© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, including reprinting/republishing this material for advertising or promotional purposes, collecting new collected works for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Digital Object Identifier 10.1109/NetSoft70012.2026.11603500

High-Speed Generation of Periodic Traffic Patterns on P4TG for DDoS and Burst-Load Evaluation

Fabian Ihle, Etienne Zink, and Michael Menth
University of Tübingen, Chair of Communication Networks
Email: {fabian.ihle, etienne.zink, menth}@uni-tuebingen.de

***Abstract*—Traffic generators are essential tools for evaluating the robustness and performance of networked systems. P4TG is an open-source, hardware-accelerated traffic generator implemented in P4 for the Intel Tofino™ ASIC. It has been adopted by researchers and industry due to its flexibility and multi-terabit generation capability, and its low cost compared to other traffic generators. However, like most existing generators, it primarily produces constant bit rate traffic, which does not reflect the highly time-varying behavior observed in real networks, such as flashcrowds and microbursts. Such patterns are difficult to emulate at scale with current tools. We present a data plane mechanism for P4TG that shapes periodic, time-varying traffic patterns, including patterns representative of DDoS attacks and burst-load scenarios. Pattern shaping in P4TG can be applied to its generated traffic at an aggregate throughput of up to 4 Tbit/s. We evaluate pattern accuracy and analyze scalability across different sampling resolutions and periods. Further, we demonstrate practical use cases, including zero-loss throughput determination and buffer capacity measurement. Finally, we present microburst-based attack scenarios that overload UDP receivers, switch buffers, and degrade TCP throughput on shared links while remaining undetectable to conventional rate monitoring.**

***Index Terms*—Data Plane Programming, Network Testing, Microbursts, P4, Traffic Generator**

I. Introduction

Traffic generators are essential for evaluating the performance and resilience of networked systems. They enable controlled experiments by producing and analyzing packet streams that stress devices and protocols under well-defined conditions. Software-based generators are flexible and inexpensive, but cannot sustain the line rate performance required for modern high-speed networks. Hardware generators offer much higher throughput, yet at the cost of a lack of programmability and significantly higher expenses. This gap motivates the development of programmable, hardware-accelerated generators that combine flexibility with terabit-scale performance.

P4TG [1] is a recent example of such a system. Implemented in P4 on Intel Tofino™ ASICs, it provides high configurability and supports an aggregate traffic generation capacity of up to $4\,\mathrm{Tbit/s}$. While P4TG currently generates constant bit rate (CBR) and Poisson-distributed traffic, which are well suited for evaluating sustained load conditions, real network traffic often deviates from such rates. In many scenarios, traffic shows highly variable and time-dependent patterns. Flashcrowds, for example, occur when large numbers of legitimate users simultaneously access a service, creating rapid spike, plateau, and decay phases [2]–[4]. Malicious workloads such as distributed denial-of-service (DDoS) attacks similarly show characteristic patterns, including pulsing or on/off behavior [5]–[9]. Short-lived microbursts, i.e., high-rate spikes lasting only microseconds, have been widely observed in operational networks and are known to cause packet loss while often remaining invisible to coarse-grained measurements [9]–[11]. For controlled evaluation, these patterns can be approximated as periodic functions with configurable amplitude and frequency. Accurately reproducing them is crucial for evaluating how networks react to overload conditions, yet existing traffic generators do not support patterns at multi-terabit rates [12]–[16].

The contribution of this work is manifold. We introduce a programmable traffic pattern abstraction for P4TG that enables shaping of periodic traffic patterns at line rate. Our design performs pattern shaping entirely in the data plane while the control plane configures the pattern. This enables P4TG to reproduce a wide range of time-varying patterns, defined through mathematical functions at an aggregate throughput of up to $4\,\mathrm{Tbit/s}$. We provide exemplary pattern functions, including sine, square, sawtooth, and flashcrowd. We analyze the scalability of the mechanism in terms of maximum period and table footprint and show that the generated traffic accurately follows the configured patterns. We show its applicability with practical use cases that determine the zero-loss throughput, and measure the buffer capacity of a device. Further, we demonstrate three microburst-based attack scenarios which overload a UDP target, a switch buffer, and a shared TCP link while remaining nearly invisible to conventional monitoring systems. These capabilities allow controlled, high-speed evaluation of networked systems under realistic load conditions that are difficult to reproduce with existing traffic generators.

II. Technical Background

In this section, we provide a brief introduction to the P4 programming language and the Intel Tofino™ ASIC, followed by an overview of the traffic generator P4TG.

A. The P4 Programming Language and the Intel Tofino

P4 [17] is a domain-specific programming language for the data plane of network devices. It defines how packets are parsed

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant 503231190, and the use of Claude Sonnet 4.5 to assist in developing scripts for experiment automation.

and processed in a pipelined structure. A P4 program consists of a programmable parser, a sequence of control blocks, and a programmable deparser. Each control block contains match-action tables (MATs) that select actions based on packet header fields or metadata. Packets are matched against these tables using pre-defined key fields, and the associated action is then executed. MATs support several match types, including exact, ternary, and range matches.

The Intel Tofino™ 1 and 2 switching ASICs [18] are hardware platforms that execute P4 programs at line rates of up to $400\,\mathrm{Gbit/s}$ per port. They follow a pipelined architecture in which each stage can perform table lookups and simple arithmetic operations. Extern objects extend the functionality of P4. Meter externs implement a token-bucket algorithm and are used for rate control and shaping. Register externs provide stateful memory that allows P4 programs to maintain state across packets. In addition, the Intel Tofino™ provides a configurable packet generation extern that can inject packets with up to $400\,\mathrm{Gbit/s}$ into the pipeline from an internal generator.

B. The Traffic Generator P4TG

P4TG [1] is an open-source [19] traffic generator implemented in the P4 programming language for the Intel Tofino™ 1 and 2 switching ASICs. It leverages the ASIC's internal traffic generation port to produce high-speed packet streams that can be internally multicasted to up to ten physical ports. This enables an aggregate generation capacity of up to $4\,\mathrm{Tbit/s}$ ($10 \times 400\,\mathrm{Gbit/s}$) on the Intel Tofino™ 2. P4TG supports configurable traffic based on Ethernet, IPv4, and IPv6, and can optionally apply a variety of encapsulations, including VLAN, QinQ, MPLS, VxLAN, and SRv6 [20]. The frame size, header fields, and traffic characteristics, such as the generation rate, can be freely configured, allowing users to generate CBR as well as Poisson-distributed traffic. Further, P4TG supports IPv4/6 address randomization enabling millions of different IP flows, e.g., for emulating sources of DDoS attacks. Generated packets carry a lightweight UDP header that embeds P4TG-specific metadata, such as sequence numbers and timestamps, to facilitate detailed performance analysis. The data plane collects extensive traffic statistics, including round-trip times (RTTs) and inter-arrival times (optionally via histogram-based measurement [21]), TX/RX rates, packet loss, and detailed frame type and size distributions. These statistics are exported to the control plane through a REST API and visualized in real time using a web-based frontend. P4TG has been adopted by various research groups and industrial stakeholders, including Airbus [22] and the German Federal Government [23], to evaluate network prototypes and operational environments.

While P4TG provides comprehensive generation and measurement capabilities, it does not perform traffic pattern shaping beyond CBR or Poisson-distributed traffic. Therefore, in this work, we introduce periodic pattern shaping for generated traffic in P4TG.

III. Related Work

In this section, we first review related work on traffic patterns that characterize DDoS attacks and flashcrowds. Next, we summarize the generation of periodic traffic patterns in other traffic generators.

A. Traffic Patterns and DDoS Characteristics

We first summarize traffic patterns, and then characterize DDoS traffic.

1) Traffic Patterns: Traffic on the Internet is rarely CBR-shaped. Instead, it shows periodicity and sudden spikes in volume. We consider three types of traffic patterns: square waves, microbursts, and flashcrowds.

Square Waves: A number of works demonstrate that adversaries deliberately employ periodic or on/off attack patterns to evade or manipulate mitigation systems. These pulsing on/off attack strategies can be abstracted as square-wave traffic patterns, characterized by alternating phases of high and low offered load. Bremler-Barr *et al.* [5] and Asudi *et al.* [6] describe the Yo-Yo attack in which an attacker injects periodic overload bursts to force cloud auto-scaling systems to oscillate between scale-up and scale-down phases. Li *et al.* [7] analyze DNS-based pulsing attacks in which low-rate queries lead to amplified responses, producing high-volume bursts. A recent study by Kopp *et al.* [24] observed pulse-wave DDoS attack patterns at an Internet exchange point. They state that pulsed DDoS attacks constitute $27\,\%$ of DDoS attacks.

Microbursts: Microbursts are short-lived spikes in traffic that typically last only tens to hundreds of microseconds [10], yet they can cause sharp increases in latency and momentary packet loss [9]. Microbursts can be viewed as square waves with a very short high phase. Microbursts may originate from queuing inside network devices [11] or may be triggered as part of DDoS attacks. Traditional monitoring approaches, such as SNMP [25] and NetFlow [26], sample traffic at second-scale intervals, and therefore cannot detect such short-lived spikes. With coarse sampling, the instantaneous burst is averaged out, the measured rate remains low, and only the resulting packet loss reveals that congestion occurred. Consequently, several works [9]–[11], [27], [28] investigate the observability of microbursts and devise methods to detect them at microsecond- to millisecond-level resolution. However, existing studies typically evaluate microburst behavior at modest rates of up to $10\,\mathrm{Gbit/s}$ [9], [11] or rely on observing naturally occurring bursts rather than generating them in a controlled manner. In contrast, this work enables the generation of microburst patterns at aggregate rates of up to $4\,\mathrm{Tbit/s}$, providing a controlled environment to study network behavior under short-lived overload events.

Flashcrowds: The flashcrowd pattern represents a class of highly non-stationary traffic originating from legitimate users. These events occur when a large number of users begin accessing the same service simultaneously, leading to a rapid increase in demand, e.g., during major news events, software releases, or live-streamed events. Behal *et al.* [2] show that such

events can overload bandwidth, CPU, and memory resources in a manner similar to DDoS attacks. Ari *et al.* [3] characterize flashcrowd patterns with a ramp-up phase with linear growth, a sustained high-demand plateau, and a ramp-down phase with logarithmic decay. Flashcrowds are hard to distinguish from DDoS attacks which is why Silva *et al.* [4] propose models to distinguish them.

2) DDoS Traffic Characteristics: Several measurement studies quantify the temporal and volumetric characteristics of DDoS attacks. Mao *et al.* [29] show that most DDoS attacks last less than one hour, occasionally up to 12 hours, and can reach rates near one million packets per second (pps). Although such rates are manageable for ISP backbones, they overwhelm servers and security appliances. Yuan *et al.* [8] and Kopp *et al.* [24] report periodic DDoS flooding with pulse durations of 300 s and 1 minute, respectively, forming square-wave–like on/off patterns. Recent industry reports highlight a continued rise in hyper-volumetric attacks [30]–[32], i.e., attacks that exceed a rate of 1 Tbit/s. Cloudflare [30], [31] noted a substantial increase in attacks exceeding 1 Tbit/s with a record peak of 29.7 Tbit/s in Q3 2025 and surpassing the previous quarter's peak of 7.3 Tbit/s. Microsoft similarly reported a 15.72 Tbit/s attack targeting an Azure endpoint [32] in October 2025. These events typically last between tens of seconds and several minutes [30], [31].

B. Traffic Patterns in other Traffic Generators

In the following, we discuss multiple software- and hardware-based traffic generators. They are summarized in Table I.

TABLE I: Overview of traffic generators and their pattern shaping capabilities.

Traffic generator	Type	Max. generation rate	Pattern shaping
WAVE [12]	Software	n/a	Function-based
MoonGen [13]	Software	$\leq$ 120 Gbit/s	Manual
TRex [14]	Software	$\leq$ 200 Gbit/s	Not supported
P4STA [15]	Hybrid	None	Not supported
PIPO-TG [16]	Hardware	$\leq$ 1 Tbit/s	Manual
P4TG	Hardware	$\leq$ 4 Tbit/s	Function-based

The WAVE generator by Almeida *et al.* [12] orchestrates the number of concurrent application instances, e.g., `iperf`, instead of directly shaping the traffic. The number of instances over time follows predefined functions such as sinusoid and flashcrowd, from which the traffic pattern emerges indirectly.

MoonGen [13] is a scriptable DPDK-based software generator capable of packet generation up to 120 Gbit/s. It generates traffic patterns by controlling inter-packet gaps in software, but relies on NIC-specific features and the injection of invalid packets to shape timing, which fundamentally constrains the precision of its generated load.

TRex [14] is a DPDK-based traffic generator that supports CBR traffic at high speeds of up to 200 Gbit/s. However, its rate control is limited to static per-stream configurations and it cannot generate time-varying patterns such as sine waves or flashcrowd profiles.

P4STA [15] focuses on high-precision timestamping and load aggregation using P4-programmable hardware. It integrates existing software generators such as iPerf3 and MoonGen while its P4-based "Stamper" can shape the outgoing rate. However, P4STA itself does not generate traffic. It relies entirely on external software load generators, and its shaping capabilities are limited to simple rate limiting rather than programmable, time-varying patterns.

PIPO-TG [16], implemented on an Intel Tofino™ 1 ASIC, can generate traffic up to 1 Tbit/s and supports customizable headers, packet sizes, and protocol fields. PIPO-TG generates periodic traffic patterns by manually configuring a sequence of rate levels that the generator applies over time. This approach is suitable for simple step-wise patterns. However, since all rate levels must be specified explicitly, generating smooth patterns, such as sine waves and exponential decays as seen in flashcrowds, requires the manual definition of numerous intermediate rate steps. This approach becomes infeasible for approximating continuous functions with sufficient fidelity.

Overall, existing traffic generators either achieve high throughput with limited pattern flexibility (TRex, PIPO-TG) or support flexible patterns at limited rates (MoonGen). In contrast, our extension to P4TG introduces a programmable abstraction where users specify mathematical pattern functions. Those are automatically sampled, normalized, and translated into data plane rate profiles without manual rate-step configuration. This enables fine-grained generation, such as sinusoidal, square wave, sawtooth, flashcrowd, and microburst patterns at aggregate rates up to 4 Tbit/s.

IV. Pattern Shaping Mechanism in P4TG

In this section, we describe the implementation of the pattern shaping mechanism. Pattern shaping in P4TG is implemented through two coordinated components in the control and data plane. They are illustrated in Figure 1.

First, the control plane samples the pattern, normalizes it, and installs the resulting parameters into the data plane prior to generation. Next, the data plane determines the position of each generated packet within a period, and enforces the desired traffic pattern by dropping packets that exceed the target rate. In the following, we describe both mechanisms in more detail.

A. Sampling and Normalization of Periodic Functions

First, the user configures the desired pattern via the REST API or web-based frontend with the pattern function f_{pattern}, the period length T, the sampling factor k, and the maximum packet rate R_{max} in packets per second (pps), i.e., the amplitude. Then, before traffic generation starts, the control plane derives

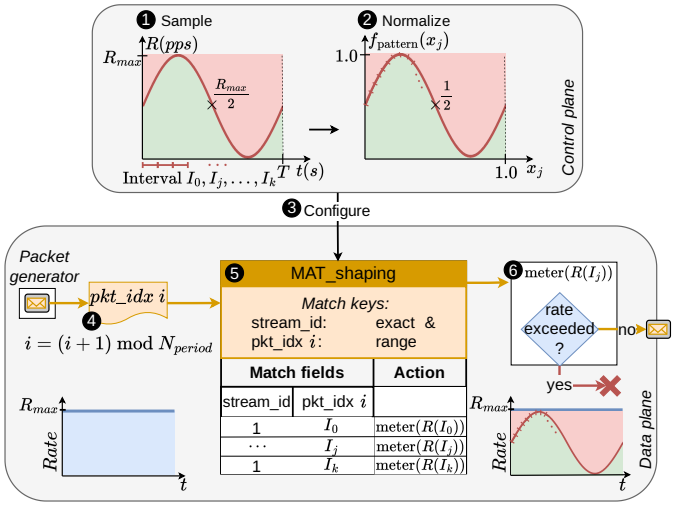


Fig. 1: The control and data plane mechanisms for pattern shaping in P4TG.

all parameters required for periodic pattern shaping which is illustrated in step ❶ to ❸ in Figure 1. These parameters include the number of packets per period, the sampling intervals based on the sampling factor k, and the rate for each sampled interval. This section covers the operations the control plane performs to sample and normalize the configured patterns.

1) Determining the Number of Packets per Period: Given the period T and the rate R_{max}, the control plane first computes how many packets N_{period} are generated during one full period:

$$N_{\text{period}} = R_{\text{max}} \cdot T. \tag{1}$$

This value is given to the data plane.

2) Sampling the Period into k Intervals: To approximate the pattern, the period is divided into k equally sized sampling intervals. This is shown in step ❶ in Figure 1. Each interval contains

$$N_{\text{interval}} = \frac{N_{\text{period}}}{k} \tag{2}$$

packets. This yields the interval boundaries

$$I_j = [j \cdot N_{\text{interval}},\ (j+1) \cdot N_{\text{interval}}), \qquad j = 0, 1, \ldots k-1. \tag{3}$$

These intervals define the entries of the shaping MAT used in the data plane. A sampling factor of $k = 32$ provides a good tradeoff between the accuracy of the sampled pattern and required table space. This is further evaluated in Section V-A2.

3) Mapping Intervals onto the Pattern Function: Each interval index j is normalized to a value $x_j \in [0, 1)$, as illustrated in step ❷ of Figure 1:

$$x_j = \frac{j}{k}. \tag{4}$$

The normalized value is then passed to the configured pattern function

$$f_{\text{pattern}} : [0, 1) \times P \to [0, 1], \tag{5}$$

which defines the desired pattern, e.g., sine, square, sawtooth, or flashcrowd. Here, P denotes the set of pattern-specific parameters, allowing functions to be parameterized, e.g., by an exponential decay rate in the flashcrowd pattern. Examples for f_{pattern} are given in Section IV-C.

4) Computing the Target Rate for Each Interval: Finally, the output amplitude of the pattern during interval I_j determines the applied shaping rate:

$$R(I_j) = f_{\text{pattern}}(x_j) \cdot R_{\text{max}}. \tag{6}$$

Afterwards, these rates are written into the shaping MAT and configure meter externs in the data plane, shown in step ❸ in Figure 1. Traffic exceeding these rates is dropped in the data plane.

B. Data Plane Enforcement of Traffic Patterns

An overview of the data plane logic is shown in step ❹ to ❻ in Figure 1. For periodic pattern shaping, traffic is generated per stream as CBR traffic with a configurable rate R_{max}. The P4 pipeline then enforces the configured pattern by selectively dropping packets whose rate exceeds the sampled rate $R(I_j)$ for each interval (computed in Section IV-A). The data plane mechanism consists of two building blocks: a packet-based mechanism for determining the position of a packet within the period, and a token-bucket algorithm for shaping.

1) Packet-based Periodicity: Packets need to be mapped to a position in the periodic pattern. To determine a packet's position within the period, P4TG maintains a per-stream packet counter in a 32 bit register. This counter increments with each packet and wraps around after a configured number N_{period} producing a periodic packet index

$$i \in [0,\ N_{\text{period}}), \tag{7}$$

shown in step ❹ in Figure 1. This provides a lightweight approach to periodicity, though it can also produce non-periodic patterns by limiting execution to a single period. In principle, periodicity can also be derived from timestamps [33], which supports arbitrarily sized intervals but requires significant pipeline resources. Since P4TG discretizes patterns into k equally sized intervals, a wrap-around packet counter is sufficient and more resource-efficient.

2) Pattern Shaping: Once the position of a packet within a period is determined, the packet must be metered according to the pattern. For this purpose, the pipeline contains a MAT that stores k equally-sized intervals of one period, i.e., the intervals I_j calculated in Section IV-A. Based on the current packet's position within the period, it matches one of these intervals in the MAT, shown in step ❺. Each interval is associated with a meter extern, implementing a token bucket algorithm as defined in RFC 2698 [34]. Here, tokens accumulate at a configured rate, and each packet consumes tokens proportionally to its size. Packets for which no tokens remain are dropped. The token generation rate for each interval directly corresponds to the target rate $R(I_j)$ computed by the control plane. Packets exceeding $R(I_j)$ are therefore dropped, shown in step ❻.

With this approach, shaping is executed entirely in the data plane without requiring dynamic reconfiguration of the traffic generator, enabling pattern shaping at line rate.

3) Range-to-Ternary Conversion: In the data plane, packets are matched to the sampled intervals of the shaping MAT based on the $32\,\mathrm{bit}$ packet index within the current period. Although P4 supports native range matching, i.e., matching a field to an interval, the Intel Tofino™ architecture restricts range matches to fields of at most $20\,\mathrm{bit}$. As a result, the full $32\,\mathrm{bit}$ packet index cannot be matched using hardware-supported range operators. To overcome this limitation, we employ a range-to-ternary conversion algorithm [35]. The algorithm decomposes a single numeric interval with a lower and an upper bound into a set of multiple ternary (wildcard) entries whose union covers the original interval. This enables efficient matching over the entire $32\,\mathrm{bit}$ space at the cost of additional MAT entries. The resulting increase in table footprint and its implications for pattern shaping are evaluated in Section V-A3.

C. Pattern Functions for Traffic Shaping

P4TG supports shaping functions f_{pattern} defined over the normalized interval $[0,\ 1)$. In the current version, the control plane provides four built-in functions for traffic patterns: sine, square, sawtooth, and flashcrowd. New functions can be easily added to the control plane. Each function maps a normalized phase value $x \in [0, 1)$ to an amplitude in $[0, 1]$ which directly scales the target rate for the corresponding sampling interval. The function and all its parameters, including the period and sampling factor, are fully integrated into the web-based frontend and REST API for configuration. The definitions of these functions are given below.

Sine Pattern: The sine pattern is a smooth periodic sine wave normalized to $[0, 1]$:

$$f_{\text{sine}}(x) = \tfrac{1}{2}\big(1 + \sin(2\pi x)\big). \tag{8}$$

While sinusoidal patterns are not explicitly discussed in prior work on traffic characterization, we include a sine function to demonstrate the ability to model continuous functions. It also serves as a convenient reference pattern for evaluating shaping accuracy in Section V-A4.

Square Wave Pattern: The square wave pattern is a binary on/off pattern with high and low phases where the rate during the low phase is defined by the low parameter, and the high phase lasts until t_{high}:

$$f_{\text{square}}(x, \text{low}, t_{\text{high}}) = \begin{cases} 1, & 0 \le x < t_{\text{high}}, \\ \text{low}, & t_{\text{high}} \le x < 1. \end{cases} \tag{9}$$

This pattern models pulsing or on/off DDoS attacks, and microbursts such as those described in Section III-A1. Three microburst-based attack scenarios using this pattern are evaluated in Section V-B to Section V-D.

Sawtooth Pattern: The sawtooth pattern contains a continuous linear increase with a reset at the period boundary:

$$f_{\text{sawtooth}}(x) = x. \tag{10}$$

It facilitates the determination of the zero-loss throughput [36] of a device under test (DuT) which is described and applied in Section V-B2.

Flashcrowd Pattern: The flashcrowd pattern has a rate of zero until t_0, followed by a linear ramp-up until t_1, followed by exponential decay with the decay rate λ:

$$f_{\text{flash}}(x, t_0, t_1, \lambda) = \begin{cases} 0, & 0 \le x < t_0, \\ \dfrac{x - t_0}{(t_1 - t_0)}, & t_0 \le x < t_1, \\ \min\left(1,\ e^{\left(-\lambda \dfrac{x - t_1}{1 - t_1}\right)}\right), & t_1 \le x < 1. \end{cases} \tag{11}$$

This pattern models flashcrowd events and DDoS attacks.

V. Evaluation

In this section, we first evaluate the scalability of the proposed mechanism. Next, we demonstrate various applications, including a buffer capacity measurement, a zero-loss throughput determination, and microburst-based attacks on a UDP target, on a switch, and on a shared TCP link.

A. Scalability

We evaluate the maximum configurable period for pattern shaping, the impact of the sampling factor k, the number of required entries in the pattern shaping MAT, and the accuracy of the generated patterns at terabit rates.

1) Maximum Period for Patterns: For pattern periodicity, the number of packets in a period is calculated based on the packet rate as described in Section IV-A. This value determines the wrap-around point of the packet index register in the data plane. Since this register is $32\,\mathrm{bit}$ wide, it can represent at most 2^{32} distinct packet indices. Consequently, the maximum number of packets per period, and therefore the maximum achievable period length, is bounded by this limit. Given a packet rate R_{max} (in pps), the maximum configurable period is

$$T_{\text{max}} = \frac{2^{32}}{R_{\text{max}}}. \tag{12}$$

Table II lists example values of T_{max} for different packet rates, and shows that P4TG supports pattern periods that match those reported for real-world DDoS attacks in Section III-A2.

2) Impact of the Sampling Factor: The control plane samples each periodic pattern into k discrete intervals. A higher sampling factor improves the accuracy of the shaped pattern but also increases the required MAT space. In this experiment, we evaluate how different sampling factors k affect the pattern shape. For that purpose, we configure four $100\,\mathrm{Gbit/s}$ streams for generation in parallel with different sampling factors $k \in \{8, 16, 32, 64\}$ and a period of $T = 40\,\mathrm{s}$. We use a sine-wave pattern as it most clearly exposes differences in sampling accuracy, and measure the generated TX rate with P4TG. Traffic is generated for $60\,\mathrm{s}$ and the experiment is repeated ten times. The aggregated results are shown in Figure 2.

TABLE II: Maximum period T_{max} for different link rates and frame sizes.

Rate (Gbit/s)	**L2 frame size (Byte)**	**Rate R_{max} (Mpps)**	**Maximum period T_{max} (s)**
100	1518	8.12	528
100	512	23.46	182
100	64	148.15	28
400	1518	32.4	132
400	512	93.75	45
294*	64	437.5	9

*Maximum rate P4TG can generate with 64 B frames on a single port.

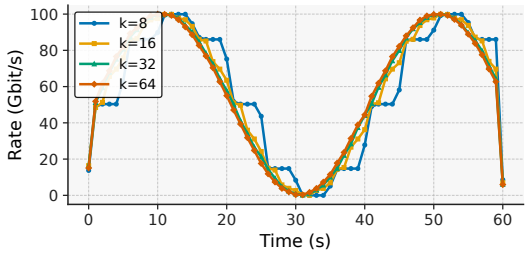


Fig. 2: Measured TX rate in P4TG with different sampling factors.

Figure 2 illustrates that with a low sampling factor of $k \in \{8, 16\}$, the resulting waveform deviates noticeably from the ideal sine shape. Sampling factors $k \in \{32, 64\}$ produce smooth waveforms that closely match the target function. Therefore, $k = 32$ serves as a practical default. However, this value can be configured using the web frontend, or the REST API.

3) MAT Resource Usage for Pattern Shaping: The control plane divides the configured period into k equally sized sampling intervals and installs them into the shaping MAT of the data plane. The required number of MAT entries varies depending on the configured parameters, i.e., the sampling factor k, the traffic rate R_{max}, and the period T. Since the Intel Tofino™ architecture supports range matching only for fields up to 20 bit, interval matching over the full 32 bit packet index requires the range-to-ternary conversion described in Section IV-B3. This conversion expands each numeric interval into multiple ternary entries whose union covers the same range, thereby increasing the number of required MAT entries. Figure 3 shows an overview of the number of required ternary MAT entries for $R_{\text{max}} \in \{10, 50, 100, 150\}$ Mpps, $k \in \{16, 32, 64, 128\}$, and $T \in \{20, 40, 60\}$ s.

Missing data points in Figure 3, e.g., for $T = 60$ and $R_{\text{max}} = 150\,\text{Mpps}$, result from the period exceeding the maximum duration as evaluated in Section V-A1. Generally, the number of required MAT entries increases with the period length and packet rate, and ranges from a minimum of 345 for $k = 16, T = 20\,\text{s}$ to a maximum of 2913 for $k = 128, T = 40\,\text{s}$. Using fewer sampling intervals significantly reduces the required number of MAT entries. For $k = 32$, the number of

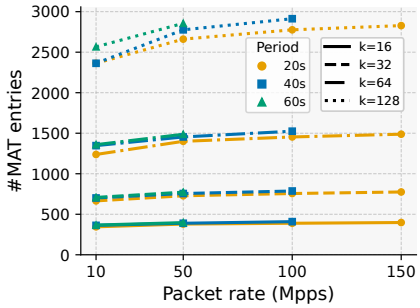


Fig. 3: Required number of ternary MAT entries based on the packet rate R_{max}, the period T, and the sampling factor k.

required entries ranges from 665 to 773 while still providing sufficient shaping granularity for the evaluated patterns as shown in Section V-A2. Accordingly, P4TG uses $k = 32$ as the default configuration. The current shaping MAT provides space for 40000 entries which is sufficient to accommodate multiple patterns simultaneously.

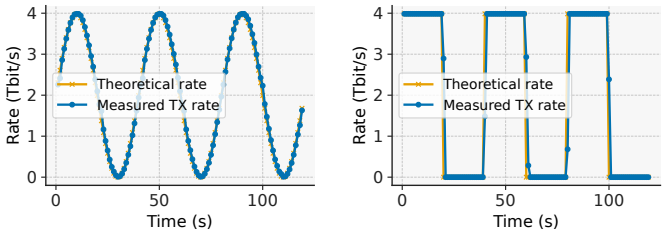
4) Pattern-Rate Accuracy at High Throughput: In this section, we assess the accuracy of the generated pattern shapes. We configure P4TG to generate $4\,\text{Tbit/s}$ of traffic across ten $400\,\text{Gbit/s}$ ports using $1518\,\text{B}$ frames. This is the maximum rate P4TG can generate, but pattern shaping can be applied at any lower rate. We evaluate four pattern functions, i.e., $f_{\text{pattern}} \in \{f_{\text{sine}}, f_{\text{square}}, f_{\text{sawtooth}}, f_{\text{flash}}\}$, each with a period of $40\,\text{s}$. For f_{flash}, we use the parameters $t_0 = 0.2$, $t_1 = 0.25$, and $\lambda = 4$. Traffic is generated for $120\,\text{s}$ resulting in three full cycles per pattern, and the experiment is repeated ten times. The traffic is looped back to P4TG, where the L1 TX and RX rates are measured. The theoretical rates and the aggregated per-port results are shown in Figure 4. As the TX and RX rates are identical for all experiments, only the TX rates are displayed for readability.

Figure 4 illustrates that the measured L1 rates closely follow the configured patterns across all three cycles and all pattern types. To quantify the similarity between the theoretical and observed pattern, we compute the normalized root-mean-square deviation (NRMSD), defined as

$$\text{RMSD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{13}$$

$$\text{NRMSD} = \frac{\text{RMSD}}{\max(\hat{y}_i) - \min(\hat{y}_i)} \tag{14}$$

An NRMSD of 0.1 indicates that the measured values deviate from the ideal pattern by $10\,\%$ of the pattern's total rate range. Across the evaluated patterns, the NRMSD remains low, with $\text{NRMSD}(f_{\text{sine}}) = 0.02$, $\text{NRMSD}(f_{\text{square}}) = 0.14$, $\text{NRMSD}(f_{\text{sawtooth}}) = 0.09$, and $\text{NRMSD}(f_{\text{flash}}) = 0.04$. For f_{square}, the NRMSD is slightly higher than for the other patterns because of its abrupt transitions between minimum and maximum rates. Because P4TG reports rates as averages over fixed $1\,\text{s}$ measurement intervals, intervals that contain a rate



(a) Sine. (b) Square.

(c) Sawtooth. (d) Flashcrowd.

Fig. 4: Generated traffic patterns measured in P4TG.

Fig. 5: The testbed setup.

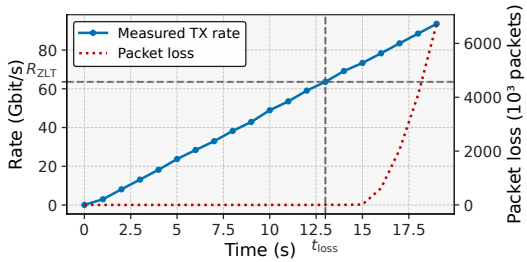


Fig. 6: Zero-loss throughput determination of the receiving application.

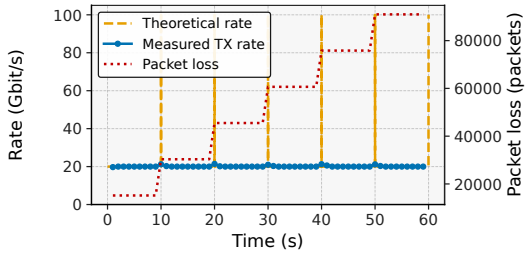


Fig. 7: Configured microburst rate, measured TX rate, and packet loss.

transition reflect an average of the high- and low-rate phases, producing intermediate values at the transition points. This smoothing effect is limited to the measurement process and does not affect the traffic generation. Overall, the results show that P4TG accurately reproduces diverse rate profiles even at an aggregate throughput of up to $4\,\mathrm{Tbit/s}$, demonstrating that fine-grained pattern shaping is feasible at the highest supported line rates.

B. Microburst Impact on a UDP Target

In this section, we demonstrate a zero-loss throughput determination, and then analyze the behavior of a UDP target under periodic microburst traffic.

1) Testbed: The testbed for the microburst attack emulation is illustrated in Figure 5. Traffic is generated by P4TG using $1518\,\mathrm{B}$ frames and forwarded over a $100\,\mathrm{Gbit/s}$ link to a server equipped with a ConnectX-5 NIC. On the server, a kernel-bypass application receives the generated UDP traffic and returns all packets to P4TG for measurement. It is not provisioned to sustain $100\,\mathrm{Gbit/s}$ and can only process up to a limited rate before packet loss occurs. This makes it a suitable target for testing short-lived overload events.

2) Zero-Loss Throughput Determination: We first determine the maximum rate the application can receive without loss. RFC 2544 [36] defines the zero-loss throughput R_{ZLT} as the highest rate at which a DuT forwards packets without packet loss. The zero-loss throughput is determined by repeatedly adjusting the generated rate and observing packet loss. Using pattern shaping, this procedure can be performed in a single continuous experiment. For that purpose, P4TG generates a sawtooth pattern with a peak rate of $R_{\mathrm{max}} = 100\,\mathrm{Gbit/s}$ and a period of $20\,\mathrm{s}$. Generated traffic is returned to P4TG where packet loss is measured. The experiment is repeated ten times. The rate at which the first loss occurs defines the receiver's saturation point R_{ZLT}. Figure 6 shows the results of the zero-loss throughput measurement using the sawtooth pattern.

As shown in Figure 6, the application experiences packet loss at $t_{\mathrm{loss}} = 13\,\mathrm{s}$. It can therefore process approximately $R_{\mathrm{ZLT}} = 65\,\mathrm{Gbit/s}$ without dropping packets. Any higher rate results in loss. While this experiment establishes a baseline for the processing capability of the server application, it also showcases the use of the sawtooth pattern to probe for the maximum throughput.

3) Packet Loss due to Microbursts: Second, we test the behavior of the server application under microbursts. A background CBR stream of $20\,\mathrm{Gbit/s}$ is generated to emulate steady user traffic. In addition, a second stream introduces periodic microbursts using a square wave pattern with the parameters $R_{\mathrm{max}} = 80\,\mathrm{Gbit/s}$, a burst duration of $t = 500\,\mu\mathrm{s}$, and a period of $T = 10\,\mathrm{ms}$. Each burst therefore spikes to a combined rate of $100\,\mathrm{Gbit/s}$ for $500\,\mu\mathrm{s}$ every $10\,\mathrm{ms}$. Traffic is generated for $60\,\mathrm{s}$ and the experiment is repeated ten times. We observed packet loss while the measured 1-second averaged TX rate only showed $24\,\mathrm{Gbit/s}$ of the combined $100\,\mathrm{Gbit/s}$. To visualize the pattern more clearly, we repeat the experiment with longer bursts ($t = 10\,\mathrm{ms}$, $T = 10\,\mathrm{s}$) that are visible on a second scale. Figure 7 shows the configured (theoretical) rate, the measured TX rate, and the observed packet loss.

While the bursts exceed the receiver's $65\,\mathrm{Gbit/s}$ saturation

point, the 1-second averaged TX rate only shows small fluctuations of up to 21.6 Gbit/s. This is because the rate counters exposed by P4TG are averaged over 1 s intervals, similar to other approaches [25], [26], which smooths out the 10 ms spikes. As a result, the burst is effectively hidden in the reported rate, and the overload becomes visible only through the packet-loss peaks.

C. Buffer Capacity Measurement and Microburst Impact on Forwarding Device

RFC 8239 [37] describes a benchmarking methodology for data center networks, including buffer and microburst testing. We demonstrate an application of pattern shaping for measuring a device's buffer capacity through controlled burst overload, similar to RFC 8239. We connect P4TG to an Intel Tofino™ 2 switch via a 400 Gbit/s link, returning traffic over a 100 Gbit/s bottleneck link. P4TG generates square-wave traffic with 64 B frames, period $T = 20\,\text{ms}$, and L2 burst rate $R^{L2}_{\text{max}} = 77.24\,\text{Gbit/s}$ (101 Gbit/s on L1). Excess traffic from the burst accumulates in the switch's buffer. At a critical burst duration t_{crit}, the buffer fills completely and packet loss begins. We vary the burst duration t, i.e., the duration of the high phase of the square wave, to identify t_{crit} where packet loss first occurs. The buffer capacity C can then be derived as

$$C = (R^{L2}_{\text{max}} - R^{L2}_{B}) \cdot t_{\text{crit}} \tag{15}$$

where R^{L2}_{max} is the L2 burst rate, R^{L2}_{B} is the L2 bottleneck bandwidth, and t_{crit} is the shortest burst duration that causes packet loss. Through binary search, we identify $t_{\text{crit}} \approx 19.6\,\text{ms}$. Using Equation 15, with $R^{L2}_{\text{max}} = 77.24\,\text{Gbit/s}$, 64 B frames, and thus, $R^{L2}_{B} = 76.19\,\text{Gbit/s}$, we calculate an effective buffer capacity of approximately 2.57 MB of frame data. With 64 B frames requiring one 176 B cell each in the Tofino™ 2, this results in $\frac{2.57\,\text{MB}}{64\,\text{B}} = 40196$ cells. This value approximately corresponds to the configured buffer size of 40156 cells in the Tofino™ 2.

Next, we leverage the measured buffer capacity to demonstrate a microburst attack on the switch. With a rate of $R_{\text{max}} = 294\,\text{Gbit/s}$[1] and the identified buffer size of 2.57 MB, the theoretical critical burst duration is approximately 139 μs according to Equation 15. In our measurement, we observed packet loss with a minimum microburst duration of $t_{\text{crit}} = 138\,\mu\text{s}$ using a rate of 294 Gbit/s, confirming our previous buffer capacity measurement. Further, while we observed packet loss from buffer exhaustion in the switch, the 1-second rate averaging revealed only 4.44 Gbit/s (1.5%) of the 294 Gbit/s. This demonstrates how attackers can cause forwarding-plane buffer exhaustion while remaining nearly invisible to conventional monitoring systems.

D. Impact of Microbursts on TCP Traffic

We demonstrate how P4TG-generated microbursts degrade TCP performance while remaining mostly undetectable by conventional rate monitoring.

[1]Maximum rate P4TG can generate with 64 B frames on a single port.

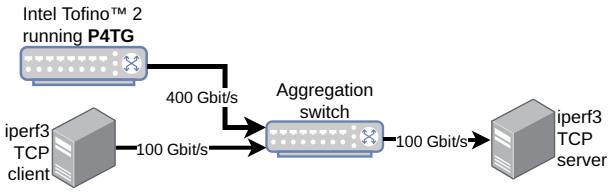
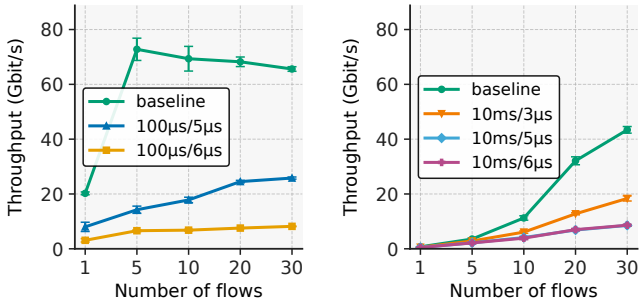


Fig. 8: Testbed for evaluating the impact of microbursts on TCP throughput.



(a) No delay. (b) Delay of 10 ms.

Fig. 9: Impact of microbursts on TCP traffic.

1) Testbed: The testbed shown in Figure 8 consists of two servers running an `iperf3` client and server, P4TG on an Intel Tofino™ 2 switch, and a second Intel Tofino™ 2 for traffic aggregation.

P4TG is connected via a 400 Gbit/s link and generates UDP microbursts with peak rate R_{max} and a frame size of 64 B. The generated traffic targets a server running `iperf3` which receives TCP Cubic traffic from a client. The aggregation switch combines both traffic streams over a single 100 Gbit/s bottleneck link. We vary the number of concurrent TCP flows n, the period T, and the burst duration t, denoting each configuration as T/t. Each scenario runs for 60 s with ten repetitions.

2) TCP Throughput under Microburst Attacks: Figure 9 shows the TCP throughput reported by the `iperf3` server with 95% confidence intervals.

In Figure 9a, for $n \geq 5$, the baseline throughput ranges from 60 to 70 Gbit/s. With microbursts of $R_{\text{max}} = 294\,\text{Gbit/s}$, and microburst parameters 100 μs/6 μs, throughput drops below 8 Gbit/s. The buffer capacity of the aggregation switch identified in Section V-C saturates approximately at 139 μs of traffic at an L1 rate of 294 Gbit/s, and at 78 μs with the TCP baseline of 70 Gbit/s included. The microbursts are shorter than the switch's buffer capacity, indicating that there is no packet loss on the bottleneck link and the throughput degradation results from server overload rather than a switch buffer overflow. The 1-s rate averaging measured only 17.6 Gbit/s (9%) of the 294 Gbit/s for a microburst with 100 µs/6 µs, mostly masking the attack.

The testbed's low RTT of approx. 3 µs allows TCP to recover quickly from bursts. To model realistic conditions, we add

a constant $10\,\mathrm{ms}$ delay using `netem` at the client. With an increased RTT, a single microburst every $10\,\mathrm{ms}$, i.e., once per RTT, is sufficient to disrupt TCP's congestion control and degrade throughput. We therefore increase the microburst period to $10\,\mathrm{ms}$. Without added delay, $T = 10\,\mathrm{ms}$ showed negligible TCP impact. As shown in Figure 9b, the baseline drops to at most $45\,\mathrm{Gbit/s}$ compared to Figure 9a due to the added delay. Further, Figure 9b shows that the TCP throughput degrades to $8\,\mathrm{Gbit/s}$, similar to the no-delay case with shorter period, while using the same burst durations. Crucially, rate averaging measured only $0.1\,\mathrm{Gbit/s}$ (0.03%) of the $294\,\mathrm{Gbit/s}$ bursts, rendering the attack invisible to conventional monitoring.

VI. Conclusion

In this work, we extended P4TG with a mechanism for shaping generated traffic with periodic, realistic traffic patterns. Unlike generators limited to either CBR traffic or low generation rates, our design supports fine-grained shaping of periodic traffic patterns, such as sine, sawtooth, square, and flashcrowd, at up to $4\,\mathrm{Tbit/s}$. Pattern shaping is achieved by control plane sampling and normalization, combined with data plane enforcement at line rate. While the current implementation targets Intel Tofino™ hardware, the shaping logic uses common P4 constructs, i.e., registers, meters, and MATs. Therefore, the mechanism is portable to other P4-programmable targets, such as the emerging X2 platform of Xsight [38]. Our evaluation demonstrates that the generated traffic closely follows the configured patterns, even at multi-terabit rates. The scalability analysis quantified the supported pattern periods and table resource requirements, confirming that the approach remains practical across a wide range of configurations. We further showed that pattern shaping enables practical use cases such as a single-run zero-loss throughput determination using a sawtooth pattern, following the RFC 2544 [36] methodology, and a buffer capacity measurement with square-wave probing. Finally, we demonstrated three microburst-based attack scenarios in which short, high-intensity bursts overload a UDP target, a switch buffer, and a shared TCP link while evading detection by monitoring systems with second-scale sampling. By making periodic pattern shaping practical on programmable switch hardware, this work establishes P4TG as a platform for controlled and repeatable studies of networked systems under DDoS attacks, burst-load effects, and time-varying traffic patterns.

References

[1] S. Lindner et al. P4TG: 1 Tb/s Traffic Generation for Ethernet/IP Networks. *IEEE Access*, 11:17525–17535, February 2023.

[2] S. Behal et al. Characterizing DDoS Attacks and Flash Events: Review, Research Gaps and Future Directions. *Computer Science Review*, 25:101–114, 2017.

[3] I. Ari et al. Managing Flash Crowds on the Internet. In *IEEE/ACM MASCOTS*, pp. 246–249, 2003.

[4] A. A. Alves da Silva et al. A Proposal to Distinguish DDoS Traffic in Flash Crowd Environments. *IJISP-IGI*, 16(1), 2022.

[5] A. Bremler-Barr et al. DDoS Attack on Cloud Auto-Scaling Mechanisms. In *IEEE ICC*, 2017.

[6] N. Asadi et al. WaveSurfer – Scheduling Irregular Pulsing Attacks on Microservice Autoscaling. In *ACM SIGCOMM Posters and Demos*, pp. 55–57, 2025.

[7] X. Li et al. DNSBomb: A New Practical-and-Powerful Pulsing DoS Attack Exploiting DNS Queries-and-Responses. In *IEEE SP*, pp. 4478–4496, 2024.

[8] J. Yuan et al. Monitoring the Macroscopic Effect of DDoS Flooding Attacks. *IEEE TDSC*, 2(4):324–335, 2005.

[9] D. Shan et al. Observing and Mitigating Micro-Burst Traffic in Data Center Networks. *IEEE/ACM ToN*, 28(1):98–111, 2020.

[10] Q. Zhang et al. High-Resolution Measurement of Data Center Microbursts. In *IEEE IMC*, pp. 78–85, 2017.

[11] R. Joshi et al. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *APSYS*, 2018.

[12] L. Almeida et al. WAVE – Um gerador de cargas múltiplas para experimentação em redes de computadores. In *SBRC*, pp. 9–16, 2023.

[13] P. Emmerich et al. MoonGen: A Scriptable High-Speed Packet Generator. In *IEEE IMC*, Tokyo, Japan, October 2015.

[14] TRex Team. TRex – Realistic Traffic Generator, 2022. visited on 2025-11-28.

[15] R. Kundel et al. P4STA: High Performance Packet Timestamping with Programmable Packet Processors. In *IEEE/IFIP NOMS*, 2020.

[16] F. G. Costa et al. PIPO-TG: Parameterizable High-Performance Traffic Generation. In *IEEE NOMS*, 2024.

[17] P. Bosshart et al. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM CCR*, 44(3):87–95, July 2014.

[18] Intel. Intel Tofino Native Architecture—Public Version. https://github.com/barefootnetworks/Open-Tofino. visited on 2025-01-09.

[19] S. Lindner et al. GitHub: P4TG. https://github.com/uni-tue-kn/P4TG. visited on 2025-12-10.

[20] F. Ihle et al. Enhancements to P4TG: Protocols, Performance, and Automation. In *KuVS NetSoft*, April 2025.

[21] F. Ihle et al. Enhancements to P4TG: Histogram-Based RTT Monitoring in the Data Plane. *ReNeSys*, September 2025.

[22] R. Mayr et al. High-Throughput Electrical Switching System on a Space-Grade Adaptive Computing Platform. In *EDHPC*, 2025.

[23] German Federal Agency for Public Safety Digital Radio. Die BDBOS verstärkt ihre Bemühungen im Open Source Umfeld. https://www.linkedin.com/posts/bdbos_bdbos-opensource-p4tg-activity-7385570407245922304-19Uk, 2025. visited on 2025-12-04.

[24] D. Kopp et al. DDoS on Repeat: Measuring Pulse-Wave DDoS in the Wild. In *ITC*, 2025.

[25] J. D. Case et al. RFC3410: Introduction and Applicability Statements for Internet-Standard Management Framework, December 2002.

[26] S. Panchen et al. RFC3176: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, September 2001.

[27] K. Gao et al. Bandwidth-efficient Microburst Measurement in Large-scale Datacenter Networks. In *APNet*, pp. 14–20, 2023.

[28] H. Zheng et al. μMon: Empowering Microsecond-level Network Monitoring with Wavelets. In *ACM SIGCOMM*, pp. 274–290, 2024.

[29] Z. M. Mao et al. Analyzing Large DDoS Attacks Using Multiple Data Sources. In *ACM SIGCOMM LSAD*, pp. 161–168, 2006.

[30] Cloudflare. DDoS Threat Report for 2025 Q2. https://radar.cloudflare.com/reports/ddos-2025-q2. visited on 2025-11-28.

[31] Cloudflare. DDoS Threat Report for 2025 Q3. https://radar.cloudflare.com/reports/ddos-2025-q3. visited on 2025-12-09.

[32] Sean Whalen (Microsoft). Defending The Cloud: Azure Neutralized a Record-Breaking 15 Tbps DDoS Attack. https://techcommunity.microsoft.com/blog/azureinfrastructureblog/defending-the-cloud-azure-neutralized-a-record-breaking-15-tbps-ddos-attack/4470422. visited on 2025-12-09.

[33] F. Ihle et al. P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking. *IEEE TNSM*, 21(5):5273–5290, 2024.

[34] J. Heinanen et al. RFC2698: A Two Rate Three Color Marker, September 1999.

[35] P. Gupta et al. Algorithms for Packet Classification. *IEEE Network*, 15(2):24–32, 2001.

[36] S. Bradner et al. RFC2544: Benchmarking Methodology for Network Interconnect Devices, March 1999.

[37] L. Avramov et al. RFC8239: Data Center Benchmarking Methodology, August 2017.

[38] Xsight Labs. X2 Switch. https://xsightlabs.com/switches/x2. visited on 2026-03-26.