UNIVERSIDADE FEDERAL DO PAMPA

Filipo Gabert Costa

PIPO-TG: Parameterizable High Performance Traffic Generation

Alegrete
2023

Filipo Gabert Costa

PIPO-TG: Parameterizable High Performance Traffic Generation

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Caggiani Luizelli

Co-orientador: Me. Francisco Germano Vogt

Alegrete
2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

C837p Costa, Filipo
PIPO-TG: Parameterizable High Performance Traffic Generation / Filipo Costa.
63 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, CIÊNCIA DA COMPUTAÇÃO, 2023.
"Orientação: Marcelo Luizelli".

1. Traffic Generation. 2. P4. 3. Computer Networks. 4. Experiments generation. 5. Intel Tofino. I. Título.

FILIPO GABERT COSTA

PIPO-TG: PARAMETERIZABLE HIGH PERFORMANCE TRAFFIC GENERATION

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computaçaõ da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 07 de dezembro de 2023.

Banca examinadora:

_____

Prof. Dr. Marcelo Caggiani Luizelli
Unipampa

_____

Me. Francisco Germano Vogt
Unicamp

_____

Prof. Dr. Fábio Diniz Rossi
IFFAR

_____

Prof. Dr. Weverton Luis da Costa Cordeiro

UFRGS

Assinado eletronicamente por **MARCELO CAGGIANI LUIZELLI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 14/12/2023, às 13:34, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Fábio Diniz Rossi**, **Usuário Externo**, em 14/12/2023, às 13:35, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Weverton Luis da Costa Cordeiro**, **Usuário Externo**, em 14/12/2023, às 13:43, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Francisco Germano Vogt**, **Usuário Externo**, em 14/12/2023, às 18:46, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1330395** e o código CRC **A8C49C7C**.

This work is dedicated to my family, friends and everyone who somehow participated in my academic trajectory, especially to my friends i made in academy, whom i will remember forever, and whom i hope to always see throughout my life.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my family, my father and mother, who did everything possible for me to achieve this dream and complete this stage in my life. They shaped me as a person and made me who I am today. I want to express my gratitude to my brother, Cassiano, with whom I shared not only the same home for many years but also moments that I will never forget and will carry with me throughout my life.

In addition to my family, I would like to thank Professor Dr. Marcelo Caggiani Luizelli, who always supported me, provided numerous opportunities, and taught me a lot within the academic landscape.

I also want to express my gratitude to the friends I made during my college journey, especially Francisco Vogt and Rafael Dalosto, with whom I have shared experiences since the beginning of college until today. A special thanks to Francisco and Ariel Goes for always believing in me and my potential, giving me countless opportunities. From the bottom of my heart, I am grateful to them.

Not least, I want to thank many other friends who have been part of the journey so far, with whom I shared numerous happy moments that I will always cherish. Lauriano, Kalew, Yuri, Pregui, Takeshi, Sandro, Douglas, Diego, Victor - each of them holds a special place in my memories. I may have forgotten to mention someone, but they know how important they are to me.

"Enjoy life today, for yesterday is gone and tomorrow may never come." - Alan Watts

ABSTRACT

In recent years, the demand for network resources by applications has seen a substantial increase due to requirements such as minimum latency for real-time applications (e.g., VoIP, gaming) and minimum bandwidth for data-intensive activities (e.g., VR, video streaming). As computer networks continue to expand in size and complexity, researchers and professionals face intricate challenges in managing and optimizing network performance. In this context, the generation of network traffic plays a pivotal role. Accurate and realistic traffic generation enables comprehensive assessments of network performance, efficiency, and security. By simulating real-world scenarios and traffic patterns, researchers can gain valuable insights into network behavior and evaluate the effectiveness of protocols, algorithms, and security measures. Traffic generation serves as a fundamental tool for advancing the field of computer networks, facilitating experimentation, and enabling the development of innovative solutions to meet the evolving demands of modern network infrastructures. This research focuses on the creation of PIPO-TG, a traffic generator specifically designed for the Tofino Switch. Powered by the P4 programmable data plane technology, PIPO-TG offers customizable packet forwarding on the Tofino architecture, ensuring accurate performance evaluations without bottlenecks or distortions. The primary objective of PIPO-TG is to generate is to develop a highly customizable traffic generator that can generate realistic and diverse traffic patterns, including the emulation of network anomalies and behavior patterns at a 100Gb/s per port, 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.

Key-words: Traffic Generation. P4. Computer Networks. Experiments generation. Intel Tofino.

RESUMO

Nos últimos anos, a demanda por recursos de rede por parte de aplicativos tem experimentado um aumento substancial, devido a requisitos como latência mínima para aplicativos em tempo real (por exemplo, VoIP, jogos) e largura de banda mínima para atividades intensivas em dados (por exemplo, VR, transmissão de vídeo). À medida que as redes de computadores continuam a se expandir em tamanho e complexidade, pesquisadores e profissionais enfrentam desafios complexos na gestão e otimização do desempenho da rede. Nesse contexto, a geração de tráfego de rede desempenha um papel crucial. A geração precisa e realista de tráfego permite avaliações abrangentes do desempenho, eficiência e segurança da rede. Ao simular cenários e padrões de tráfego do mundo real, os pesquisadores podem obter insights valiosos sobre o comportamento da rede e avaliar a eficácia de protocolos, algoritmos e medidas de segurança. A geração de tráfego serve como uma ferramenta fundamental para avançar no campo de redes de computadores, facilitando experimentação e permitindo o desenvolvimento de soluções inovadoras para atender às demandas em constante evolução das infraestruturas de rede modernas. Esta pesquisa se concentra na criação do PIPO-TG, um gerador de tráfego especificamente projetado para o Tofino Switch. Alimentado pela tecnologia de plano de dados programável P4, o PIPO-TG oferece encaminhamento de pacotes personalizável na arquitetura Tofino, garantindo avaliações precisas de desempenho sem gargalos ou distorções. O principal objetivo do PIPO-TG é desenvolver um gerador de tráfego altamente personalizável que possa gerar padrões de tráfego realistas e diversos, incluindo a emulação de anomalias de rede e padrões de comportamento a 100Gb/s por porta, permitindo que os pesquisadores avaliem o desempenho da rede sob condições variáveis, fornecendo encaminhamento de pacotes personalizável com planos de dados programáveis P4. Nossas principais contribuições incluem personalização de cabeçalho de pacote definida pelo usuário e código aberto para reprodutibilidade. Esses esforços promovem a colaboração dentro da comunidade de pesquisa para avançar nas técnicas de geração de tráfego. Mostramos que o PIPO-TG requer apenas algumas linhas de código para simular cenários de rede heterogêneos (por exemplo, *bursts* de tráfego e ataques DDoS), mantendo o desempenho e a flexibilidade do hardware.

Palavras-chave: Geração de tráfego. P4. Redes de Computadores. Geração de experimentos. Intel Tofino.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

APIs Application Programming Interfaces

ARP Address Resolution Protocol

ASIC Application-Specific Integrated Circuit

ASICs Application-Specific Integrated Circuits

CPU Central Processing Unit

DDoS Distributed Denial of Service

DLPI Data Link Provider Interface

DPDK Data Plane Development Kit

FPGA Field-programmable Gate Array

FPGAs Field-programmable Gate Arrays

GPLv2 GNU General Public License version 2

GRE Generic Routing Encapsulation

IATs Inter-arrival times

ICMP Internet Control Message Protocol

IP Internet Protocol

JIT just-in-time

lpm longest prefix matching

MAC Media Access Control

MAT Matching Action Table

MATs Matching Action Tables

MPLS Multi Protocol Label Switching

NICs Network Interface Cards

NTAPI Network Testing API

P4 Programming Protocol-independent Packet Processors

PCI Peripheral Component Interconnect

Pktgen Packet Generator

POF Protocol Oblivious Forwarding

QoS Quality of Service

QUIC Quick UDP Internet Protocol

REPL Read-Eval-Print Loop

RTTs round-trip times

SDK Software Development Kit

SDN Software-Defined Network

TCP Transmission Control Protocol

TNA Tofino™ Native Architecture

TNA's Tofino™ Native Architecture

UDP User Datagram Protocol

VHDL VHSIC Hardware Description Language

VoIP Voice over Internet protocol

VR Virtual Reality

WAVE Workload Assay for Verified Experiments

CONTENTS

1 INTRODUCTION

In this chapter, we will provide an in-depth exploration of the context and motivation surrounding the integration of a traffic generator with the Tofino switch(INTEL, 2021a). By examining the current networking landscape and identifying the challenges and limitations faced by traditional traffic generators, we aim to establish a clear rationale for our research. Furthermore, we will outline the objectives of this work and present the contributions it brings to the field.

1.1 Context and Motivation

In recent years, the demand for network resources by applications has increased((ITU), 2022) due to requirements such as (i) minimum latency (e.g., Voice over Internet protocol (VoIP), gaming) and (ii) minimum bandwidth (e.g., Virtual Reality (VR), video streaming). The size and complexity of computer networks have continuously expanded, presenting ever more intricate challenges for researchers and professionals (KHATIB et al., 2023).

The advent of Software-Defined Network (SDN) and network programmability with languages like Programming Protocol-independent Packet Processors (P4) has significantly transformed network monitoring strategies and management. This evolution empowers network operators to devise solutions capable of execution within the network, leveraging cutting-edge technologies such as the Tofino switch. However, it is imperative that these solutions undergo rigorous evaluation in critical scenarios to assess their robustness, precision, and accuracy. Thorough testing under stress conditions is essential to ensure their reliability and effectiveness in real-world applications. This validation process becomes pivotal in guaranteeing the seamless integration and optimal performance of these innovative solutions within diverse network environments.

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 (LIU et al., 2022). 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 (WILES, 2023; EMMERICH et al., 2015; TREX, 2023) 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 (IXIA..., ; PLAKALOVIC; KALJIC; MEHIC, 2022) 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 (ZHOU et al., 2019; LINDNER; HÄBERLE; MENTH, 2023) 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(ZHOU et al., 2019) needs an auxiliary CPU to create its packets and is not completely open-source, P4TG(LINDNER; HÄBERLE; MENTH, 2023) 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.

Solving all 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 limitations. Tofino's native traffic generation unit does not support resolving these limitations, so we must resolve them through the P4 code. In turn, it has restrictions such as instruction limits and does not support complex comparisons, floating point operations, and loops.

1.2 Objectives and Contributions

It is within this context that PIPO-TG is positioned, focusing on the creation of a traffic generator specifically designed for the Tofino Switch. Powered by the P4 programmable data plane technology, the PIPO-TG offers the capability to customize and forward packets on the Tofino™ Native Architecture (TNA) architecture with line-rate packet generation, ensuring accurate performance evaluations without introducing bottlenecks or distortions. By leveraging P4 programmability, our goal is to develop a traffic generator that can generate realistic and diverse traffic patterns. It incorporates the ability to emulate various network anomalies and behavior patterns. By simulating scenarios such as network congestion, link failures, packet drops, and other disruptive events, PIPO-TG enables researchers to assess the resilience and performance of network systems under realistic and challenging conditions enabling comprehensive testing and evaluation of network systems.

The PIPO-TG, a parameterizable and high-performance traffic generation solution is built on top of the TNA (INTEL, 2021c) and relies on P4 language to describe and customize packet generations on the TNA architecture. PIPO-TG can generate network traffic up to 1Tbps line rate, ensuring accurate performance evaluations without introducing bottlenecks or distortions. PIPO-TG introduces support for arbitrary traffic patterns using a high-level

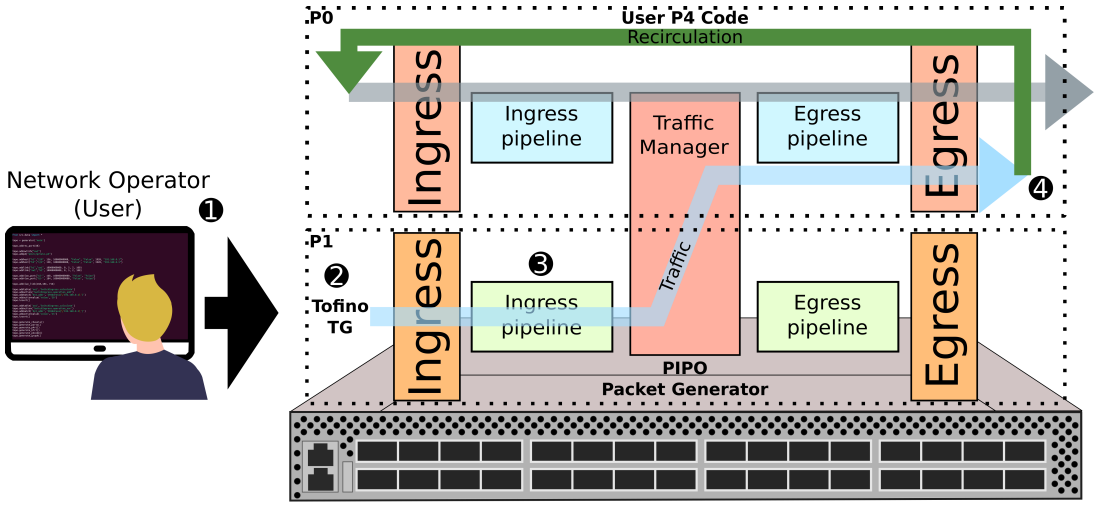


Figure 1 – Overview of PIPO-TG traffic generation in TNA.

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 generation process: ❶ users set the traffic generation 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 the designated physical port.

1.3 Main Contributions

Considering the proposed objectives, the main contributions of our work are summarized as follows:

- A hardware-based traffic generator for line-rate traffic generation;
- A user-friendly 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

1.3.1 Research Contributions

Next, we describe the work's academic contributions, including works submitted to national and international conferences and open-source artifacts. The work has been submitted and will be presented (if accepted) at the following conferences:

1. **Filipo Gabert Costa**, Francisco Germano Vogt, Fabricio Rodriguez, Ariel Góes De Castro, Marcelo Caggiani Luizelli, and Christian Rothenberg. “PIPO-TG: Parameterizable High-Performance Traffic Generation”. **Submitted in:** *IEEE/IFIP Network Operations and Management Symposium (NOMS)* Seoul, South Korea, 2024.

2. **Filipo Gabert Costa**, Francisco Germano Vogt, Fabricio Rodriguez, Ariel Góes De Castro, Marcelo Caggiani Luizelli, and Christian Rothenberg. “PIPO-TG Unboxed: Crafting Tailored Traffic Generation for Industrial URLLC Realities”. **To be submitted in:** *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)* Rio de Janeiro, Brazil, 2024.

Furthermore, the work produced a completely open-source tool for generating traffic, which can be found in the following repository:

1. **PIPO-TG traffic generator.** <https://github.com/FilipoGC/PIPO-TG>

1.4 Outline

The remainder of this work is organized as follows. Chapter 2 discusses our background topics and the related works. In Chapter 3, we introduce the PIPO-TG architecture, features, and implementation. Chapter 4 presents and discusses an evaluation of the proposed approach in three use cases and the results. Last, in Chapter 5, is to conclude the work with final remarks and perspectives for future work.

2 BACKGROUND AND RELATED WORK

This chapter presents an overview of the background and related works relevant to the research presented in this work. The chapter starts by discussing the background of the research, followed by a review of the related works in the field.

2.1 Traffic Generation

In computer networking, traffic generation is the process of producing artificial traffic that mimics the behavior of real-world network traffic for testing and evaluating network devices, protocols, and applications. Traffic generation plays a crucial role in the development, testing, and validation of modern networks, especially in the context of data centers, cloud computing, and software-defined networking.

Network traffic generators are essential tools in the design, development, and management of modern networks. With increasing network complexity and limitations on replaying measured traces, synthetic traffic generation has become crucial. These generators inject controlled packets into the network, capturing the characteristics of actual traffic. However, despite the wide range of available traffic generators, there is still a lack of consensus on validation methods and metrics for assessing their accuracy(MOLNÁR; MEGYESI; SZABÓ, 2013). Further research is needed to establish standardized evaluation approaches and enhance the effectiveness of traffic generators in various network scenarios. When comparing hardware and software-based traffic generators, it is important to consider the financial implications of investing in high-performance resources.Table[1] provides a comprehensive overview of the key factors to consider in this comparison. It highlights the advantages and limitations of each approach, including the cost-effectiveness of software-based solutions and the specialized design and accuracy of hardware-based solutions. By analyzing these factors, researchers and practitioners can make informed decisions about which type of traffic generator is most suitable for their specific needs.

In the broader context of traffic generation, the artificial traffic produced is instrumental for testing and evaluating network devices, protocols, and applications in various contexts, such as data centers, cloud computing, and software-defined networking. Traffic generators are specifically designed to inject controlled packets into the network, offering a high degree of flexibility. This flexibility enables network engineers to replicate traffic behavior, simulate specific events, and recreate desired traffic patterns. Whether emulating heavy data transfers of a file transfer protocol, intermittent bursts of a VoIP call, or continuous flow of web traffic, these generators provide the versatility needed to recreate a wide array of network scenarios.

Traffic generators can be categorized into two groups: software-based and hardware-based. Software-based generators rely on software applications to simulate and generate network traffic, while hardware-based generators use specialized hardware components for this purpose. The choice between these two approaches involves careful consideration of

critical factors, as outlined in Table 1. This table presents the benefits of traditional software or hardware traffic generation techniques and compares them with the assistance of PIPO-TG. The decision between hardware and software-based generators ultimately depends on specific requirements and objectives, empowering engineers to optimize network configurations effectively and ensure optimal performance under diverse conditions.

In this section, we will subdivide traffic generators into two very distinct groups: software-based traffic generators and hardware-based traffic generators. This division is based on the way traffic is generated, where software-based traffic generators rely on software to generate traffic, while hardware-based traffic generators use specialized hardware to generate traffic.

Table 1 – Comparison of SW/HW traffic generation.

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

- **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.

- **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.

- **Resources:** Resources encompass the capabilities and functionalities offered by a traffic generator. Due to their dedicated hardware components, hardware-based genera-

tors often provide advanced features and capabilities. Software-based generators can vary widely regarding available features, depending on the specific software used and its capabilities.

- **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.

2.1.1 Software-based

Traffic generators implemented on software platforms offer numerous advantages for network evaluation and testing. These platforms, often developed by research units or universities, are known for their cost-effectiveness and flexibility. They provide researchers with the ability to easily deploy multiple nodes, enabling the replication of distributed scenarios, even with a large number of nodes. Moreover, software traffic generators are often open-source and freely available, making them accessible to a wide range of users.

One significant advantage of software traffic generators is the ease of code modification and extension. Researchers can adapt the generator to their specific needs by adding new functionalities, implementing custom traffic models, and integrating support for various operating systems and hardware platforms. This flexibility empowers researchers to conduct experiments that align closely with real-world scenarios and test actual network implementations. However, it is important to acknowledge that software generation faces challenges that can impact experiment accuracy. The generated traffic may deviate from the operator's intentions, leading to potential issues and inaccuracies in the evaluation process. Unlike hardware generation, where detailed datasheets with certified information are available, software platforms cannot provide the same level of information regarding confidence intervals and imposed values such as bit rate(BOTTA; DAINOTTI; PESCAPÉ, 2010).

Software traffic generation is a valuable approach that offers flexibility, cost-effectiveness, and accessibility for network evaluation and testing. By leveraging the capabilities of software platforms, researchers can conduct realistic experiments and customize the traffic generation process to suit their specific research objectives. However, it is crucial to be aware of the challenges and limitations associated with software generation to ensure accurate and reliable evaluations. Additionally, software network testers offer the greatest level of flexibility among all types of network testers. However, optimizing the performance of a software network tester requires significant development efforts. Furthermore, since the Central Processing Unit (CPU) is a universal computing platform, it has inherent limitations on packet generation throughput. To meet the demand for higher throughput, network testing tasks require additional CPU cores, resulting in linearly increased equipment and power costs.

2.1.2 Hardware-based

Hardware-based traffic generators, on the other hand, offer superior accuracy and performance capabilities compared to software-based alternatives. These platforms, typically provided by commercial vendors, are known for their precision and reliability. While they may be more expensive and utilize closed-source technologies, they deliver robust and precise traffic generation.

One advantage of hardware traffic generators is their ability to provide detailed data sheets. These data sheets contain certified information, including confidence intervals for the imposed values, ensuring precise and reliable traffic generation(EMMERICH et al., 2017). This makes hardware-based traffic generators well-suited for demanding testing and evaluation scenarios that require precise control over traffic characteristics. Despite the higher initial invesent, the benefits of using hardware-based traffic generators, such as improved precision and performance, make them a valuable choice for organizations and researchers aiming to conduct thorough and accurate network performance evaluations.

2.2 P4

As the P4 language plays a fundamental part in the development of our traffic generator, in this section we present its structure and main components. Traditional networking devices such as routers and switches are limited in their programmability capabilities, with vendors having control over the underlying algorithms, while network operators struggle to detect network anomalies - e.g., misconfiguration. With the growing, adoption of SDN concepts and the emergence of network programmable languages such as P4(BOSSHART et al., 2014) and Protocol Oblivious Forwarding (POF)(SONG, 2013), were emerging for defining and implementing custom algorithms. With the increasing adoption of SDN concepts and programmable switches, operators can define only the data forwarding functionalities directly in the data plane, while the control logic remains separated. This brings a set of benefits: (i) by decoupling the control plane from the data plane, network operators can define routing and monitoring algorithms independently of the manufacturer; (ii) the internal pipeline of forwarding devices can be entirely defined by the operator, allowing customization of parser elements, headers, and match+action processing logic with tables - also defined by the operator.

For example, with programmable switches, network operators can replace embedded control plane algorithms with self-defined ones in SDN, resulting in simplified complex algorithms, enhancing flexibility, efficiency, and security in use cases like data centers or 5G networks. Programmable data planes offer the ability to implement custom data plane algorithms and define new protocol headers and forwarding behaviors, utilizing various programming models like P4. P4 is the most widely used programming language and concept for data plane programming.

2.2.1 Introduction

P4 is a high-level language for programming protocol-independent packet processors that allows users to define how data packets are processed in network devices. It separates the packet processing logic from the underlying hardware infrastructure, allowing users to program the behavior of network devices in a more flexible and customized way than with traditional technologies.

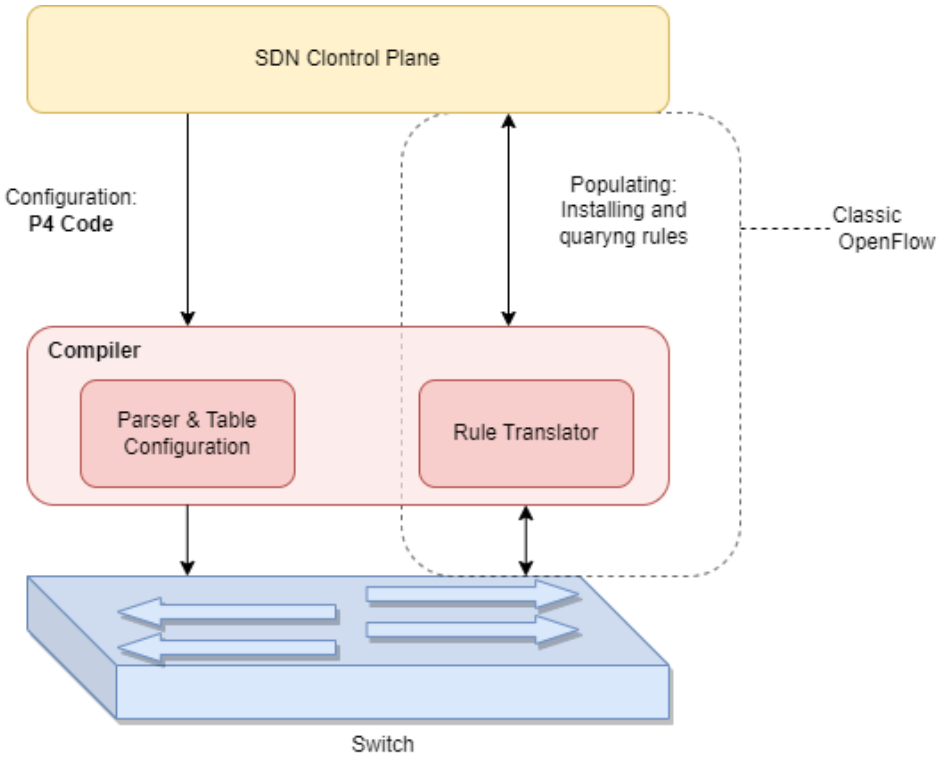


Figure 2 – P4 & OpenFlow

It defines the behavior of data planes and operates with SDN interfaces such as OpenFlow or P4Runtime. It provides networking programmers with the capability to precisely define the packet processing mechanisms in forwarding devices, such as switches or Network Interface Cards (NICs), by writing code that is compiled into low-level instructions for a range of networking targets, including Field-programmable Gate Arrays (FPGAs), SmartNICs, and software switches. The P4 forwarding model exhibits slight differences compared to the OpenFlow data plane [2]. The accompanying diagram demonstrates the distinctions between P4 and OpenFlow and illustrates the relationship between P4, utilized for switch configuration and specification of packet processing rules, and existing Application Programming Interfaces (APIs) like OpenFlow, which are specifically designed for populating forwarding tables in fixed-function switches.

With P4, switches can forward incoming packets using multiple stages of match/action, and the language supports a programmable parser to allow for new headers and protocols. The P4 forwarding model can be described in two types of operations: configure

and populate. During the configure operation, the parser is programmed, and the order of match/action stages is organized, specifying which header fields are processed in each stage. During the populate operation, table entries are added and/or removed, according to the parser, and inserted into the match/action tables specified during configuration. One of the key features of P4 is its protocol independence, which means that it can be used with any network protocol. As stated in the paper P4: Programming Protocol-Independent Packet Processors, P4 enables one to express arbitrary, packet-by-packet forwarding behaviors, using a high-level, domain-specific language (BOSSHART et al., 2014). This flexibility is particularly useful for researchers and developers who need to test and prototype new network protocols or functions.

Another significant feature of P4 is its ability to separate the data plane and the control plane. According to Bosshart et al. (2014) (BOSSHART et al., 2014), P4 effectively segregates the forwarding (data plane) and control (control plane) functions, making the forwarding plane fully programmable. This separation allows for more efficient and dynamic control of network traffic, as the data plane can be modified and updated without disrupting the control plane. P4 also provides the ability to specify forwarding rules at a fine-grained level. As described in (BOSSHART et al., 2014), P4 allows users to write expressive and precise forwarding policies (BOSSHART et al., 2014). This fine-grained control enables network administrators to implement specific policies and rules for network traffic, such as Quality of Service (QoS) policies or traffic engineering.

To summarize, P4 is a powerful language that provides network administrators and developers with the flexibility and customization needed to efficiently manage and control network traffic. Its protocol independence, separation of the data and control planes, and fine-grained forwarding rules make it a valuable tool for researchers, developers, and network administrators.

2.2.2 Control Program

The control program is a crucial component of the P4 language that allows users to define the behavior of the network devices. According to the official P4 language specification, the control program describes the actions to take for each incoming packet based on the packet's headers and any other contextual information that the user wishes to include (FOUNDATION, 2019). This means that the control program can be used to specify how network devices should handle different types of traffic, such as prioritizing certain types of packets or dropping packets that match specific criteria.

One of the main benefits of the control program in the P4 language is its flexibility. Users can define their control programs to meet specific networking requirements rather than relying on pre-defined protocols. As noted in a paper on P4 language, with P4, users can define the control plane programmatically, opening up the possibility of new, customized protocols(BOSSHART et al., 2014). This flexibility allows users to optimize the behavior of

their network devices for their specific use case, resulting in improved performance and efficiency.

In addition to its flexibility, the control program in P4 language also enables network operators to make real-time changes to the behavior of their network devices. As noted in a white paper on P4 language, P4 provides rapid innovation in networking by enabling the programmability of the control plane. It provides the flexibility to adapt to evolving network requirements and conditions, facilitating quick and efficient network adaptation (FOUNDATION, 2019). This means that network operators can modify the control program on-the-fly to respond to changing network conditions, rather than waiting for pre-defined protocols to be updated or replaced. This can result in faster response times and improved network reliability.

2.2.3 Multiple Pipelines

The P4 pipelines can adapt to suit each architecture. In the case of the Tofino™ Native Architecture (TNA's) P4 pipeline, as illustrated in Figure[4], every physical port has the capability to transmit and receive packets. Hence, the P4 pipeline of the Intel Tofino is divided into two main sections: ingress and egress.

Incoming packets received by an input port undergo processing in the programmable ingress parser. This step involves extracting and storing packet headers for future use within the ingress control, such as matching in Matching Action Table (MAT). Subsequently, the packet is processed by the ingress control, undergoing multiple matches against user-defined Matching Action Tables (MATs) and potential header manipulation.

At the ingress control, the process of matching a packet against a Match Action Table that can be visualization in Figure[3], MATs in P4 utilize lookup keys that are formed using packet metadata, which are then used to perform row matching within the MAT. In case of a hit, the defined action is applied along with the specified action data. Instead, In case of a miss, the default action is applied.

The Lookup key is formed, comprising specific header and/or metadata fields extracted from the packet. This lookup key serves as the basis for comparison with the stored keys within the MAT. The comparison is performed based on a predetermined match type, with the P4 core library defining three standard match types: exact, ternary, and longest prefix matching (lpm). Each component of the lookup key undergoes a comparison operation, ensuring that the packet's attributes align with the stored key using the specified match type.

During the ingress processing, the packet's destination is determined, which involves selecting an egress port. At the end of the ingress section, the packet is serialized through the ingress deparser, emitting the headers according to the user-defined ingress deparser. Following this, the packet enters the traffic manager, which can replicate the packet if needed, such as in the case of multicast. The packet is then queued for the chosen egress port and later queued based on Intel Tofino's underlying scheduling strategy. It is then sent to the

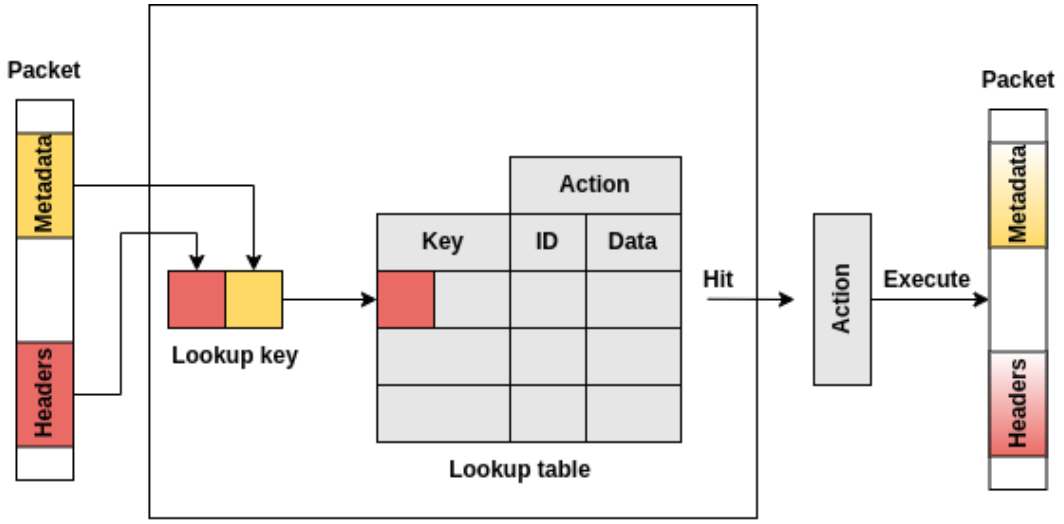


Figure 3 – Match-action Table(MAT) in P4

egress parser.

Similar to the ingress counterparts, the egress parser, egress control, and egress deparser perform similar operations. Finally, the packet is transmitted through the specified output port or may be placed back into the ingress section of the P4 pipeline if the port is configured as a recirculation port.

2.3 Tofino Native Architecture (TNA)

TNA is a design philosophy created by Barefoot Networks, a company that was acquired by Intel in 2019 for its Tofino family of network processing units. The TNA approach is based on the idea of making the hardware programmable, which allows the software to control the behavior of the device, making it easier to customize the network processing pipeline. The Tofino chip is designed to be programmable in a way that is completely different from traditional Application-Specific Integrated Circuits (ASICs), and Barefoot Network Tofino Software Development Kit (SDK) and P4 language are the keys to unlocking its potential(INTEL, 2021b).

The Tofino chip was specifically designed to work with P4 and the P4 Runtime interface, which allows network operators to dynamically configure and control the forwarding plane of the device. This makes it possible to tailor the behavior of the network to the specific needs of the application, improving performance and reducing latency.

The traffic manager is an important component of the Intel Tofino chip and plays a key role in managing the flow of network traffic. The traffic manager is responsible for scheduling packets and managing the queues in the switch. Tofino switch architecture consists of a set of ingress pipelines[4], which parse packets and extract metadata, and an egress pipeline, which takes the results of the ingress processing and decides where the packet should be sent next(INTEL, 2021b).

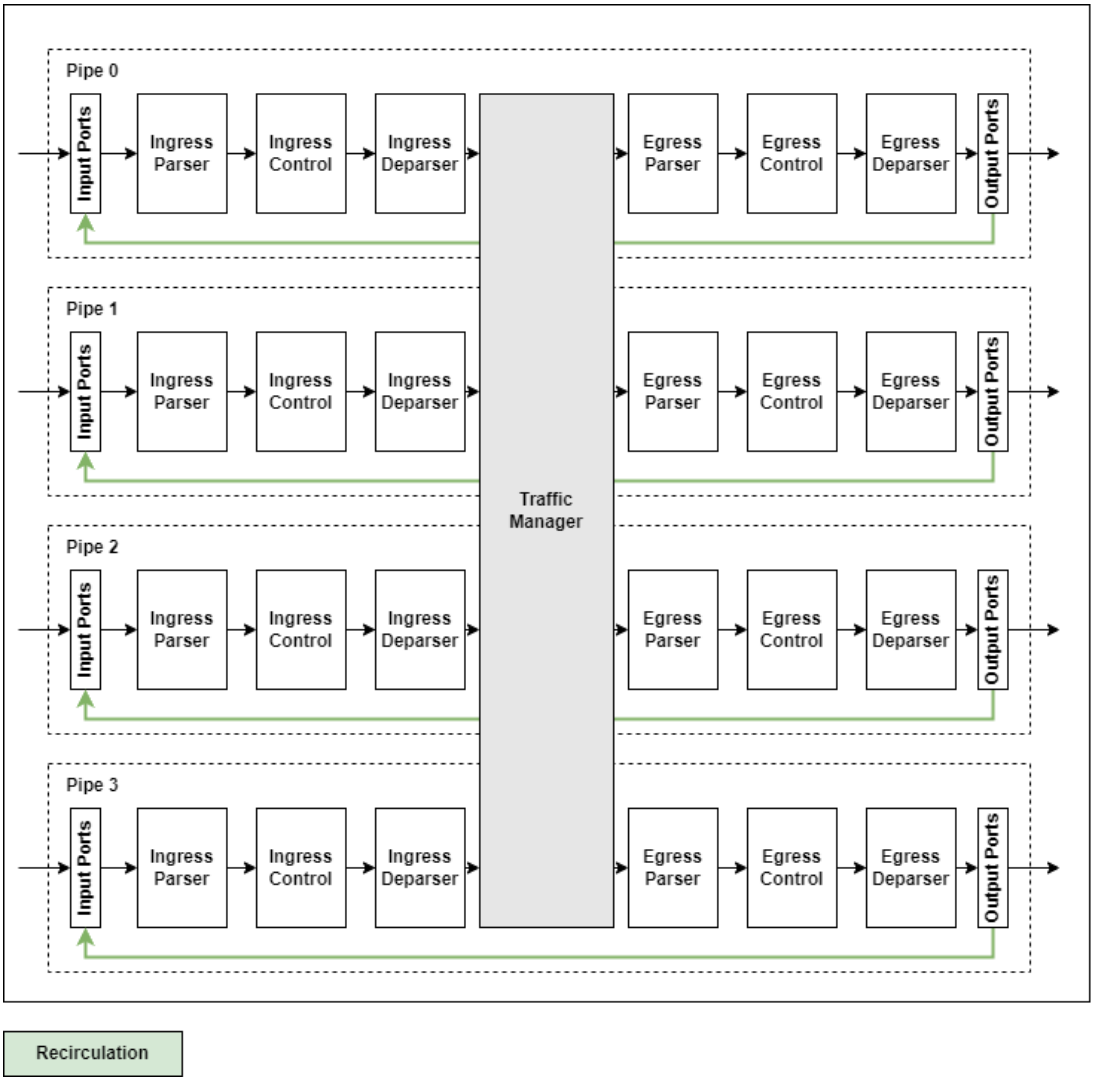


Figure 4 – Simplified version of P4 Pipelines of the Tofino Native Architecture

In Figure 4, the P4 pipeline in the TNA showcases the relationship between the ingress pipelines and the traffic manager, followed by the egress pipelines, ultimately leading to recirculating. One of the benefits of the TNA approach is the ability to update the software running on the device without having to make changes to the underlying hardware. This makes it easier to add new features and functionalities to the network, reducing the time and cost of deploying new services. Tofino is also designed to be fully programmable, so it can be updated with new features and functionality as needed without requiring any changes to the hardware. Overall, TNA and the use of P4 enable network operators to have more control over the behavior of their networks, allowing them to tailor the network to the specific needs of their applications.

2.3.1 Traffic Generation with Tofino

Traffic generation using the Tofino switch has been explored in some studies, such as (LINDNER; HÄBERLE; MENTH, 2023), (RODRIGUEZ et al., 2022), and (LAI et al., 2019). The

switch's packet generator is a versatile component that offers a range of functionalities for efficient packet generation in our work. It operates through the pipeline interface, specifically utilizing the pipe port 68 for seamless communication and data transfer. With its eight independent generators or applications, the packet generator enables the simultaneous execution of various packet-generation tasks.

The packet generator's capabilities are further enhanced by its data plane triggers. Each application has its trigger mechanism, enabling precise control over packet generation. The trigger options include timer-based triggers, which can be configured as periodic or one-shot timers. This allows for the generation of packets at regular intervals or for specific durations. Additionally, the packet generator supports port-down triggers. When a port becomes unavailable or inactive, it triggers the packet generator to generate specific packets or perform predefined actions. This functionality adds resilience and adaptability to the packet generation process, enabling efficient testing and evaluation of network scenarios.

Another unique capability of the packet generator is its ability to trigger packet generation based on the first four bytes of a recirculated packet. This feature enables the packet generator to respond or generate packets based on the content of the recirculated packets, enhancing its flexibility in handling complex network scenarios. Overall, the packet generator in the Intel Tofino switch provides researchers and network engineers with a powerful tool for generating and testing packets. Its diverse range of triggers, independent generators, and packet data buffers ensure accurate and efficient packet generation, making it an invaluable component in network testing and evaluation(FOUNDATION, 2021).

2.4 Workload Assay

One of the primary objectives of traffic generators is to create a realistic network scenario. As we know, computer networks can undergo slight variations and encounter various possible scenarios, *e.g.* bursts, ON-OFF patterns, mobile broadband fluctuations, or oscillations. To design experiments that closely resemble real-world conditions, the generation of workloads for traffic generators becomes crucial. This allows researchers to define load models to be generated based on mathematical functions. For instance, WAVE(Workload Assay for Verified Experiments)(ALMEIDA et al., 2023) proposed the *sinusoidal* load model based on the *sine* function, resulting in periodic load behavior. By incorporating such load models, researchers can achieve more accurate and representative simulations of network traffic.

Developing a thorough understanding of the generated workload is crucial for an accurate analysis of results in scientific experiments. In the context of computer network research, synthetic traffic generators are commonly used. However, these generators often fail to accurately portray the behaviors of real-world applications. This is where the Workload Assay for Verified Experiments (WAVE) comes into play.

2.5 Related Work

In the realm of computer networking, traffic generation refers to the creation of network traffic that imitates real-world behavior. This practice is essential for testing and evaluating network devices, protocols, and applications in data centers, cloud computing, and software-defined networking. 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 (MOLNÁR; MEGYESI; SZABÓ, 2013), 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 (refer to the table for details). This analysis assists professionals in selecting the traffic generator that suits their requirements.

In this context of traffic generation, various software-based and hardware-based approaches have been developed. This section provides an overview of existing research in both categories. In the field of network traffic generation, open-source alternatives to expensive proprietary solutions have become increasingly popular among universities and the research community. These open-source traffic generators provide cost-effective or even free options that offer high flexibility. (ADELEKE; BASTIN; GURKAN, 2022) Surveyed over 7000 papers related to computer networking published between 2006 and 2018 (13 years). From these papers, they identified and listed the top 92 traffic generators based on their usage. We will now discuss the methods traditionally used by the scientific community, focusing on the first five in the list proposed by (ADELEKE; BASTIN; GURKAN, 2022), excluding Moongen (EMMERICH et al., 2015) and Pktgen (WILES, 2023), which will be introduced in the next subsection. One of them *e.g.*, Iperf2 (MCMAHON; AUCKLAND, 2005) is a widely utilized open-source tool that allows users to measure bandwidth and network performance. It supports various protocols and provides both client and server modes for generating and measuring traffic.

Recently, both software- and hardware-based approaches have been developed in the context of traffic generation, mainly used for academic evaluation. More specifically, open-source traffic generators provide cost-effective – or even free – options with high flexibility regarding traffic configuration.

Netperf (JONES, 1996) is a versatile benchmark tool designed to evaluate different aspects of network performance. It primarily focuses on measuring bulk data transfer and request/response performance using Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) protocols and the Berkeley Sockets interface. While its main features revolve around these areas, Netperf also supports tests for Data Link Provider Interface (DLPI) and Unix Domain Sockets, and IPv6 tests can be included based on the conditional compilation.

The primary goal of httperf (MOSBERGER; JIN, 1998) is not to focus on a specific benchmark but rather to offer a reliable and high-performance tool that enables the cre-

ation of both micro- and macro-level benchmarks. The distinguishing features of httperf include its robustness, allowing it to generate and sustain server overload, its support for the HTTP/1.1 protocol, and its extensibility for incorporating new workload generators and performance measurements. In addition to describing the design and implementation of httperf, their paper also shares the experiences and insights gained during the development of the tool.

Scapy (BIONDI, 2011) is a robust Python library that offers powerful capabilities for interactive packet manipulation. It allows users to create, decode, send, and capture packets across a wide range of protocols. With Scapy, it is possible to match requests and replies, forge packets, and perform various network-related tasks. It serves as both a Read-Eval-Print Loop (REPL) for interactive use and a library for integrating custom network layers. Additionally, Scapy is cross-platform, offering native support for Linux, macOS, most Unix-based systems, and Windows with Npcap. It is released under the GNU General Public License version 2 (GPLv2) license, and, starting from version 2.5.0+, it is compatible with Python 3.7+ (including PyPy).

Netcat (*HOBBIT*, 1995) is a versatile Unix utility that facilitates the transfer of data over network connections using TCP or UDP protocols. It serves as a reliable tool that can be used directly or seamlessly integrated into other programs and scripts. Beyond its core functionality, Netcat offers a wide range of features, making it a valuable tool for network debugging and exploration. It can create various types of connections and includes several built-in capabilities. Despite its usefulness, it is surprising that Netcat was not provided as a standard Unix tool earlier, given its cryptic yet widely recognized nature.

2.5.1 Software-based traffic generation

Packet Generator (Pktgen), also known as Packet Generator, is a software-based tool that generates network traffic by leveraging the Data Plane Development Kit (DPDK) (PROJECTS, 2023) fast packet processing framework(WILES, 2023), Created in 2010 by Keith Wiles the Pktgen is capable of generating traffic at wire rate, achieving 10Gbit/s speeds with 64-byte frames. Acting as both a transmitter and receiver at line rate. With runtime environment, the Pktgen configuration interface, allows users to easily define and manage traffic flows.

Real-time metrics for multiple ports can be displayed, providing valuable insights into network performance. Additionally, Pktgen enables the generation of packets in sequence by iterating through source or destination Media Access Control (MAC), IP addresses, or ports. Supporting including UDP, TCP, Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), Generic Routing Encapsulation (GRE), Multi Protocol Label Switching (MPLS), and Queue-in-Queue, making it versatile for different network scenarios. The software is highly configurable via Lua, released under a BSD license.

TRex(Realistic Traffic Generator)(TREX, 2023) is a cost-effective, open-source traffic generator that operates in both stateful and stateless modes, leveraging the power of DPDK.

It is capable of generating L3-7(layer 3 to 7 in network layers) traffic. In its stateless mode, TRex supports multiple streams. It provides detailed statistics, including latency and jitter, at the stream/group level, with advanced stateful functionality that enables the emulation of Layer 7 traffic, incorporating fully-featured scalable TCP/UDP. In addition, TRex is its scalability, reaching speeds of up to 200Gbit/s with a single server, making it suitable for high-performance testing scenarios.

Built on the just-in-time (JIT) compiler LuaJIT(PALL, 2023) and the packet processing framework DPDK, MoonGen(EMMERICH et al., 2015) was idealized in 2015, that can saturate 10 Gb/s links using minimal-sized packets while efficiently utilizing a single CPU core. Leveraging the powerful packet processing framework DPDK, MoonGen achieves impressive scalability through linear multi-core scaling, enabling even higher transmission rates. Our extensive testing has confirmed MoonGen's capability to reach an astounding 178.5 million packets per second (Mpps) at a remarkable speed of 120 Gbit/s. Furthermore, MoonGen 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.

2.5.2 Hardware-based traffic generation

The HyperTester (WANG; XU; WU, 2019), (ZHOU et al., 2019) is a hybrid software-based traffic generation and hardware-based traffic replication, the implementation of HyperTester on the Tofino switch, along with numerous network testing tasks, has yielded promising results. The authors introduce the Network Testing API (NTAPI), which provides a means 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. HyperTester is a fully implemented solution that operates on a single programmable switch and consists of three layers: NTAPI, switch CPU, and switching Application-Specific Integrated Circuit (ASIC). The process begins with network operators defining various tasks using NTAPI. The switch CPU then compiles these tasks and generates template packets along with the necessary P4 program. The sender module generates test packets based on the template packets, while the switch CPU analyzes the test statistics obtained from the switching ASIC. Finally, the switching ASIC accelerates and modifies the template packets, generating the final test traffic at the desired speed on the sender module. Evaluations on the hardware testbed demonstrate that HyperTester achieves line-rate packet generation, reaching speeds of 400Gbit/s, while maintaining highly accurate rate control.

P4STA (KUNDEL et al., 2020), (KUNDEL et al., 2022), is an open-source framework that combines the flexibility of software-based traffic load generation with the accuracy of hardware packet timestamping. The evaluation results obtained using an off-the-shelf P4-programmable switch, show that a time resolution of up to 1 nanosecond can be achieved on these programmable data plane platforms. Moreover, the authors show how to combine

the traffic load of multiple software-based load generators to achieve a measurement load of up to 100Gbit/s per port. In general, is an architecture that combines multiple load generation sources and enhances them with accurate hardware timestamps before sending them to the device under test. Once the device under test responds by sending back the packets, they are appended with a second timestamp and duplicated to an external host. At the external host, the hardware timestamps are extracted and can be utilized to calculate round-trip times (RTTs) and other relevant metrics.

The FPGA-Based Synthetic Ethernet Traffic Generator, proposed by Yuan (YUAN et al., 2017). In their paper, Fast Flow-Based Ethernet Traffic Generator on Field-programmable Gate Array (FPGA) presents a cost-effective solution for network evaluation. The system utilizes FPGA technology to generate synthetic Ethernet traffic at a rate of 10 Gbit/s. The configuration data from a computer is transmitted to the FPGA(COMBO-LXT) board via a Peripheral Component Interconnect (PCI) Express bus, allowing for efficient traffic generation. The FPGA board is equipped with user-friendly interfaces for controlling the data flow and achieving a 10Gbit/s link rate. Each hardware block is linked to an interface to facilitate seamless communication and operation.

In the High-Speed FPGA-Based Ethernet Traffic Generator (PLAKALOVIC; KALJIC; MEHIC, 2022), presents a scalable Ethernet traffic generator based on FPGA is presented. The proposed solution demonstrates the capability to fully utilize a 40Gb/s link while offering the flexibility to manipulate traffic characteristics at the packet level. Although initially designed for the DE10-Pro system, the proposed architecture can be easily adapted to other FPGA boards with minimal development effort and modifications, ensuring portability and versatility. The hardware design of the packet generator was implemented using VHSIC Hardware Description Language (VHDL), adhering to predetermined characteristics and signal definitions. This approach ensured compatibility with Avalon MM and Avalon ST interfaces, facilitating seamless integration and communication within the system.

The authors of (LINDNER; HÄBERLE; MENTH, 2023) present P4TG, a traffic generator based on the P4-programmable Intel Tofino ASIC. P4TG operates in two modes: generation mode and analysis mode. In generation mode, P4TG is capable of generating up to 1 Tb/s of Ethernet traffic, distributed across 10 ports at a rate of 100 Gb/s each. It supports packet customization and provides measurements of L1 and L2 transmit and receive rates, 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 Gb/s, surpassing the performance of other traffic generators. Additionally, P4TG is capable of generating random traffic, enhancing its versatility.

In analysis mode, P4TG analyzes external traffic and provides measurements of rates, frame types and sizes, and samples IATs. It also acts as a transparent forwarder, allowing for traffic analysis without disrupting connectivity. The paper provides a detailed description of P4TG's implementation and includes a performance comparison with other traffic genera-

tors. The results highlight the superior performance of P4TG in generating stable IATs and its ability to analyze traffic effectively.

2.5.3 Outline

In this section, we provide a summary of all the presented works, sorting them into specific categories. This provides a good insight into the focus of these works, their limitations, and where our strategy fits in relation to these works. The table 2 allows us to compare the most relevant related works and PIPO-TG. Below we described the 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., Sinusoid and Flashcrowd (ALMEIDA et al., 2023)).
- **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.

Table 2 presents the performance of each work in the defined categories, as well as the performance of the proposed strategy. As we can observe, our work is the only one that propose a highly customizable traffic generator that needs a single pipeline for TNA architecture that allows to specify user-defined packet header customization with an open source code for foster reproducibility.

Table 2 – Review of related works.
- unspecified maximum link rate.

		Feature			
	Propose	**100+ Gbps**	**Workload Assay**	**Custom Traffic**	**Internal Generation**
Software-based	Pktgen(WILES, 2023)	✗	✗	✓	✓
	MoonGen(EMMERICH et al., 2015)	✗	✗	✓	✓
	TRex(TREX, 2023)	✗	✗	✓	✓
	Iperf2(MCMAHON; AUCKLAND, 2005)	-	✗	✓	✓
	NetPerf(JONES, 1996)	-	✗	✗	✓
	httpperf(MOSBERGER; JIN, 1998)	-	✗	✗	✓
	Scapy(BIONDI, 2011)	-	✗	✓	✓
	Netcat(*HOBBIT*, 1995)	-	✗	✗	✓
	Wave(ALMEIDA et al., 2023)	-	✓	✗	✗
Hardware-based	Yuan(YUAN et al., 2017)	✗	✗	✓	✓
	HyperTester(WANG; XU; WU, 2019)	✓	✗	✓	✗
	P4STA(KUNDEL et al., 2020)	✓	✗	✗	✗
	Plakalovic(PLAKALOVIC; KALJIC; MEHIC, 2022)	✗	✗	✓	✓
	P4TG(LINDNER; HÄBERLE; MENTH, 2023)	✓	✗	✗	✓
	PIPO-TG	✓	✓	✓	✓

As we can observe, similarly to P4TG (LINDNER; HÄBERLE; MENTH, 2023), Hyper-Tester (ZHOU et al., 2019) and P4STA (KUNDEL et al., 2020), 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.

3 PIPO-TG DESIGN & IMPLEMENTATION

In this chapter, we discuss the details of our PIPO-TG traffic generator. First, we introduce the details about the traffic generator design and workflow, and next, we describe more about the implementation details. Finally, we present the main challenges identified to finish the PIPO-TG implementation.

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

Furthermore, while we generate the traffic, we can simultaneously execute another P4 code (from the user) that receives this traffic and can carry out its operation normally. This enables the possibility of testing a P4 code on a single P4 switch, without the need for an external server for traffic generation. In the next sections, we discuss the traffic generation process using P4-TG, its architecture, main components, and currently identified limitations.

3.1 Architecture

The PIPO-TG architecture is presented in Figure 5, illustrating the high-level components and their interconnections. These components are divided into blocks, first, the input of the user will be processed by the processing module, which will generate the files, and finally, this goes to execute these files, and they are described below:

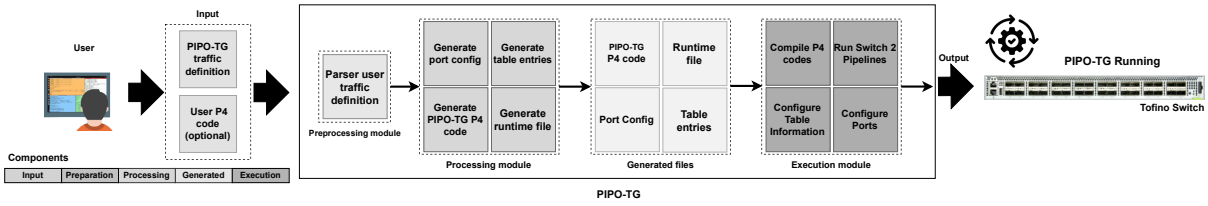


Figure 5 – PIPO-TG architecture and workflow

3.1.1 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 Internet Protocol (IP) 10.0.0.2 and a custom header to be sent via physical port 5. Additionally, the user defines configuration details such as the pipeline generation port, port bandwidth, and the type of traffic limitation desired. As mentioned, the user can also define a P4 code

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 (Mbps) and the type
#(port_shaping or meter)
myTG.addThroughput(100, "meter")
#start traffic generator
myTG.generate()
```

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.

- **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 (for example, 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 the processing module will use.

- **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.

- **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 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.

- **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.

3.2 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. The main features available in PIPO-TG are summarized below:

- **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 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 (similarly to conventional headers).

- **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 (ALMEIDA et al., 2023) 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 use the multiple pipeline support to execute a user-defined P4 code that receives the traffic generated by PIPO-TG. More details about this feature will be discussed in the next section.

Note that the features discussed are not the literal commands present in the PIPO-TG script. To see the available commands and how to use each feature, access our GitHub repository (COSTA et al., 2023).

3.3 Implementation

Next, we will discuss the PIPO-TG implementation, presenting the strategies for implementing the available features. Table 3 summarizes the strategies used to implement each feature. Additionally, we discuss each of them in more detail below.

- **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.

- **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.

- **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).

- **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.

- **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.

Table 3 – PIPO-TG implementation overview.

Feature	Implementation approach
Traffic crafting	Tofino internal traffic generation unit to create 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 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 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

- **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.

- **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.

3.4 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 the user's P4 code must pass by recirculation in the user pipe so 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.

4 EVALUATION

In this chapter, 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. These use cases seek to demonstrate the potential of PIPO-TG in reproducing the most varied events.

Setup. The experiments were conducted in a Tofino switch (Edge-Core Wedge100BF-32X) directly connected to a local server (Intel Xeon E5-2620v2, dual-port 10G Intel X540-AT2 NIC, and 64GB of memory running Ubuntu 20.04) connected via 10G SFP+ interfaces.

4.1 Resource utilization

In this section, we present the resource utilization of our PIPO-TG P4 code compiled for the Tofino 1 using the SDE version 9.9. Table 4 compares the PIPO-TG resource utilization with the resource utilization of the switch.p4, baseline of P4 code for switching. For the measured example, we consider traffic generation using a throughput definition with meters and the definition of two customizable protocols. We chose this case because these are the features that have the most influence on the P4 code due to the use of meters and the definition of new headers.

As we can see in the table, PIPO-TG uses very few switch resources, proving to be a lightweight tool for Tofino traffic generation. In addition to the resources presented in the table, PIPO-TG uses only 4 of the 12 available processing stages. This gives indications of the scalability of PIPO-TG, demonstrating that it is still possible to implement many other features, as we still have a large amount of resources available.

Table 4 – Hardware resource utilization

Resource	**Switch.p4**	**PIPO-TG**
Hash Bits	32.3%	1.7%
SRAM	29.8%	1.0%
TCAM	28.4%	0.0%
VLIW Actions	34.6%	2.3%
Stateful ALUs	15.6%	0.0%

4.2 PIPO-TG vs state-of-the-art

We compare PIPO-TG against P4TG (LINDNER; HÄBERLE; MENTH, 2023) and HyperTester (ZHOU et al., 2019) 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 5, we compare PIPO-TG, P4TG, and HyperTester qualitatively regarding their characteristics. Below, we describe key characteristics and how each generator fits into them.

- **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 Protocol (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.

4.3 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 (BALADOR et al., 2022), such as data centers, by alternating traffic workload to

Table 5 – Qualitative evaluation of PIPO-TG vs state-of-the-art

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**

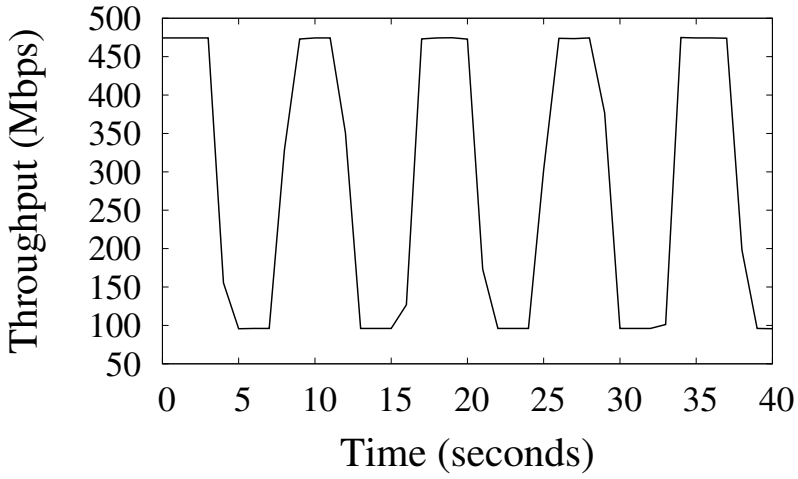
Algorithm 2 PIPO-TG traffic alternation code snippet.

```
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")
#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")
#start traffic generation
myTG.generate()
```

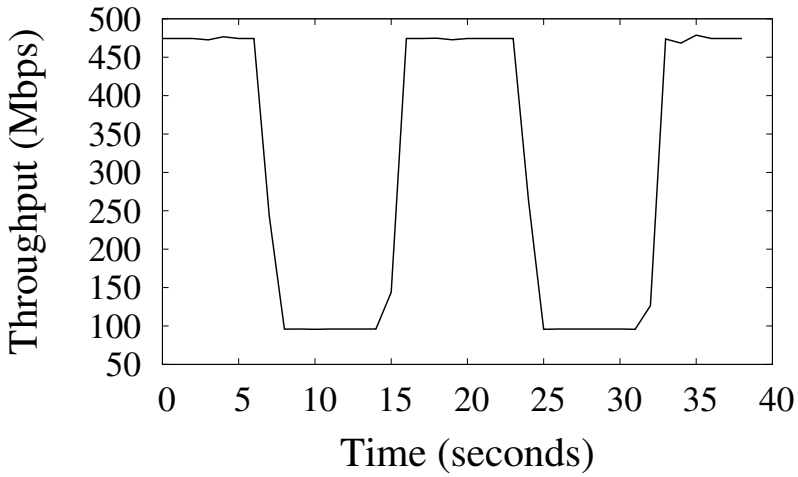
assess how well the network manages congestion and prioritizes traffic. Similarly, it may be interesting to stress-test (SOÓS; JANKY; VARGA, 2019) a network testbed/device to understand how it performs under different loads. In this case, a traffic alternation pattern can be valuable. Figure 6 presents 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 Figure 6(a), we alternate the throughput every 4 seconds, while at the Figure 6(b), 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. It means there is 98.58% (or 70X) less code when compared to the generated files.

4.4 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 (ARI et al., 2003), TCP Incast scenarios (ALIZADEH et al., 2010), TCP segment offloading or application-level batch processing (KAPOOR et al., 2013). 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.



((a)) 4-second square wave shape.



((b)) 8-second square wave shape.

Figure 6 – Simulated traffic alternation.

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 PIPO-TG codes (P4 and table 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 7 shows the result of the generated bursts, with bursts of approximately 2 seconds arriving every 8 seconds on the server

Algorithm 3 PIPO-TG burst traffic code snippet.

```
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")
#([Throughputs], [Intervals])
myTG.addVariance([10000, 90000], [8, 2])
#start traffic generator
myTG.generate()
```

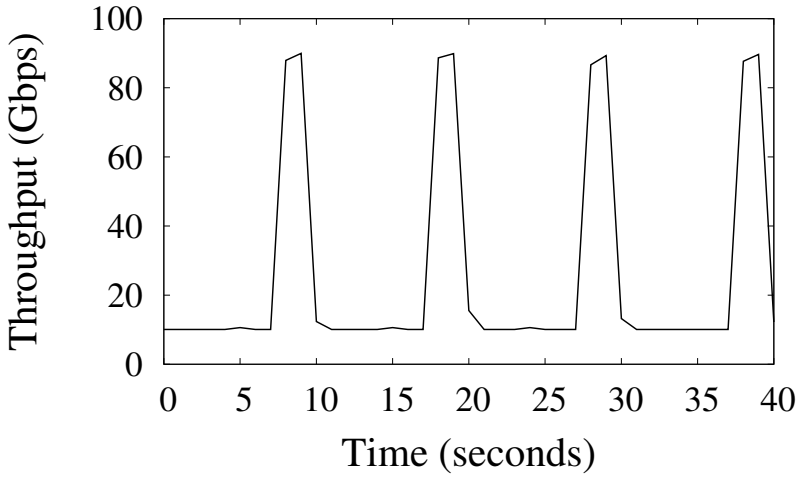


Figure 7 – Generation of network bursts every 8 seconds.

4.5 Use case III: DDoS simulation

The third use case involves simulating a DDoS attack scenario (VANITHA; UMA; MAHIDHAR, 2017). Strategies for modeling DDoS attacks (VANITHA; UMA; MAHIDHAR, 2017; CETINKAYA; ISHII; HAYAKAWA, 2019) 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 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 8 displays the source IP distribution in the traffic. We captured the first 10K packets

Algorithm 4 PIPO-TG DDoS simulation code snippet.

```
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")
#define troughtput(Mbps) and the type(port_shaping or meter)
myTG.addThroughput(10000, "meter")
myTG.addIP(src="192.168.1.0",srcRandom = True, srcMask = 24,dst= "192.168.2.2")
#start traffic generator
myTG.generate()
```

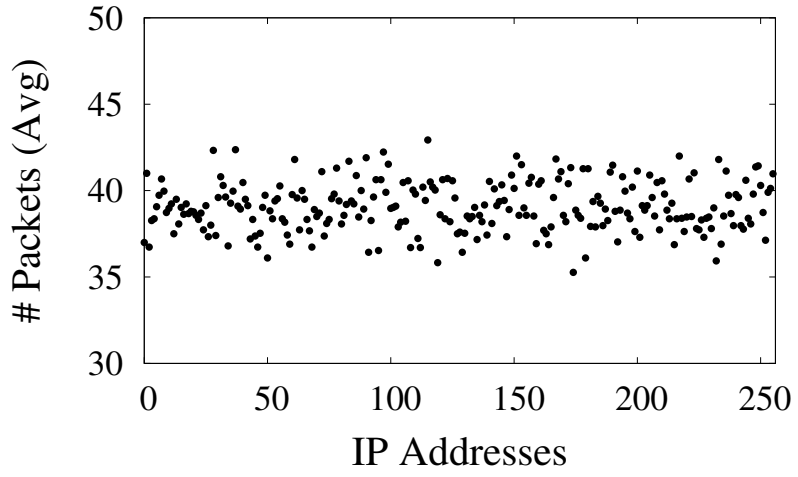


Figure 8 – Average of packets per IP address.

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 simulating this network scenario.

4.6 Discussion

PIPO-TG stands out in supporting custom protocols, allowing users to work with customizable headers. This feature is crucial for adapting to diverse networking scenarios where predefined protocols may not suffice. P4TG is limited to generating Ethernet/IP packets, limiting its flexibility in this aspect. The Number of flows, PIPO-TG excels in creating a higher number of distinct flows compared to P4TG, which is restricted to 7 different flows due to Tofino traffic generation limitations. This flexibility is essential for simulating complex network scenarios with diverse flow patterns. In Tofino internal traffic generation, both PIPO-TG and P4TG leverage the Tofino internal generation unit, eliminating the need for an external CPU or server. HyperTester, on the other hand, relies on the CPU for packet generation.

This distinction impacts the efficiency of traffic generation in terms of resource utilization.

- Workload Alternation: The demonstrated ability of PIPO-TG to generate workload alternation patterns proves valuable in scenarios such as congestion control monitoring and network stress testing. The concise code implementation, as illustrated in Algorithm 2, emphasizes PIPO-TG's efficiency in generating complex traffic patterns with minimal coding effort.

- Burst Simulation: PIPO-TG's capability to simulate traffic bursts, as showcased in Algorithm 3, addresses real-world scenarios where bursty traffic can lead to network challenges. The significant reduction in code effort (98.8%) highlights PIPO-TG's practicality in defining and implementing burst simulations with ease.

- DDoS Simulation: The DDoS simulation use case, outlined in Algorithm 4, demonstrates PIPO-TG's effectiveness in modeling sophisticated network attack scenarios. The substantial reduction in code effort (98.5%) underscores PIPO-TG's efficiency in simulating complex distributed attacks, offering a streamlined approach for researchers and practitioners.

In conclusion, PIPO-TG emerges as a versatile and efficient traffic generator, surpassing its counterparts in several key aspects. Its support for custom protocols, ability to generate diverse traffic patterns, and open-source nature contribute to its appeal in the networking community. The presented use cases showcase PIPO-TG's practical utility in simulating a wide range of scenarios with minimal coding effort, making it a valuable tool for network researchers and practitioners.

5 FINAL REMARKS

In this chapter, we present a comprehensive overview of the key aspects addressed in this study. We begin by summarizing the main points covered throughout this work, highlighting the key findings and contributions. Additionally, we review the results obtained from our research until here, efforts and discuss their implications in the context of the research objectives. Furthermore, to conclude this study, we outline potential avenues for final work, including areas that warrant further investigation, aspects that were not addressed within the scope of this research, and proposed solutions to address any limitations identified. By examining these aspects, we aim to provide a holistic perspective on the contributions of this work and lay the foundation for future advancements in the field.

5.1 Conclusion

Traffic generation in computer networks is crucial for assessing and optimizing network performance. By simulating various scenarios and traffic patterns, it enables researchers and network engineers to identify potential bottlenecks, evaluate the efficiency of protocols, and enhance overall system reliability. Additionally, realistic traffic generation is fundamental for testing and validating the scalability and resilience of networks, ensuring they can handle diverse loads and remain robust in real-world usage. 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.

5.2 Future Work

In future works, we plan to develop a platform for monitoring the generated traffic. This expansion will allow real-time analysis and detailed configuration of the traffic being generated. Besides that, we want to incorporate the support for stateful connections, encompassing widely used protocols like TCP, QUIC, and others. This is currently supported in a basic way by HyperTester with TCP traffic. However, we plan to resolve all limitations and also implement other stateful protocols. Additionally, we aim to enrich PIPO-TG's capabilities by enabling the replaying of packet captures, with a specific emphasis on accommodating .pcap files as input.

In a strategic move towards comprehensive improvement, we have outlined specific plans to evaluate the scalability of PIPO-TG. This evaluation will delve into the impact of numerous custom headers on its performance, for example, allowing us to meticulously study and analyze the results. By doing so, we aim to refine and optimize PIPO-TG to ensure its robustness and efficiency under varying conditions.

In conclusion, our roadmap for future work on PIPO-TG has many paths. From expanding functionalities to evaluating scalability and integrating with P7(RODRIGUEZ et al., 2022), each initiative is carefully designed to contribute to the ongoing evolution and efficacy of PIPO-TG. As we undertake these efforts, our focus is on delivering high-quality solutions that empower users to effectively address their networking challenges working in the SDN environment.

BIBLIOGRAPHY

ADELEKE, O. A.; BASTIN, N.; GURKAN, D. Network traffic generation: A survey and methodology. **ACM Computing Surveys (CSUR)**, ACM New York, NY, v. 55, n. 2, p. 1–23, 2022. Cited in page 35.

ALIZADEH, M. et al. Data center tcp (dctcp). In: **Proceedings of the ACM SIGCOMM 2010 Conference**. [S.l.: s.n.], 2010. p. 63–74. Cited in page 51.

ALMEIDA, L. C. de et al. Wave-um gerador de cargas múltiplas para experimentação em redes de computadores. In: SBC. **Anais Estendidos do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2023. p. 9–16. Cited 4 times in the pages 34, 39, 40, and 46.

ARI, I. et al. Managing flash crowds on the internet. In: IEEE. **11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.** [S.l.], 2003. p. 246–249. Cited in page 51.

BALADOR, A. et al. Survey on decentralized congestion control methods for vehicular communication. **Vehicular Communications**, Elsevier, v. 33, p. 100394, 2022. Cited in page 50.

BIONDI, P. **Scapy**. 2011. [Acces: June 26, 2023]. Disponível em: <https://scapy.net/>. Cited 2 times in the pages 36 and 40.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **ACM SIGCOMM Computer Communication Review**, ACM New York, NY, USA, v. 44, n. 3, p. 87–95, 2014. Cited 2 times in the pages 28 and 30.

BOTTA, A.; DAINOTTI, A.; PESCAPÉ, A. Do you trust your software-based traffic generator? **IEEE Communications Magazine**, IEEE, v. 48, n. 9, p. 158–165, 2010. Cited in page 27.

CETINKAYA, A.; ISHII, H.; HAYAKAWA, T. An overview on denial-of-service attacks in control systems: Attack models and security analyses. **Entropy**, MDPI, v. 21, n. 2, p. 210, 2019. Cited in page 53.

COSTA, F. et al. **PIPO-TG: Parametrizable High Performance Traffic Generatior**. [S.l.]: GitHub, 2023. <https://github.com/FilipoGC/PIPO-TG>. Cited in page 46.

EMMERICH, P. et al. Mind the gap-a comparison of software packet generators. In: IEEE. **2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)**. [S.l.], 2017. p. 191–203. Cited in page 28.

EMMERICH, P. et al. Moongen: A scriptable high-speed packet generator. In: **Proceedings of the 2015 Internet Measurement Conference**. [S.l.: s.n.], 2015. p. 275–287. Cited 4 times in the pages 21, 35, 37, and 40.

FOUNDATION, O. N. **P4 Language Specification version 1.2.0**. 2019. Accessed: May 07, 2023. Disponível em: <https://p4.org/p4-spec/docs/P4-16-v1.2.0-spec.html>. Cited 2 times in the pages 30 and 31.

FOUNDATION, O. N. **2021 P4 Workshop**. 2021. [Acces: June 26, 2023]. Disponível em: <https://opennetworking.org/2021-p4-workshop-content/>. Cited in page 34.

HOBBIT. **Netcat**. 1995. [Acces: June 26, 2023]. Disponível em: <https://nc110.sourceforge.io/>. Cited 2 times in the pages 36 and 40.

INTEL. **Intel Tofino**. 2021. [Access: May 07, 2023]. Disponível em: <https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html>. Cited in page 21.

INTEL. **Intel Tofino Native Architecture—Public Version**. 2021. Access: May 07, 2023]. Disponível em: <https://github.com/barefootnetworks/Open-Tofino>. Cited in page 32.

INTEL. **P416 Intel Tofino Native Architecture—Public Version.** 2021. <https://github.com/barefootnetworks/Open-Tofino">. Cited in page 22.

(ITU), I. T. U. **Individuals using the Internet**. 2022. [Acces: June 29, 2023]. Disponível em: <https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx>. Cited in page 21.

IXIA traffic generator. Available on: <"http://www.ixiacom.com">. Cited in page 22.

JONES, R. **Netperf. Hewlett-Packard.** 1996. [Acces: June 26, 2023]. Disponível em: <https://github.com/HewlettPackard/netperf>. Cited 2 times in the pages 35 and 40.

KAPOOR, R. et al. Bullet trains: A study of nic burst behavior at microsecond timescales. In: **Proceedings of the ninth ACM conference on Emerging networking experiments and technologies**. [S.l.: s.n.], 2013. p. 133–138. Cited in page 51.

KHATIB, E. J. et al. Designing a 6g testbed for location: Use cases, challenges, enablers and requirements. **IEEE Access**, IEEE, v. 11, p. 10053–10091, 2023. Cited in page 21.

KUNDEL, R. et al. P4sta: High performance packet timestamping with programmable packet processors. In: IEEE. **NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium**. [S.l.], 2020. p. 1–9. Cited 3 times in the pages 37, 40, and 41.

KUNDEL, R. et al. Network testing utilizing programmable network hardware. **IEEE Communications Magazine**, IEEE, v. 60, n. 2, p. 12–17, 2022. Cited in page 37.

LAI, Y.-K. et al. Sketch-based entropy estimation for network traffic analysis using programmable data plane asics. In: IEEE. **2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)**. [S.l.], 2019. p. 1–2. Cited in page 33.

LINDNER, S.; HÄBERLE, M.; MENTH, M. P4tg: 1 tb/s traffic generation for ethernet/ip networks. **IEEE Access**, IEEE, v. 11, p. 17525–17535, 2023. Cited 6 times in the pages 22, 33, 38, 40, 41, and 49.

LIU, W.-x. et al. Programmable data plane intelligence: advances, opportunities, and challenges. **IEEE Network**, IEEE, 2022. Cited in page 21.

MCMAHON, B. K. R.; AUCKLAND, T. **Iperf: The TCP/UDP bandwidth measurement tool**. 2005. [Acces: June 19, 2023]. Disponível em: <https://sourceforge.net/projects/iperf2/>. Cited 2 times in the pages 35 and 40.

MOLNÁR, S.; MEGYESI, P.; SZABÓ, G. How to validate traffic generators? In: IEEE. **2013 IEEE International Conference on Communications Workshops (ICC)**. [S.l.], 2013. p. 1340–1344. Cited 2 times in the pages 25 and 35.

MOSBERGER, D.; JIN, T. Httperf—a tool for measuring web server performance. **SIGMETRICS Perform. Eval. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 26, n. 3, p. 31–37, dec 1998. ISSN 0163-5999. Disponível em: <https://doi.org/10.1145/306225.306235>. Cited 2 times in the pages 35 and 40.

PALL, M. **LuaJIT**. 2023. [Acces: June 20, 2023]. Disponível em: <http://luajit.org/>. Cited in page 37.

PLAKALOVIC, M.; KALJIC, E.; MEHIC, M. High-speed fpga-based ethernet traffic generator. In: IEEE. **2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT)**. [S.l.], 2022. p. 1–6. Cited 3 times in the pages 22, 38, and 40.

PROJECTS, L. **Data Plane Development Kit**. 2023. [Acces: May 19, 2023]. Disponível em: <https://www.dpdk.org/>. Cited in page 36.

RODRIGUEZ, F. et al. P4 programmable patch panel (p7) an instant 100g emulated network on your tofino-based pizza box. In: **Proceedings of the SIGCOMM'22 Poster and Demo Sessions**. [S.l.: s.n.], 2022. p. 4–6. Cited 2 times in the pages 33 and 58.

SONG, H. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In: **Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking**. [S.l.: s.n.], 2013. p. 127–132. Cited in page 28.

SOÓS, G.; JANKY, F. N.; VARGA, P. Distinguishing 5g iot use-cases through analyzing signaling traffic characteristics. In: IEEE. **2019 42nd International Conference on Telecommunications and Signal Processing (TSP)**. [S.l.], 2019. p. 562–565. Cited in page 51.

TREX, T. **TRex Realistic Traffic Generator**. 2023. [Acces: May 20, 2023]. Disponível em: <https://trex-tgn.cisco.com/>. Cited 3 times in the pages 21, 36, and 40.

VANITHA, K.; UMA, S. V.; MAHIDHAR, S. Distributed denial of service: Attack techniques and mitigation. In: **2017 International Conference on Circuits, Controls, and Communications (CCUBE)**. [S.l.: s.n.], 2017. p. 226–231. Cited in page 53.

WANG, J.; XU, M.; WU, J. Hypertester: High-performance network testing driven by programmable switches. 2019. Cited 2 times in the pages 37 and 40.

WILES, K. **Pktgen-DPDK Documentation, Release 22.07.2**. 2023. [Acces: May 19, 2023]. Disponível em: <https://buildmedia.readthedocs.org/media/pdf/pktgen-dpdk/latest/pktgen-dpdk.pdf>. Cited 4 times in the pages 21, 35, 36, and 40.

YUAN, D. et al. A fast, affordable and extensible fpga-based synthetic ethernet traffic generator for network evaluation. In: IEEE. **2017 3rd IEEE International Conference on Computer and Communications (ICCC)**. [S.l.], 2017. p. 1036–1040. Cited 2 times in the pages 38 and 40.

ZHOU, Y. et al. Hypertester: high-performance network testing driven by programmable switches. In: **Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies**. [S.l.: s.n.], 2019. p. 30–43. Cited 4 times in the pages 22, 37, 41, and 49.