packet. In this context, PIPO-TG seeks to extend Tofino's traffic generation capabilities and simplify its configuration.

TABLE I: Comparison of SW/HW traffic generation.

Characteristic	Software-based	Hardware-based	PIPO-TG
Usability	✓	✗	✓
Accuracy	✗	✓	✓
Performance	✗	✓	✓
Resources	✗	✓	✓
Flexibility	✓	✗	✓

III. Related work

Traffic generators serve as tools for designing and managing networks. They introduce controlled packets into networks to replicate traffic patterns. However, establishing evaluation methods and metrics to ensure accuracy remains a challenge [12], which calls for research. When comparing hardware-based generators with software-based ones, it is important to consider cost implications. Software-based solutions are more cost-effective, while hardware-based options offer design and precision (see Table I). This analysis assists professionals in selecting the traffic generator that suits their requirements. Recently, both software- and hardware-based approaches have been developed in the context of traffic generation, mainly used for academic evaluation. Next, we describe the main software and hardware solutions for traffic generation, emphasizing how PIPO-TG differs from them.

A. Software-based traffic generation

Traditional software approaches [13]–[16] are widely used for bandwidth measurement. Iperf3 [13] is the de-facto open-source tool that supports various protocols and provides client and server modes for generating and measuring traffic. Similarly, Netperf [14] leverages the Berkeley Sockets interface and supports Data Link Provider Interface (DLPI), Unix Domain Sockets, and IPv6 measurements. Netcat [15] is also a data transfer over TCP/UDP tool for Unix systems. It is a reliable tool that can be directly or seamlessly integrated into other programs and scripts. Httperf [16] allows the operator to generate and sustain server overload. It supports the HTTP/1.1 protocol and its extensibility for incorporating new workload generators and performance measurements.

In contrast, some approaches [2]–[4] rely on DPDK [17] to generate network traffic. Pktgen [2] reaches up to 10 Gbps for 64-byte frames, acting as both a transmitter and receiver at line rate. Similarly, MoonGen [3] reaches up to 10 Gbps for minimal-sized frames within a single core. For multiple cores, it can generate up to 120 Gbps. Furthermore, it provides unparalleled flexibility by enabling users to customize the packet generation logic through user-controlled Lua scripts. Adding to its capabilities, MoonGen harnesses the untapped potential of commodity NICs by utilizing hardware features.

TABLE II: Comparative review of related work on traffic generators (TG).

	Software-based TG			Hardware-based TG					
Feature	Pktgen [2]	MoonGen [3]	TRex [4]	Yuan [10]	HyperTester [7]	P4STA [11]	Plakalovic [6]	P4TG [8]	**PIPO-TG**
100 Gbps on multiple ports	✗	✗	✗	✗	✓	✓	✗	✓	✓
Workload assay	✗	✗	✗	✗	✗	✗	✗	✗	✓
Custom traffic	✓	✓	✓	✓	✓	✗	✓	✗	✓
Internal generation.	✓	✓	✓	✓	✗	✗	✓	✓	✓

TRex [4] is an open-source traffic generator that operates in stateless and stateful modes, generating traffic on layers L3 to L7. It reaches up to 200 Gbps within a single server, making it suitable for high-performance testing scenarios. Scapy [18] is a robust Python library that allows users to create, decode, send, and capture user-defined packets. Additionally, it is cross-platform, offering native support for Linux, macOS, most Unix-based systems, and Windows with Npcap, while LuaJIT [19] and MoonGen [3] are built on Lua language.

B. Hardware-based traffic generation

More recently, hardware-based solutions [6]–[8], [10], [11] have been developed due to their ability to generate traffic at higher rates. Yuan et al. [10] introduces a fast flow-based Ethernet Traffic Generator on FPGA as a cost-effective solution for network evaluation. The system offers a user-friendly interface for control and generates up to 10 Gbps. HyperTester [7] and P4STA [11] are hybrid software- and hardware-based traffic replicators. The first runs on a single Tofino switch. It involves using the Network Testing API (NTAPI) to define triggers for packet manipulation and statistic collection. By utilizing these expressions, template packets are created along with a corresponding P4 program that enables the desired functionality. Evaluations on the hardware testbed demonstrate that HyperTester achieves line-rate packet generation, reaching speeds of 400 Gbps while maintaining highly accurate rate control. Similarly, P4STA [11] leverages the accuracy of hardware packet timestamping. It runs on an off-the-shelf P4-programmable switch and reaches 100 Gbps per port, keeping a 1-nanosecond hardware timestamp resolution. Once the device under test responds by sending back the packets, they are appended with a second timestamp and duplicated to an external host. The hardware timestamps are extracted at the external host and can be utilized to calculate Round-trip times (RTTs) and other relevant metrics. Plakalovic et al. [6] is implemented on VHDL and fully utilizes a 40 Gbps link while offering high flexibility to manipulate traffic characteristics at the packet level in FPGA boards. The hardware design of the packet generator. P4TG [8] is another Tofino-based work. It can generate up to 1 Tbps of Ethernet traffic distributed across ten ports at 100 Gbps each. Also, it supports packet customization and provides measurements of L1 and L2 transmission and receive rate, packet loss, out-of-order packets, 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, surpassing the performance of other traffic generators. Additionally, P4TG is capable of generating random traffic, enhancing its versatility.

C. Comparison

Table II compares the most relevant related work and PIPO-TG considering the following key criteria:

- **100 Gbps on multiple ports:** The capability to generate 100 Gbps or more in multiple ports and at the same time.
- **Workload assay:** The possibility to generate workloads for more realistic experiments. Instead of generating traffic based on a fixed throughput, workload patterns can be based on a pre-defined model (e.g., and Flashcrowd [20]).
- **Custom traffic:** The capability to generate traffic with customizable parameters. It allows researchers to define specific traffic patterns, such as traffic volume, packet size, distribution, or specific protocols.
- **Internal generation:** The ability to generate traffic without the need for extra resources or servers.

As we can observe, similarly to P4TG [8], HyperTester [7] and P4STA [11], PIPO-TG can generate 100 Gbps on multiple ports. However, PIPO-TG is the only one that combines this performance with a high-level interface and allows customizable traffic generation and following workload assay models. As a disadvantage, PIPO-TG does not have a monitoring system integrated into the generator.

IV. PIPOTG

PIPO-TG is a traffic generator that uses Tofino's traffic generation capabilities combined with Python and P4 processing to generate up to 100 Gbps per port of parametrizable traffic. PIPO-TG allows users to define the traffic patterns using a user-friendly script similar to Scapy to define the traffic protocols, packet size distribution, throughput, and others.

Furthermore, while generating traffic, users can simultaneously execute another P4 code that receives this traffic and can carry out its operation normally. It allows testing a P4 code on a single P4 switch without needing an external server for traffic generation. In the following sections, we discuss the traffic generation process using PIPO-TG, its architecture, main features, implementation, and current limitations.

A. Architecture

The PIPO-TG architecture presented in Figure 3 illustrates the high-level components and their interconnections. These components are described below:

*1) **Input**:* As input, PIPO-TG receives the traffic patterns definition and, optionally, a user-provided P4 code. To define the generated traffic, the user needs to write a simple Python script describing the traffic patterns along with configuration parameters. Algorithm 1 presents an example of PIPO-TG input code to generate IP packets at 100 Mbps with destination

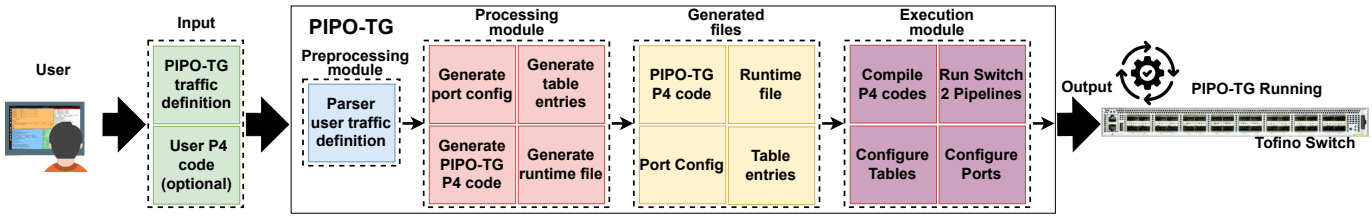


Fig. 3: PIPO-TG architecture

Algorithm 1 Example of PIPO-TG traffic generation input

```
import PipoTG
#instantiate the traffic generator
myTG = PipoGenerator()
#define the generation port
myTG.addGenerationPort(68)
#Phys port, Port ID (D_P), Port BW
myTG.addOutputPort(5, 160, "100G")
#set IP header with dst addr
myGenerator.addIP(dst= "10.0.0.2")
#create a 8 bits custom header
customHeader = Header(name="myHeader", size=8)
customHeader.addField(Field("metadata", 8))
myTG.addHeader(customHeader)
#define Throughput and type (port_shaping/meter)
myTG.addThroughput(100, "meter")
#start traffic generator
myTG.generate()
```

IP 10.0.0.2 and a custom header to be sent via physical port 5 (configured to receive up to 100 Gbps). Additionally, the user defines configuration details such as the pipeline generation port, port bandwidth, and the type of traffic limitation desired (more information in Section IV-C). As mentioned, the user can also define a P4 code to run simultaneously along with PIPO-TG. The P4 code is optional because the user can directly send the generated traffic to an external server without the need to go through a new P4 code.

*2) **Preprocessing module**:* The preprocessing module parses, analyzes, and prepares the input data for the processing module. In this step, PIPO-TG analyzes whether user-defined traffic is consistent with Tofino restrictions (e.g., checking whether custom headers are byte-aligned) and whether all configuration parameters have been declared with valid values. In case of any problem, the unit returns a message to the user with what needs to be corrected. Otherwise, the unit only configures the data structures used by the processing module.

*3) **Processing module**:* The processing module uses the data structures configured by the preprocessing module to process the input data and generate all the necessary configuration files. In this process, we use the user-defined traffic patterns to generate the PIPO-TG P4 code, the table entries script, the port configurations, and the script for execution and interaction. Note that this processing generates unique configuration files that match the defined traffic patterns, and any changes require the generation of new files.

*4) **Generated files**:* The generated files are the configuration files necessary to run the traffic generator. They include the PIPO-TG P4 code, which receives the packets generated by the Tofino TG unit and performs additional processing on the packets according to the specified traffic patterns. They also have a Python script that adds all the table entries necessary to activate the traffic generator, in addition to configuring the meters and defining the required packet streams. Finally, they have the execution script (shell script) and the file configuring the ports after Tofino is started.

*5) **Execution module**:* This step coordinates the execution of PIPO-TG. It includes compiling the P4 codes, initializing the switch, configuring the ports, adding the table entries, and then initiating traffic generation. The output of this module is the traffic being generated and forwarded to the user's P4 code or the defined physical port.

B. Main features

PIPO-TG offers several features and characteristics for traffic generation. These features allow users to generate different types of traffic, being able to simulate unlimited network scenarios. Note that the features that will be presented are not the literal commands present in the PIPO-TG script. To see the available commands and how to use them, access our repository [21]. Next, we present the main features available:

Packet crafting. The most basic function of PIPO-TG is the ability to generate basic Ethernet packets for a defined output port. By default, PIPO-TG will generate 64B Ethernet packets at 100 Gbps and forward them to the configured port.

Throughput definition. Users can specify their desired traffic transfer rate in Mbps, with the ability to set rates of up to 100 Gbps for a particular port. Furthermore, users can assign up to 10 ports (due to multicast limit) to receive identical traffic, resulting in a maximum achievable throughput of 1 Tbps.

Common protocols. Users can create packets with protocols like Ethernet, IP, TCP, and UDP. Furthermore, the user can define the specific value for the fields (e.g., a fixed IP source and destination) or a random number of values to alternate (e.g., 100 random IPs). Alternatively, the user can specify a limited distribution (e.g., 10% of packets with IP X, etc).

Custom protocols. In addition to conventional protocols, PIPO-TG allows the definition of any customized protocol or header for the generated packets. Unlike other strategies that only support traditional protocols such as Ethernet and IP, with PIPO-TG, the user is free to create any custom protocol, defining its fields and distributions.

Packet size definition and distribution. The users define the packet size for the generated packets using a fixed definition (i.e., 64B, 128B, 256B, 512B, 1024B, 1280B, and 1518B) or

following a desired distribution (e.g., 20% 64B, 20% 128B, 40% 512B, 20% 1024B).

Workload assay. Instead of the user defining a fixed transfer rate for the traffic, the idea is to enable users to define parameters such as minimum and maximum points in addition to the time distribution in those points. It allows users to reproduce different traffic distributions, such as those presented in [20] for workload assay generation. Currently, PIPO-TG only supports a limited number of throughput points and enables the creation of a simplified version of the Flashcrowd model.

User P4 code support. Users can benefit from the multiple pipeline support to execute a user-defined P4 code that receives the traffic generated by PIPO-TG pipeline.

C. Implementation

Next, we will discuss the PIPO-TG implementation, presenting the strategies for implementing the available features. Table III summarizes the strategies used to implement each feature. Additionally, we discuss each of them in more detail:

*1) **Traffic crafting**:* To create the packets, we use the packet definition available in Tofino packet generation. The most basic packet that can be defined is an Ethernet packet with a size of 64B. Furthermore, the user can generate traffic on multiple ports (e.g., to generate up to 1 Tbps) with the traffic manager multicast function to replicate the generated packets.

*2) **Throughput**:* To ensure the throughput defined by the user, we use two methods: port shaping, limiting the output port, or the meter configuration. In the high-level script, the user can choose between both options. To generate more than 100 Gbps, we use the Tofino multicast function to replicate the traffic generated. However, multicast is limited to 10 ports, therefore we can reach up to 1 Tbps.

*3) **Common protocols**:* We leverage the packet definition available in the Tofino traffic generation for standard protocols. Since Tofino already generates packets with the desired protocols, we only change the P4 code when the user defines more complex configurations (e.g., Random IPs, which have their distribution controlled using the random extern [9]).

*4) **Custom protocols**:* To generate flows with customizable protocols, we generate the packets with the standard Ethernet headers and use the P4 code to define and add the user-defined headers. In the P4 code, in addition to including customizable headers, we use tables and table entries to configure the fields.

*5) **Packet size**:* To define the packet size and switch between different packet sizes, we use the eight packet streams available in Tofino traffic generation. With this, we can generate up to 8 different package sizes, and we control their distribution through the P4 code and the random extern [9].

*6) **Workload assay**:* For workload generation and complex traffic patterns, we use the definition of multiple meters combined with hardware timestamp monitoring. Therefore, we define multiple throughput limits using the meters and switch between them according to the time.

*7) **User P4 code support**:* PIPO-TG takes advantage of Tofino's multiple pipeline support for executing different P4 codes in different pipelines. Our scripts configure the PIPO-TG P4 code to run in one pipeline and the user code to run in another. The generated traffic is received by the PIPO-TG P4 code, and after processing, changes for the user P4 code pipeline using the traffic manager (See Figure 1). To do this, we forward the packet to the recirculation port of the user pipeline and set the $egress\_bypass$ flag. Thus, the packet switches to the user's pipeline without executing egress processing but is recirculated to be executed by the ingress processing of the user's pipeline.

TABLE III: PIPO-TG implementation overview.

Feature	Implementation approach
Traffic crafting	Tofino internal traffic generation unit to create, and P4 code to parse, edit, and forward
Throughput [Mbps]	Port shaping in the output port, or Meter algorithm to drop packets
Common headers	Tofino packet generation unit to define, and P4 code to edit using tables and the random extern
Custom headers	P4 code to include custom headers, Tables and table entries to modify fields
Packet size	Tofino TG unit to create different streams, and P4 code with random extern to coordinate
Workload assay	P4 code and hardware timestamp to measure time, Different meters to control throughput
User P4 code support	Tofino multi pipeline support Traffic manager + bypass egress to change pipeline

D. Limitations

While PIPO-TG enhances the traffic generation capabilities of Tofino and offers a user-friendly interface and high flexibility, it is essential to acknowledge that it also has some existing limitations. Firstly, as we use a P4 code to generate PIPO-TG traffic when testing a user P4 code, there will be one less pipeline to receive and send traffic, that is, 16 fewer physical ports available. Furthermore, packets sent to from PIPO-TG to the user's P4 code must recirculate to the user pipe meaning that the user code can receive a maximum of 100 Gbps.

Although PIPO-TG has several features, we have some restrictions when using multiple elements together. For example, it is impossible to combine the generation of random packet sizes with the generation of random IPs with the variation of customizable header parameters (see our GitHub documentation for more details). Additionally, PIPO-TG does not support stateful connections (such as TCP) and only sends packets without saving the state or waiting for a response. Finally, unlike P4TG, T-REX, and other traffic generators, PIPO-TG does not have an integrated interface for monitoring the generated traffic.

V. Evaluation

In this section, we delve into the comprehensive evaluation of PIPO-TG. First, we present a brief comparison between PIPO-TG and other traffic generators. Next, we present three use cases for PIPO-TG: workload alternation, burst simulation, and Distributed Denial of Service (DDoS) simulation. We do not evaluate PIPO-TG resource utilization, because as Tofino pipelines have isolated resources, the PIPO-TG resources do not impact the user's pipeline.

Setup. The experiments were conducted in a Tofino switch (Edge-Core Wedge100BF-32X) directly connected to a local server using a high-speed 100 Gbps cable.

A. PIPO-TG vs. State-of-the-Art

We compare PIPO-TG against P4TG [8] and HyperTester [7] towards different facets. Given that PIPO-TG and P4TG utilize the same hardware unit for traffic generation and HyperTester uses Tofino for traffic replication, the results would be similar in performance and accuracy. Then, in Table IV, we compare PIPO-TG, P4TG, and HyperTester qualitatively regarding their characteristics.

Custom Protocols. Refers to working with customizable headers, including new user-defined protocols. HyperTester and PIPO-TG support custom protocols, while P4TG is limited to generating Ethernet/IP packets.

Number of flows. We assess the limitation of traffic generators in creating many distinct flows. P4TG is restricted to 7 different flows due to Tofino traffic generation limitations. In contrast, PIPO-TG, which extends Tofino's traffic generation with P4 modifications, and HyperTester CPU-based packet generation do not face this limitation.

Tofino internal traffic generation. We evaluate whether the traffic is generated using the Tofino internal generation unit, that is, without the need for a CPU or any other server. In this case, P4TG and PIPO-TG generate traffic using the Tofino unit, while HyperTester relies on the CPU to generate packets and only amplifies its traffic with Tofino.

Workload generation. We assess the capacity of traffic generators to produce diverse traffic behaviors rather than adhering to a static throughput. It allows users to create various workload models, including random bursts and throughput fluctuations. Among the generators, only PIPO-TG provides support for this feature, while the others are restrained to generating traffic at a fixed rate.

User-defined P4 code. This feature evaluates the traffic generator's native support for a user's P4 code. It means that in addition to generating traffic with Tofino, users can direct this traffic to a P4 code running on the same device. Only PIPO-TG supports user-defined P4 since the other two generators use Tofino to generate traffic.

Stateful connections. We assess whether the traffic generator is capable of establishing stateful connections (e.g., TCP and Quick UDP Internet Connection (QUIC)) and sending traffic according to the messages it receives (e.g., sending a Syn-Ack after a Syn or responding to messages with Acks). In this feature, despite having limitations, only HyperTester is capable of establishing connections.

Open-source artifacts. We assess whether the traffic generator has artifacts available to the community. In this sense, despite having a public repository on Github, HyperTester does not make its TNA P4 codes available, while P4TG and PIPO-TG have their solutions completely open for community use.

B. Use case I: Workload alternation

The first use case is the ability of PIPO-TG to generate workload alternation patterns. This behavior may be helpful in different scenarios. For instance, we can monitor congestion control [22], such as data centers, by alternating traffic workload to assess how well the network manages congestion and prioritizes traffic. Similarly, it may be interesting to stress-test [23] in network devices to understand how it performs under different loads. In this case, a traffic alternation pattern can be valuable. Figure 4 presents two distinct square curve patterns. For both scenarios, we send alternating traffic where the user defines a lower- (100 Mbps) and upper-bound (500 Mbps). On the left (Figure 4a), we alternate the throughput every 4 seconds, while at the right (Figure 4b), we perform this modification at half the frequency – i.e., every 8 seconds. The Algorithm 2 presents the necessary code in PIPO-TG to generate both curves. We only need seven lines of code to generate the two curves using PIPO-TG (The 3 shown in the algorithm, after the 4 lines to initialize the generator and configure the port, as shown in Algorithm 1). It means there is 98.58% (or 70X) less code compared to the generated files.

TABLE IV: Qualitative validation

Characteristic	HyperTester	P4TG	PIPO-TG
Custom protocols	Yes	No	**Yes**
Number of flows	Yes	No	**Yes**
Tofino internal traffic generation	No	Yes	**Yes**
Workload generation	No	No	**Yes**
Support for user P4 code	No	No	**Yes**
Open-source artefacts	No	Yes	**Yes**
Stateful connections	Yes	No	**No**

Algorithm 2 PIPO-TG traffic alternation code snippet.

```
#courve (a) - 4 seconds
myTG.addThroughput(max=500,min=100,interval=4)
#courve (b) - 8 seconds
myTG.addThroughput(max=500,min=100,interval=8)
myTG.addIP(src="192.168.1.10",dst= "192.168.2.20")
```

C. Use case II: Burst simulation

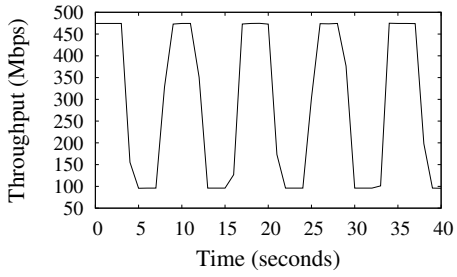
Our second use case is a burst simulation. In real network scenarios, traffic bursts may occur for several reasons – e.g., Flash crowds [24], TCP Incast scenarios [25], TCP segment offloading or application-level batch processing [26]. These bursts can cause problems like congestion, packet losses, increased latency, and others. Therefore, developing and efficiently evaluating solutions for this type of event is essential in current networks.

We demonstrate how to use PIPO-TG to simulate traffic bursts in the network. PIPO-TG allows users to define traffic bursts at specific intervals. Algorithm 3 presents the additional code necessary to generate burst traffic using PIPO-TG. In this example, we define that the bursts will be standard IP packets sent to port 5. Instead of limiting a throughput, we use the command $addVariance()$ to define that we will have a throughput of 10 Gbps for 8s, followed by 90 Gbps for 2s. It means that we will have regular traffic of 10 Gbps, and every 8s, we will have a burst of 90 Gbps lasting 2s.

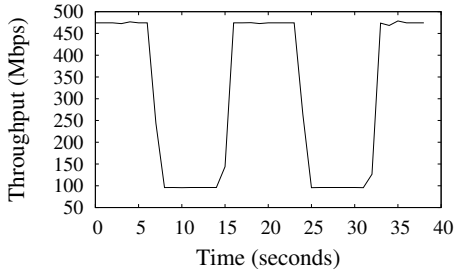
In just six lines of code, we define an experiment simulating bursts, whereas the generated PIPO-TG codes (P4 and table

Algorithm 3 PIPO-TG burst traffic code snippet.

```
#([Throughputs], [Intervals])
myTG.addVariance([10000, 90000],
                 [8, 2])
```



(a) 4-second square wave shape.



(b) 8-second square wave shape.

Fig. 4: Simulated traffic alternation.

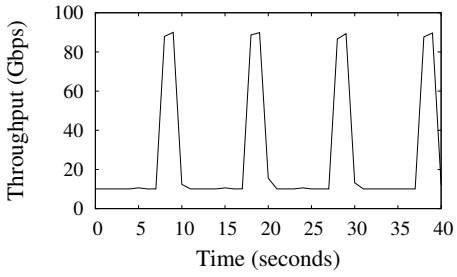


Fig. 5: Generation of network bursts every 8 seconds.

entries) for the same burst scenario consist of around 500 lines. It means that PIPO-TG reduces code effort by 98.8% (or 83.3X) for this scenario. Figure 5 shows the result of the generated bursts, with bursts of approximately 2 seconds arriving every 8 seconds on the server.

D. Use case III: DDoS simulation

The third use case involves simulating a DDoS attack scenario [27]. Strategies for modeling DDoS attacks [27], [28] can take various factors into account, such as attack distribution, including protocols, payload, and load. For instance, users can specify a pool of IP attackers targeting a single destination. In this setup, a monitoring application considers the load distribution per flow and can identify the source of attackers based on source IPs.

Algorithm 4 PIPO-TG DDoS simulation code snippet.

```
myTG.addThroughput(10000, "meter")
myTG.addIP(src="192.168.1.0",srcRandom = True,
    srcMask = 24,dst= "192.168.2.2")
```

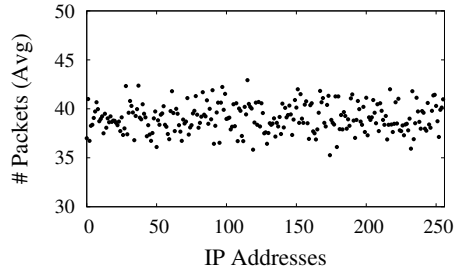


Fig. 6: Average of packets per IP address.

Algorithm 4 outlines the DDoS attack scenario. The user specifies a desired throughput, in this case, 10 Gbps, and provides a pool of IP addresses for the attackers, each with an IP base and a mask. Attackers can use a portion of the available link bandwidth to send traffic randomly. The destination IP, representing the target address for the attackers, is defined. Traffic generation starts subsequently.

While the PIPO-TG script consists of only six lines, the generated files contain around 480. This results in a remarkable code reduction of 98.5% (or 80X) using PIPO-TG. Finally, Figure 6 displays the source IP distribution in the traffic. We captured the first 10K packets in each run (30 runs) and counted the number of packets per source IP. Ideally, all IPs would have around 39 packets. The observed distribution is relatively uniform, ranging from 35.27 to 42.37 packets per IP, demonstrating PIPO-TG's efficiency in this network scenario.

VI. Final Remarks

In this work, we presented PIPO-TG, a Tofino-based traffic generator, to perform parametrizable experiments with high performance and flexibility. PIPO-TG can generate traffic with custom protocols and different throughput distributions, reaching up to 1 Tbps. In our evaluation, we explored the PIPO-TG capabilities through three key use cases: traffic alternation, burst simulation, and DDoS attack modeling. PIPO-TG demonstrated its ability to efficiently generate diverse network traffic patterns with a straightforward syntax, reducing code complexity significantly. We conducted experiments in a demanding network environment using a Tofino switch, emphasizing the distinguishing capabilities of PIPO-TG and showcasing its potential in congestion control, stress testing, and security scenarios. In future work, we have plans to extend the functionalities supported by PIPO-TG, adding support for stateful connections (e.g., TCP, QUIC), and developing a platform for monitoring the generated traffic.

REFERENCES

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

[2] K. Wiles, "Pktgen-dpdk documentation, release 22.07.2," 2023. [Acces: May 19, 2023].

[3] 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*, pp. 275–287, 2015.

[4] T. TRex, "Trex realistic traffic generator," 2023. [Acces: May 20, 2023].

[5] "Ixia traffic generator." Available on: "http://www.ixiacom.com".

[6] M. Plakalovic, E. Kaljic, and M. Mehic, "High-speed fpga-based ethernet traffic generator," in *2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT)*, pp. 1–6, IEEE, 2022.

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

[8] S. Lindner, M. Häberle, and M. Menth, "P4tg: 1 tb/s traffic generation for ethernet/ip networks," *IEEE Access*, vol. 11, pp. 17525–17535, 2023.

[9] Intel, "P416 intel tofino native architecture—public version.." https://github.com/barefootnetworks/Open-Tofino", 2021.

[10] D. Yuan, W. Yi, H. Hu, and X. Shi, "A fast, affordable and extensible fpga-based synthetic ethernet traffic generator for network evaluation," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1036–1040, IEEE, 2017.

[11] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehofe, "P4sta: High performance packet timestamping with programmable packet processors," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2020.

[12] S. Molnár, P. Megyesi, and G. Szabó, "How to validate traffic generators?," in *2013 IEEE International Conference on Communications Workshops (ICC)*, pp. 1340–1344, IEEE, 2013.

[13] M. Mortimer, "iperf3 documentation," 2018.

[14] R. Jones, "Netperf. hewlett-packard.," 1996. [Acces: June 26, 2023].

[15] *Hobbit*, "Netcat," 1995. [Acces: June 26, 2023].

[16] D. Mosberger and T. Jin, "Httperf—a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, p. 31–37, dec 1998.

[17] L. Projects, "Data plane development kit," 2023. [Acces: May 19, 2023].

[18] P. Biondi, "Scapy," 2011. [Acces: June 26, 2023].

[19] M. Pall, "Luajit," 2023. [Acces: June 20, 2023].

[20] L. C. de Almeida, J. L. da Silva, R. P. Lins, P. D. Maciel Jr, R. Pasquini, and F. L. Verdi, "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*, pp. 9–16, SBC, 2023.

[21] F. Costa, F. Vogt, A. Castro, F. Rodriguez, M. Luizelli, and C. Rothenberg, "Pipo-tg: Parametrizable high performance traffic generatior." https://github.com/FilipoGC/PIPO-TG, 2023.

[22] A. Balador, E. Cinque, M. Pratesi, F. Valentini, C. Bai, A. A. Gómez, and M. Mohammadi, "Survey on decentralized congestion control methods for vehicular communication," *Vehicular Communications*, vol. 33, p. 100394, 2022.

[23] G. Soós, F. N. Janky, and P. Varga, "Distinguishing 5g iot use-cases through analyzing signaling traffic characteristics," in *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 562–565, IEEE, 2019.

[24] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing flash crowds on the internet," in *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.*, pp. 246–249, IEEE, 2003.

[25] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, pp. 63–74, 2010.

[26] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter, "Bullet trains: A study of nic burst behavior at microsecond timescales," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 133–138, 2013.

[27] K. Vanitha, S. V. UMA, and S. Mahidhar, "Distributed denial of service: Attack techniques and mitigation," in *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, pp. 226–231, Dec 2017.

[28] A. Cetinkaya, H. Ishii, and T. Hayakawa, "An overview on denial-of-service attacks in control systems: Attack models and security analyses," *Entropy*, vol. 21, no. 2, p. 210, 2019.