GhostView: Enabling Deep Visibility in Programmable Data Planes with Minimal Server Overhead

Francisco Germano Vogt[1], Leonardo Henrique Guimarães[1], Zhiheng Yang[3], Fabricio Eduardo Rodriguez Cesen[4], Sergio Rossi Brito da Silva[1], Marcelo Caggiani Luizelli[2], Chrysa Papagianni[3], Christian Esteve Rothenberg[1]

[1]Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brazil

[2]Universidade Federal do Pampa (UNIPAMPA), Alegrete, RS, Brazil

[3]University of Amsterdam (UvA), Amsterdam, Netherlands

[4]Telefonica Research, Barcelona, Spain

f234632@dac.unicamp.br, z.yang@uva.nl, l247225@dac.unicamp.br, fabricio.rodriguezcesen@telefonica.com, s217042@dac.unicamp.br, marceloluizelli@unipampa.edu.br, c.papagianni@uva.nl, chesteve@unicamp.br

Abstract. *The rise of programmable data planes and high-performance ASICs, such as Intel Tofino, has enabled network processing at multi-terabit rates. However, gaining deep, flow-level visibility into 100 Gbps traffic remains a significant challenge, often requiring expensive server-side hardware or incurring substantial packet overhead via In-band Network Telemetry (INT). This paper presents GhostView, a lightweight and non-intrusive monitoring toolkit designed for high-performance network experimentation. GhostView strategically utilizes the often-underutilized egress pipeline of P4 switches to maintain per-flow statistics, including throughput, queuing latency, and occupancy, directly in stateful hardware registers. By employing an asynchronous reporting mechanism with custom P4 headers and a multi-threaded Python management plane, GhostView enables granular monitoring of high-speed streams with minimal server-side CPU overhead. Our demonstration shows how GhostView can provide high-fidelity real-time insights at 100 Gbps, making it a powerful asset for researchers. The toolkit is fully open-source, providing both a CLI and a web-based interface for immediate data visualization and logging.*

1. Introduction

The emergence of programmable data planes, empowered by the P4 language and high-performance ASICs like Intel Tofino, has revolutionized network experimentation [Bosshart et al. 2014, Bosshart et al. 2013]. These devices enable researchers to implement complex protocols and custom processing logic at line rates exceeding Tbps, with per-port speeds of up to 400 Gbps. However, this massive throughput creates a "visibility gap" in the network experimentation. While hardware counters provide coarse-grained metrics (e.g., total port throughput), gaining granular, flow-level insights—such as per-flow throughput, processing latency, jitter, and queueing information, remains a significant challenge in high-speed environments.

Traditionally, deep packet inspection and fine-grained monitoring are offloaded to dedicated servers using frameworks like DPDK [DPDK nd] or XDP [Høiland-Jørgensen et al. 2018]. While powerful, these solutions are resource-intensive. Processing 100 Gbps of traffic requires a substantial number of dedicated CPU cores just for packet capture and parsing, often making it cost-prohibitive or physically impossible for small-scale testbeds to monitor multi-terabit traffic in real-time. Furthermore, internal switch metadata, such as precise queuing delay or the specific egress queue used, is lost unless explicitly embedded into the packets, adding further overhead and complexity.

To address these challenges, we present GhostView, a hybrid P4-Python toolkit designed for high-performance network monitoring with minimal overhead. GhostView strategically utilizes the often-underutilized egress pipeline of programmable switches to maintain flow-level statistics directly in the data plane. By offloading the heavy lifting of state management to the switch hardware, the companion Python application can asynchronously poll metrics at configurable granularities. This approach allows even low-capacity management servers to gain deep visibility into 100 Gbps streams, providing real-time visualization and logging for post-experiment analysis.

2. Motivation and Related Work

2.1. Motivation

The primary motivation for GhostView stems from the increasing difficulty of monitoring high-speed networks without compromising performance or incurring prohibitive costs. As network interfaces move from 10 Gbps to 100 Gbps and beyond, traditional software-based monitoring (e.g., using DPDK or XDP) faces a significant "processing wall." At 100 Gbps, a server has only 5.12 nanoseconds to process a 64-byte packet. Processing such traffic requires handling hundreds of millions of packets per second to extract flow-level metrics, which requires dozens of dedicated CPU cores, which are often unavailable in research testbeds.

Furthermore, passive monitoring from a server cannot natively capture internal switch states, such as instantaneous queue depth or the specific egress queue assigned to a packet, unless this data is explicitly added to the packet header (e.g., via In-band Network Telemetry (INT) [Vogt et al. 2022]). However, adding such metadata increases the packet size, causing bandwidth overhead and potentially affecting the very performance metrics being measured.

Figure 1 analytically illustrates this pressure. As link rate increases, the per-packet processing budget decreases inversely with the line rate, while the packet arrival rate grows proportionally. This makes purely software-based packet-by-packet monitoring increasingly costly at 100 Gbps and beyond.

Existing monitoring mechanisms expose an unfavorable trade-off between visibility and cost. Hardware counters are lightweight and readily available in switches, but they typically provide only coarse-grained statistics. Software-based monitoring frameworks offer greater flexibility and richer parsing capabilities, but require substantial server-side resources to sustain high packet rates. INT enables detailed in-network visibility by carrying telemetry information inside packets, but this comes at the cost of additional packet overhead and sink-side processing.

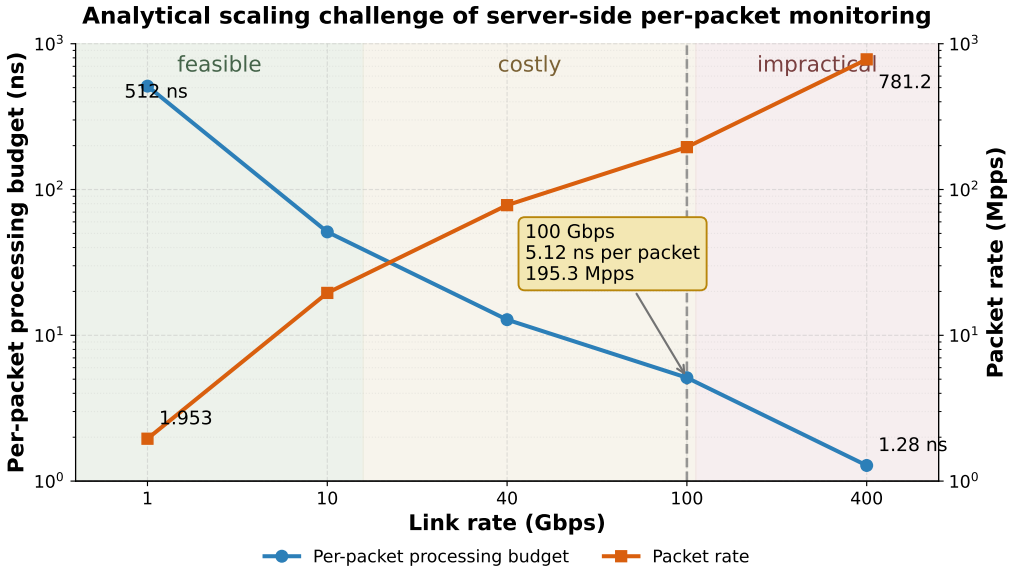

Figura 1. Scaling challenge of server-side packet-by-packet monitoring

GhostView is motivated by the need for a "middle ground:" a tool that provides the depth of hardware-level telemetry with the ease of use of a Python-based software suite, all while keeping the server load minimal. Figure 2 positions GhostView in this design space. GhostView aims to combine deeper visibility than conventional hardware counters with substantially lower server-side burden than software-only monitoring, while avoiding the continuous packet/header overhead imposed by INT.

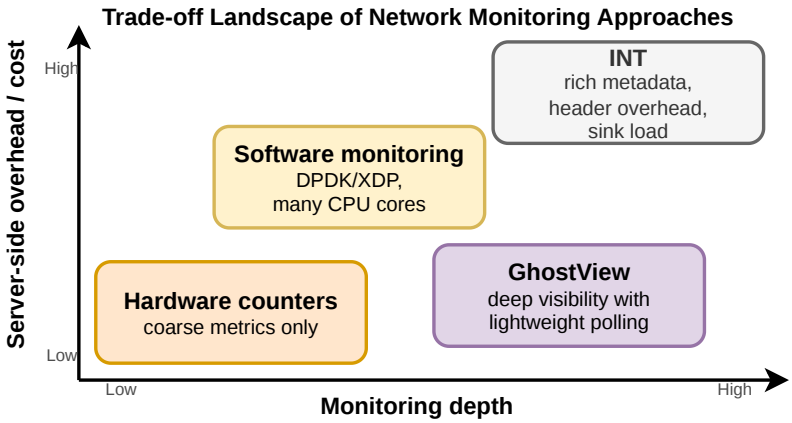

Figura 2. Conceptual trade-off landscape of network monitoring approaches

2.2. Related Work

Software-based Monitoring and Packet Generation. MoonGen [Emmerich et al. 2015]: A high-speed packet generator that uses DPDK. It highlights the CPU intensity required to maintain precision at 10 Gbps, serving as a baseline for why software-only monitoring struggles at 100 Gbps. FastClick [Barbette et al. 2015]: An evolution of the Click modular router optimized for high speed. Like DPDK-based tools, it requires significant hardware resources to avoid packet drops during analysis.

In-band Network Telemetry (INT). P4-INT [Kim et al. 2015]: The standard for embedding telemetry data into packet headers. While precise, INT requires a "sink" capable of stripping and processing these headers at line rate, often shifting the bottleneck from the switch to the monitoring server. P4Visor [Zheng et al. 2018]: A tool that adds "visibility" to P4 programs but focuses on testing the P4 logic itself rather than long-term, lightweight flow statistics for performance evaluation.

Sketch-based and hardware-efficient monitoring. UnivMon [Liu et al. 2016]: Proposes a universal sketch for monitoring multiple metrics. While efficient in memory, sketches often require complex decoding and don't provide the direct, per-flow "raw"metrics that GhostView aims to expose. NitroSketch [Liu et al. 2019]: An optimized sketching framework that reduces the update overhead in the data plane, focusing on heavy-hitter detection rather than comprehensive queueing and latency analysis. FlowRadar [Li et al. 2016]: Uses invertible Bloom lookup tables to maintain per-flow counters. It is powerful but can be complex to integrate into existing egress pipelines compared to GhostView's modular approach. ElasticSketch [Yang et al. 2018]: Adaptively allocates resources for different traffic distributions. It is excellent for traffic engineering but less focused on the "tooling/experimentation"aspect for researchers.

Control-Plane Assisted and Hybrid Approaches. Marple [Narayana et al. 2017]: A performance monitoring system that uses a dedicated query language. It is highly expressive but requires significant changes to the switch architecture, whereas GhostView is designed as a plug-and-play module. BeauCoup [Chen et al. 2020]: A system for counting distinct flows efficiently in the data plane. While GhostView also tracks flows, it extends this to include egress-specific metadata (queue timestamps) which BeauCoup does not prioritize.

3. GhostView Architecture and Workflow

The GhostView toolkit is designed to provide high-fidelity network visibility with minimal computational footprint. The architecture is divided into two main components: the P4-based monitoring module, which resides within the programmable switch, and the Python-based management application, which runs on a commodity server. Figure 3 illustrates the GhostView architecture overview, highlighting its main components.

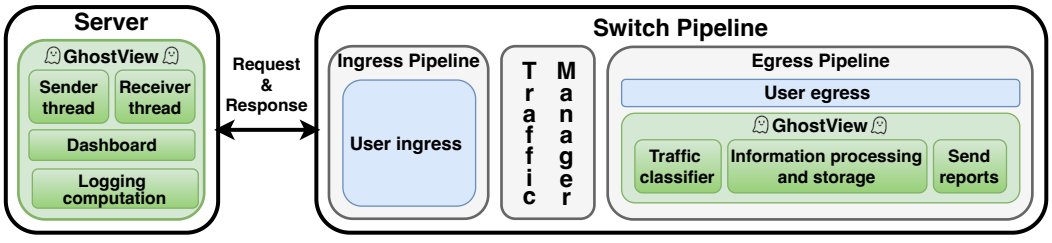

Figura 3. GhostView architecture overview

3.1. P4 Data Plane Module

The core of GhostView is implemented in the egress pipeline of the P4 architecture. By placing the monitoring logic after the Traffic Manager (TM), GhostView gains native access to critical queuing metadata that is typically unavailable in the ingress pipeline.

The module is designed to be non-intrusive, utilizing hardware resources (SRAM/TCAM) that are often left unused by the primary switching logic.

- **Stateful Storage:** GhostView maintains a set of registers to store statistics for thousands of flows and all active ports, where both parameters are user-defined during compilation to match the available memory. Flows are identified through a hardware-accelerated hash function based on a flexible N-tuple (e.g., source and destination IP addresses, protocol, and L4 ports).
- **Monitoring Metrics:** For each tracked entry, the module stores: (i) a 64-bit byte counter to prevent overflows at Tbps rates; (ii) a packet counter; (iii) the timestamp of the last packet; (iv) the egress queue ID and its instantaneous occupancy (`deq_occupancy`); (v) the enqueue timestamp to calculate queuing latency; and (vi) a sequence number for loss detection.
- **Accounting vs. Reporting:** GhostView distinguishes traffic using a custom EtherType. Normal data packets undergo the *accounting* process, updating the respective registers without latency penalties. Conversely, special monitoring packets generated by the Python application trigger the *reporting* mechanism. These packets carry a specific header that instructs the switch to read the requested register values and mirror them back to the server, bypassing the accounting logic to ensure statistics remain untainted.

3.2. Python Management Application

The management plane is responsible for orchestrating the monitoring process and processing the retrieved data. To maintain high performance and responsiveness, the application utilizes a multi-threaded architecture.

- **Sender and Receiver Threads:** The application spawns dedicated threads for different tasks. The *Sender Thread* manages the polling frequency for each flow or port as specified in the configuration. This allows for multi-granular monitoring, where critical flows can be polled at sub-millisecond intervals while background traffic is monitored every second. The *Receiver Thread* asynchronously captures reporting packets, parses the metadata, and computes real-time statistics such as throughput and average latency.
- **User Interface and Logging:** GhostView provides a real-time dashboard in the terminal for live experimentation. Optionally, all collected data can be streamed to a structured log file (e.g., CSV or JSON) for post-mortem analysis and visualization.

3.3. Key Features and Parameters

GhostView is built for flexibility. The toolkit requires a simple configuration file where the user defines:

1. **Network Interfaces:** The dedicated ports for sending and receiving monitoring packets.
2. **Flow Specification:** A detailed list of the flows (IPs/Ports) to be tracked and their respective polling periodicities.
3. **Execution Constraints:** Optional parameters such as total monitoring duration and log file paths.

This decoupled design ensures that even a low-spec management server can monitor high-performance 100 Gbps streams by adjusting the polling granularity to its own processing capacity.

4. GhostView Implementation and Interfaces

The implementation of GhostView follows a hardware-software co-design to ensure high-performance telemetry with minimal resource consumption. The data plane component is implemented entirely in the **egress pipeline** of the P4 architecture. By utilizing the egress stage, the tool gains direct access to post-queuing metadata, such as de-queue timestamps and instantaneous queue depths.

To maintain flow and port statistics, GhostView employs **stateful registers** as its primary storage mechanism. The monitoring logic identifies packets using a specific EtherType, distinguishing between data traffic (accounting) and monitoring probes (reporting). We implemented two custom headers, as shown in Table 1: `monitor_inst_h`, which carries instructions for data collection (including flow and port indices), and `monitor_h`, which encapsulates the retrieved metrics. Due to hardware constraints in the Tofino ASIC, where each register can be accessed only once per packet during processing, each monitoring probe is designed to collect information from exactly one flow and one port per pass.

Tabela 1. GhostView P4 Header Definitions

`monitor_inst_h` (10 bytes)

Field	Width (bits)
`index_flow`	32
`index_port`	32
`port`	9
`padding`	7

`monitor_h` (50 bytes)

Field	Width (bits)
`bytes_flow` / `bytes_port`	64 / 64
`timestamp`	48
`port` / `padding`	9 / 7
`pktLen`	16
`qID` / `qDepth` / `qTime` (`port`)	32 / 32 / 32
`qID` / `qDepth` / `qTime` (`flow`)	32 / 32 / 32

The control plane application is implemented in Python using a multi-threaded architecture. One dedicated thread handles the asynchronous transmission of monitoring probes based on a user-provided configuration file (CSV), which defines the indices and polling frequency for each target. A separate high-speed receiver thread captures the echoed `monitor_h` headers, parses the 48-bit timestamps and 64-bit counters, and computes real-time statistics.

4.1. CLI Interface

The current version of GhostView provides a robust Command Line Interface (CLI) designed for low-latency feedback during experiments. The terminal-based interface displays real-time per-flow throughput and queuing metrics, allowing researchers to monitor high-speed traffic without the overhead of a graphical engine. As shown in Figure 4, the CLI also manages concurrent logging, ensuring that all collected data is saved for post-experiment analysis while providing immediate visual insights into the network's state.

```
Monitor dashboard - 2025-11-02 09:15:24
Showing items seen within last 5.0 seconds. Refresh every 1.0s.

Flows:
ID    LastBytes      LastTs(ns)          Inst(Mbps)  Reg(Mbps)   EWMA(Mbps)  LastSeen(s)
2592    1197772000      1227228023801        8.050       8.054        8.049         0.39
2970     598886000      1227227856410        3.994       4.015        4.012         0.39
1693     119777200      1227228332837        0.799       0.803        0.802         0.39

Ports:
ID    LastBytes      LastTs(ns)          Inst(Mbps)  Reg(Mbps)   EWMA(Mbps)  LastSeen(s)
134     1197772000      1227227856410        7.988       8.030        8.025         0.39
133      598886000      1227228023801        4.025       4.027        4.025         0.39
132      119777200      1227228332837        0.799       0.803        0.802         0.39
```

Figura 4. GhostView current CLI dashboard

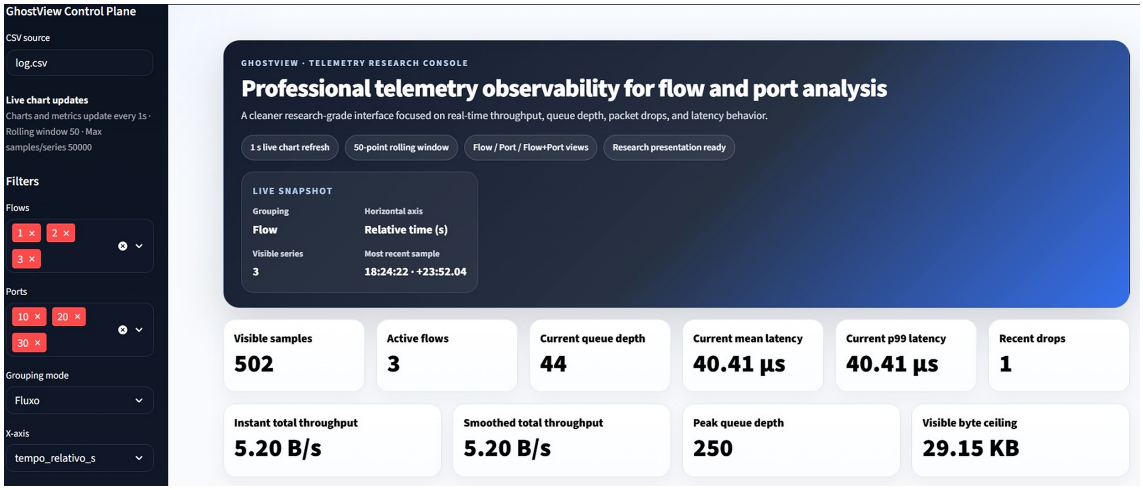

Figura 5. GhostView in-development telemetry dashboard.

4.2. Web-based Interface

To further enhance usability and accessibility, a new web-based interface is currently under development using the **Streamlit** framework. This interface provides an integrated environment for real-time network telemetry visualization and analysis, enabling users to monitor key metrics such as throughput, queue depth, packet drops, and latency across multiple flows and ports through an interactive dashboard.

The interface centralizes configuration and monitoring capabilities within a browser-based environment, eliminating the need for command-line interaction and simplifying usability. As illustrated in Figure 5, the current prototype of the Streamlit dashboard allows users to interactively explore telemetry data by dynamically filtering flows and ports, adjusting visualization and polling parameters, and analyzing system behavior through real-time charts and aggregated indicators.

Additionally, the system adopts a modular and extensible design, allowing future enhancements such as new metrics and visualization components. It is also built as a containerized solution, ensuring portability and ease of deployment across different environments. This makes the interface suitable not only for development and debugging but also for real-time demonstrations and experimental evaluations.

5. Documentation, Code, and Demonstration

The GhostView toolkit is fully open-source and designed to facilitate the adoption of high-performance monitoring in programmable network research. All source code, including the respective license and comprehensive documentation, is available at `https://github.com/FranciscoVogt/GhostView.git`. The repository provides a complete manual for installation and execution, ensuring that the primary claims and performance metrics presented in this paper can be fully reproduced by the community. Additionally, a video tutorial demonstrating the configuration and real-time operation of the tool is also available at the repository. We actively encourage the project's evolution through community contributions, bug reports, and proposals for new telemetry features.

6. Final Remarks and Future Work

In this paper, we introduced **GhostView**, a lightweight and non-intrusive monitoring toolkit designed to bridge the visibility gap in high-performance programmable networks. By strategically offloading state management to the underutilized egress pipeline of P4 switches and employing an asynchronous Python-based polling mechanism, GhostView provides granular, flow-level insights without the prohibitive CPU overhead of traditional software-based monitors. GhostView can maintain high-fidelity metrics—including per-flow throughput, queuing latency, and occupancy—at line rates of 100 Gbps, while consuming minimal server resources. The tool's modular design ensures that it can be easily integrated into existing P4 projects as a plug-and-play telemetry component, making it a valuable asset for researchers conducting large-scale network experiments.

As future work, we plan to extend GhostView's capabilities by implementing automated anomaly detection triggers directly in the data plane, allowing the tool to react to micro-bursts or packet loss in real-time. Furthermore, we are working on expanding the toolkit's compatibility with other P4 architectures beyond the Intel Tofino, such as the Portable Switch Architecture (PSA). We remain committed to the open-source evolution of GhostView, by integrating GhostView's capabilities into our other tools, such as network emulators [Cesen et al. 2023] and traffic generators [Costa et al. 2024, Vogt et al. 2025, Vogt et al. 2024], providing extensive visibility into the traffic generated/processed by these tools.

Acknowledgments

This work was supported by Ericsson Telecomunicações Ltda., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8 (CPE SMARTNESS), 2024/16207-8, 2020/05183-0, and 2023/00794-9. Also, it was supported by Foundation for Research of the State of Rio Grande do Sul (24/2551-0001394-6), CAPES, Brazil-Finance Code 001, and CNPq (405809/2025-5). Finally, this work was also supported by the Dutch National Growth Fund through the 6G flagship project "Future Network Services".

Referências

Barbette, T., Soldani, C., and Mathy, L. (2015). Fast userspace packet processing. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, page 5–16.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. (2013). Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 99–110.

Cesen, F. E. R., Vogt, F. G., De Castro, A. G., and Rothenberg, C. E. (2023). Towards multiple pipelines network emulation with p7. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 290–292. IEEE.

Chen, X., Landau-Feibish, S., Braverman, M., and Rexford, J. (2020). BeauCoup: Answering many network traffic queries, one memory update at a time. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 226–239.

Costa, F. G., Vogt, F. G., Cesen, F. R., de Castro, A. G., Luizelli, M. C., and Rothenberg, C. E. (2024). Pipo-tg: Parameterizable high-performance traffic generation. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–9. IEEE.

DPDK (n.d.). Dpdk. `https://core.dpdk.org/`. Accessed: 2026-03-17.

Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., and Carle, G. (2015). MoonGen: A scriptable high-speed packet generator. IMC '15, page 275–287.

Høiland-Jørgensen, T., Brouer, J. D., Borkmann, D., Fastabend, J., Herbert, T., Ahern, D., and Miller, D. (2018). The express data path: fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 54–66.

Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., Wobker, L. J., et al. (2015). In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, volume 15, pages 1–2.

Li, Y., Miao, R., Kim, C., and Yu, M. (2016). FlowRadar: A better {NetFlow} for data centers. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pages 311–324.

Liu, Z., Ben-Basat, R., Einziger, G., Kassner, Y., Braverman, V., Friedman, R., and Sekar, V. (2019). NitroSketch: robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 334–350.

Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., and Braverman, V. (2016). One sketch to rule them all: Rethinking network flow monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 101–114.

Narayana, S., Sivaraman, A., Nathan, V., Goyal, P., Arun, V., Alizadeh, M., Jeyakumar, V., and Kim, C. (2017). Language-directed hardware design for network performance

monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 85–98.

Vogt, F. G., Da Silva, S. R. B., Cesen, F. E. R., Costa, F. G., Luizelli, M. C., and Rothenberg, C. E. (2025). Tftg: Time fidelity traffic generation through p4/tofino programmable hardware. *IEEE Network*.

Vogt, F. G., Rodriguez, F., Costa, F. G., Luielli, M. C., Rotenberg, C. E., Patra, G., and Pongracz, G. (2024). P4 replay (p4r): Reproducing packet traces and stateful connections at line-rate on your p4-capable hardware. In *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, pages 122–124.

Vogt, F. G., Rodriguez, F., Rothenberg, C., and Pongrácz, G. (2022). Innovative network monitoring techniques through in-band inter packet gap telemetry (ipgnet). In *Proceedings of the 5th International Workshop on P4 in Europe*, pages 53–56.

Yang, T., Jiang, J., Liu, P., Huang, Q., Gong, J., Zhou, Y., Miao, R., Li, X., and Uhlig, S. (2018). Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 561–575.

Zheng, P., Benson, T., and Hu, C. (2018). P4Visor: lightweight virtualization and composition primitives for building and testing modular programs. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 98–111.