Ralf Kundel, Fridolin Siegmund, Jeremias Blendin, Amr Rizk, Boris Koldehofe. *P4STA: High Performance Packet Timestamping with Programmable Packet Processors.*
To appear in the Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS), IEEE, 978-1-7281-4973-820, 2020.

P4STA: High Performance Packet Timestamping with Programmable Packet Processors

Ralf Kundel*, Fridolin Siegmund*, Jeremias Blendin†, Amr Rizk*‡, Boris Koldehofe*
*Multimedia Communications Lab, Technische Universität Darmstadt, Germany
{*ralf.kundel, fridolin.siegmund, boris.koldehofe*}*@kom.tu-darmstadt.de*
†Barefoot Networks, Santa Clara, CA, USA
jblendin@barefootnetworks.com
‡Universität Ulm, Germany
amr.rizk@uni-ulm.de

***Abstract*—QoS requirements of current network control and management applications require the ability to conduct precise measurements of network elements, including switches, routers and Virtual Network Functions (VNFs). State-of-the-art network switches have a forwarding delay of $1\mu s$ and below and offer high bandwidths of hundreds Gigabits per second. This imposes high time accuracy and loss-detection requirements on measurement equipment that are not met by existing, software-based measurement tools. The use of specialized tools, meeting these requirements, is restricted by limited flexibility and high cost.**

In this work, we introduce P4STA, an open source framework that combines the flexibility of software-based traffic load generation with the accuracy of hardware packet timestamping. Our evaluation results, obtained using an off-the-shelf P4-programmable switch, show that a time resolution up to $1ns$ can be achieved on these programmable data plane platforms. Moreover we 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. Experiments on further programmable platforms, specifically on P4-SmartNICs and FPGAs, show similar results. With this work, we make P4STA available for the research community to advance high performance experiment measurements at nanosecond accuracy.

***Index Terms*—Performance Measurement, Latency, Throughput, P4, Timestamping, Nanosecond Precision**

I. Introduction

Benchmarking network elements such as switches, routers and softwarized virtual network functions is crucial for network management and system optimization in data centers, carrier networks and Internet exchange points. Benchmarking tools need to cope with stricter requirements in terms of ever-increasing workloads and lower latencies. For example, state-of-the-art data centers process millions of requests per second that are forwarded and load balanced by the network [20] where high latency and packet loss significantly impact the overall performance especially in the tail [6].

The total latency and packet loss of a data flow accumulate over network elements on the flow path. In order to identify the causes of performance bottlenecks, such as microbursts originating from web or cache traffic [21], along a path in a data center, it is necessary to investigate individual network elements. Bear in mind that current high performance network switches have a forwarding delay below $1\mu s$ [17] and a minuscule packet loss probability. These very low latencies and short microburst lengths that are in the order of a few microseconds require benchmarking tools at nanosecond precision not only to accurately capture them but also to capture their timing difference, hence their correlation. Note that such a precision is imperative when measuring Time-Sensitive Networks (TSN) where the jitter is required to be in the nanosecond range [16], as well as when measuring the packet duplication and multicasting behavior. In addition to the timing precision, exact packet counting is required at very small packet loss probabilities as prescribed by high reliability applications, e.g. URLLC in 5G. Both requirements, *nanosecond timing precision and exact packet counting,* make a software-based measurement impossible as their inaccuracy is magnitudes higher than the phenomenon to be investigated.

To put the previous comparison into perspective, the standard deviation for timestamps taken with tcpdump at $100Mbit/s$ load is around $100\mu s$ and the missed packet rate is roughly 0.1% [2]. This problem becomes harder if this high time accuracy and loss detection is required under high loads, e.g. up to $100Gbit/s$. Consequently, we note that loss detection and precise time measurements require hardware assistance. While some specialized, commercial solutions are available, these tools are inflexible in operation for customized scenarios and protocols; in addition to usually being very expensive [1], [4].

Currently, hardware-supported time measurement is based either on custom-built hardware for packet timestamping (e.g. Spirent, IXIA or DAG-cards), or on powerful network interface cards, which assist a software load generator by stamping packets before sending and after receiving [4] [18]. However, we note that there is a lack of flexible and programmable measurement assistance for timestamping and loss detection in laboratory environments. For example, the evaluation of the newly proposed L4S capable routers in high speed networks of 100 Gbit/s requires the generation of TCP-Prague traffic [12] in addition to normal traffic at this high network speed. This is easiest accomplished by load aggregation over many common Linux servers [12]. In the mean time, timestamping and loss detection with very high precision is required to validate the behavior of the investigated router. With P4STA we address this dependency using off-the-shelf network hardware and software-based load generators. By disaggregating load generation, timestamping and measurement, we created an open source framework which is neither limited to specific timestamping hardware nor to specific load generators to achieve high accuracies of time measurements and loss detection.

The main contributions of this paper are:

- A concept and framework for disaggregating load generation, timestamping, results capturing and test management with commodity hardware,
- A detailed analysis of the timestamping performance of a state-of-the-art P4 hardware switch and comparable test devices,
- P4STA, an open source framework for packet timestamping and load aggregation that is available on Github.

First, in Section II, we provide an overview of the related work and introduce the relevant terminology, as well as, technical challenges. We introduce the system design in Section III, describing all main components of P4STA. Our evaluation results are discussed in Section IV where we demonstrate the correctness and accuracy of P4STA. Finally, we provide a conclusion in Section V.

II. Background and Related Work

In the following we give an overview of technologies and terminologies, on which P4STA is based on before discussing the related work.

A. Background

One enabler of P4STA is the open programming language P4 [3], introduced in 2014, which permits the description of the behavior of packet header processing pipelines in a flexible way. This language allows the definition of custom header formats, header field manipulation, forwarding as well as dropping of packets. In a P4-packet processing chip, as illustrated in the sequel in Figure 2, the ingress and egress pipeline can be programmed with P4 and the behavior of buffers and Media Access Controllers (MACs) can be affected by setting particular flags in these P4-pipeline segments. For P4STA this language enables the easy capturing of timestamps and storing them in registers or in any parsed or added header field. In addition, special forwarding rules which are only typical for testbed setups, can be realized with ease. Many massive experiment design and execution frameworks, e.g. *MACI* [5], and testing frameworks as *Robot* focus on the initialization, execution and result management of repeated and parallel executed experiments. Conceptually, P4STA is complementary to such frameworks as it may be integrated either through its CLI or Python-RPC Interface.

B. Related Work

Software-based load generators such as MoonGen [4], TREX, and WARP17 [10] enable testing networking equipment. DPDK-based approaches support a timestamp granularity of $100ns$ in software and up to $10ns$ in hardware as discussed by Primorac *et al.* [18]. The authors state that commercial test tools are too expensive and that Application Specific Integrated Circuit (ASIC)-based hardware solutions are too inflexible. We go around this issue in the P4STA approach by using P4-programmable ASICs, NPUs and FPGAs, sold in higher volumes.
Micheel *et al.* [13] introduce Data Acquisition and Generation (DAG) capturing network interface cards that currently support bandwidths of up to $40Gbit/s$, as well as, timestamping with a $4ns$ granularity and time synchronization (IEEE-1588 and GPS) for distributed measurements.

The question of how to conduct latency measurements is discussed by Hernandez and Magana [7]. They conclude that one-way delay measurements are preferable over two-way, round-trip time (RTT) measurements and highlight the impact of latency on the throughput. Papagiannaki *et al.* [15] propose storing per-packet timestamps inside the sending and receiving DAG cards. They do so by matching packets with hash values of their IP headers. While this approach is seemingly promising, it is limited at a time resolution of 1 second latency for an OC-12 (622 Mbit/s) link as the hash table has only 1 million entries. Assuming a higher bandwidth or smaller packet sizes shows that this approach is not suitable for many scenarios, even with special hardware. Besides that, Morton proposes in RFC 8172 [14] that virtual network functions should also be considered as blackboxes, which is a main principle of P4STA as well.

The Open Source Network Tester (OSNT) [1] addresses two main use cases: (1) load generation and (2) monitoring with high flexibility and low costs ($< 2000\$$). Building on FPGAs as hardware platform for load generation and timestamping, the authors found out that the timestamping granularity depends on the clock frequency; in their work it was $6.25ns$. One main bottleneck identified in this work is the PCIe path for flow duplication to the host which is limited by the provided PCIe speed of the used FPGA. On the one hand, we are convinced that high bandwidth packet sending over PCIe on FPGAs leads to many challenges and the use of standard NICs makes life much easier for user-space load generators. But on the other hand, FPGAs currently suffer from having only a few ports and PCIe extend the number of available network interfaces. In-band Network Telemetry proposed by Kim *et al.* [11] is commercially available and constitutes one approach to measure QoS metrics in operational networks based on similar technologies. However, this approach, focusing on performance monitoring in networks based on attributes like queue sizes and link utilization, does not address the goals of this work such as accurate timestamping and test load aggregation.

III. System Model and Design

With P4STA we propose an open source framework for high performance packet timestamping and load aggregation. In contrast to existing approaches, the P4STA model consists of different functional components which disaggregate the load generation and timestamping functionality with commodity hardware. The modular approach enables the system to work on a wide range of platforms.

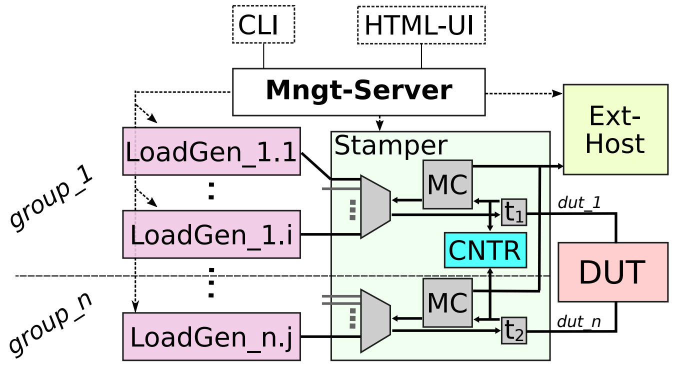

FIG. 1: Modular architecture of P4STA: n load generators, one hardware load aggregation and timestamping device and a Device Under Test in loopback.

As stated by the related work, hardware assistance is needed for high-accuracy timestamping while software flexibility is required for stateful network traffic generation. In this space, P4STA is conceived to combine the advantages of software and hardware on commodity packet processing platforms. For example, TCP congestion control and checksum validation is done with tested and powerful software, whereas time- and performance-critical actions are executed in P4-based hardware.

The overall system design is depicted in Figure 1. We describe each of its components in the following:

- **DUT:** The **D**evice **U**nder **T**est can and should be considered as a blackbox [14]. It is connected by at least one Ethernet cable to the Stamper component of P4STA. Typical are two links which are used to send and receive packets to/from the DUT but even more are possible as well, e.g. for a 3-port load balancing DUT. Additional physical or logical connections are not needed, however a initial configuration of the DUT, depending on its functionality, might be required.
- **Load Generator(s):** Typically, x86-servers are used as load generators. However, any load generator creating and sending Ethernet packets can be used as well. These generators are divided into n groups (typically $n = 2$) which is always equal to the number DUT-ports. Typical link speeds between these load generators and the Stamper component are $10Gbit/s$ and above. Supported load generators can be started automatically by the management server, as depicted in Figure 1, over the management network (dotted lines) using *ssh*. After test execution, load generator statistics and outputs can be gathered automatically as well. Both, stateless and bidirectional stateful traffic are supported by this system design. A typical packet flow of a TCP session between *LoadGen_1.1* and *LoadGen_n.1* sends packets from the first load generator to the Stamper, where the first timestamp (t_1) is taken. From there, the packets are forwarded to the DUT, traversing an unknown DUT-behavior. Coming back from the DUT, the packets enter the Stamper again, where the second timestamp (t_1) is taken and the packets are forwarded to *LoadGen_n.1*.
- **Stamper:** The central component of P4STA is the Stamper. First, it aggregates the flows of multiple load generators in order to create higher per-port loads. Second, every packet to and from the DUT can be stamped with the current time. Third, packets from the DUT can be duplicated (**Multi**Cast) to further increase the traffic load and to send packet duplicates, including both timestamps, to the external measurement host (*Ext-Host*). Last, a subset of the statistical data can be measured and stored inside the Stamper by using hardware counters/registers (*CNTR*), e.g. average delays and packet loss.
- **External Host:** All packets that go through the DUT and arrive at the Stamper for the second time can be duplicated and forwarded to the external host, extracting the timestamps for each received packet and creates a time series $(t_1, t_2)_p \forall p \in P$. After each test execution, this data is transmitted to the management server.
- **Management Server:** This server is the central controlling unit, also called P4STA-Core. It provides a CLI and web user interface for configuration, deployment, test execution and result analysis. By having access to all load generator servers, the Stamper and the external host, test preparation and execution can be initialized through this single node.

Note that these P4STA components, with except of the Stamper, can run on almost any server. For example, the external host and Stamper can be realized using the same device whereas the management server can also be realized on the same device as well. Thus, a minimum setup consists of a single server, which has a supported hardware for the Stamper (e.g. a NetFPGA or SmartNIC), and the DUT. In the following subsections we discuss some design decisions in detail.

A. Packet Switching in the Stamper Component

The load generators are separated in n groups which belong to the DUT-ports dut_1 to dut_n. All incoming packets from a load generator are forwarded to the corresponding DUT-port. However, forwarding packets, sent to and received from the DUT at the Stamper, is not that easy and can be realized in the following three ways which are all supported in P4STA:

- **Layer 1 forwarding:** This is the simplest and most robust forwarding mode which supports only 1 server per group. All forwarding decisions are made on the ingress ports only (layer 1 information). This is useful if none of the packet headers contains useful forwarding information. However, load aggregation can not be supported.
- **Layer 2 forwarding:** As Ethernet is currently the only supported link layer protocol, this mode behaves like a normal L2-switch for packets arriving at the ports `dut_1` and `dut_2`. Broadcast packets are forwarded to all load generators in this group.
- **Layer 3 forwarding:** Uses IPv4/IPv6 destination addresses instead of L2 addresses. This is useful if a

custom L2 protocol (or none) is in use containing no or insufficient destination addresses.

Additional forwarding modes can be added by the user if needed. However, with L1 forwarding almost every scenario is testable with at least reasonable performance.

B. Storing Timestamps

P4STA uses 48 bit for storing each time information, which is compatible with existing IEEE 1588 MAC-timestamping units. There are two general options to store per-packet time information:

- **Inside the packet:** Every timestamp is stored somewhere inside the packet, either in the payload or in a packet header. This approach is used by P4STA.
- **In the Stamper memory:** The packet is not modified and the timestamp is stored in the memory of the Stamper.

The first approach, storing timestamps inside the packets, has the disadvantage that either the packet size increases or existing bytes must be reused at a known position in the packet. Note, checksum updates might be required. In contrast, storing the timestamp outside the packet leaves the packet intact at the cost of memory consumption. If we assume a line rate of $100Gbit/s$, 64 byte long packets and a constant DUT latency of $10ms$ we find that about $2 \cdot 10^6$ packets are always in flight. Assuming no storage overhead, a 48 bit timestamp for each packet requires $12MB$ of memory for the in flight packets. As most eligible hardware, like programmable switches and FPGAs, have only a highly limited internal memory space (e.g. SRAM) of a few MBs due to technological reasons, storing timestamps in the hardware chip of the Stamper device is not advisable. Focusing only on devices (DUT) with a guaranteed latency below a few μs and storing the timestamps in the Stamper device might be possible but still leads to the following challenges:

1) timestamps of lost packets must be deleted after a certain time as packet loss may occur,
2) for each packet two timestamps must be sent out immediately after taking the second timestamp (t_2), as there is only enough memory for the in flight packets,
3) packets must be mapped to the timestamp taken at the first Stamper pass based on distinct hashes or sequence numbers and
4) internal memory structures must allow an address-based memory access for read and write operations concurrently.

Reducing P4STA to devices with external, low-speed memory (e.g. DRAM) is not an option, because this memory option is often not available and would significantly reduce the number of compatible devices. However, the bandwidth of these external memory technologies might be sufficient as only timestamps must be stored and not all packets. Thus, we finally decide to store both timestamps, taken before and after the DUT, inside the packet.

While IPv4 and IPv6 options often lead to packet drops in routers due to security policies, a storage location in a layer 4 protocol or higher is more appropriate for most use cases. Timestamps can be either stored in a packet header option field or inside the payload, which is both supported by P4STA. We choose an additional TCP options header as primary timestamp position, added by the Stamper to the packet. However, UDP packet timestamping is supported as well. As UDP does not support optional header fields, the timestamps will be stored in the beginning of the payload, presuming that the payload is not meaningful used. Furthermore, parsing these timestamps is fault-prone as no header field indicates a subsequent timestamp.

For every unstamped packet the Stamper allocates space for both timestamps (before and after DUT) as TCP option field in the header stack. Thereby, it guarantees constant packet sizes when the second timestamp is written after a packet traversed the DUT. UDP does not support option fields but a timestamp storage in the beginning of the UDP payload is possible as well, but only if the payload is not used by the load generator in a meaningful way. Figure 3 depicts the integration of two 48 bit timestamps as a TCP option. We use the option type `0x0f` (alternate checksum), as it is obsolete and should be ignored by all hosts and middleboxes. In addition, updates of the IP/TCP/UDP checksums are required. As the old checksum of the header is known and a one's complement checksum is used, the new checksum is simply computed based on the old checksum and the applied packet modifications in the Stamper.

One detail to account for is the packet size: When adding the additional header, e.g., 20 bytes for the TCP option, the increased size might exceed the maximum transmission unit (MTU) of DUT or load generators. In case of UDP payload timestamping, the packet size stays constant.

C. High Precision Timestamping

The next important question to answer concerns where the timestamp should be retrieved in the pipeline. In general, the first timestamp, i.e., prior to the DUT, should be taken as late as possible while the second timestamp, after returning from the DUT, should be taken as early as possible. Figure 2 depicts the $P4_{14}$ reference pipeline with additional ingress and egress ports and Media Access Controllers (MAC). Although this pipeline model is slightly specific, its principles are very similar to non P4-pipeline ASICs such as FPGAs and SmartNICs.

Depending on the capabilities of the hardware, an ingress timestamp might be provided either by the ingress MAC, the parser or the programmable match-action pipeline, where in general the first one should be used. For example, the P4-NetFPGA project uses shared resources in the packet parser. Following from this, micro congestion may occur between MAC and packet parser which can lead to measurement inaccuracy in the range of multiple *ns*. This MAC-timestamp can be provided by the programmable pipeline as a metadata value. Thus, it can be stored later somewhere in the programmable pipeline.

Taking the first timestamp and counting packets must be done after dequeueing the packets from the buffer of the

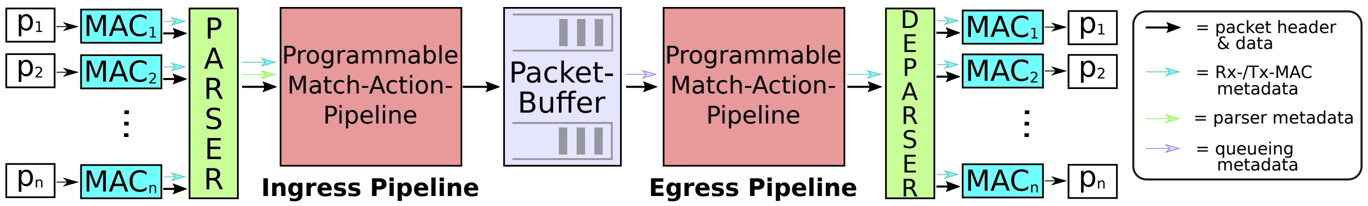

FIG. 2: Timestamping capabilities in a P4-like packet processing piepline chip.

0–7	8–15	16–31
...		
Checksum		Urgent pointer
Option=0x0f	Length=0x10	Timestamp1 [47:32]
Timestamp1 [31:0]		
unused(16)		Timestamp2 [47:32]
Timestamp2 [31:0]		
... [Payload] ...		

FIG. 3: Timestamp storage in TCP option fields.

Stamper device as this queueing can create a random added delay which would falsify the measurement results. The dequeue timestamp of a packet, provided by the packet buffer as metadata, can be added by the egress pipeline to the packet header. However, if we assume a variable processing time of the deparser, e.g. due to a shared parser for all ports, this still might lead to inaccuracy as well. Note that many target hardware, like Barefoot Tofino, NetFPGA Sume and Netronome SmartNICs, support inserting timestamps at the transmit MAC in accordance with the IEEE 1588 standard, which can be used for P4STA as well. Instead of writing the timestamp directly into the packet, an additional metadata value, called *Tx-MAC*, is provided to the MAC together with the packet. This metadata value contains the positions inside the packet where the timestamp should be inserted, as well as, where the TCP/UDP checksum is located in order to update it. Even if low level FPGA-designs are used without a P4 high level synthesis, the process of inserting an IEEE 1588 timestamp in the transmit MAC is very similar. This standard, motivated with high accuracy time synchronization, leads to many devices on the market with hardware timestamping capabilities.

Besides programmable data planes, common NICs should be considered: On the one hand, the use of P4 programmable SmartNICs allows a similar timestamp insertion to P4 programmable switches. On the other hand, current chipsets of NICs, e.g. Intel X520 or X710, do not support hardware timestamping with default kernel modules and thus the use of DPDK is required [8], [9]. Therefore, many load generator rely on DPDK, e.g. MoonGen [4], which creates a dependency between the load generator and the timestamping functionality. Furthermore, we note that for precise timestamping results, a dedicated queue on the NIC is required for the packets to be timestamped. This in turn leads to the fact that not all packets can be stamped and not all metrics can be measured in hardware. Besides that, load aggregation can only be performed with limitations, e.g. shard PCIe bus, by different load generation threads on the same machine. In summary, state-of-the-art, DPDK-based load generation and timestamping can be integrated within P4STA but, due to flexibility limitations, they are currently only supported as load generator. Currently, more flexible Stamper devices like (P4-)SmartNICs, FPGAs or programmable P4 switches are more suitable.

D. *Stateful Stamper Operations*

While storing the timestamp data inside the packet header, the following information can and will be captured and stored inside the Stamper:

- **Timestamped Packet Count:** The total number of timestamped packets and bytes before and after the DUT which enables, e.g., the calculation of packet loss.
- **Send and Received Packet Count:** The number of packets and bytes sent and received per physical port. This indicates the load distribution between multiple load generators. In contrast to timestamped packets, this may also contain noisy traffic like neighbor discovery messages.
- **Total DUT Delay:** The sum of all measured delays for the DUT in ns. Based on that and the number of stamped packets, the average delay can be derived. Note that the interpretation of these values requires a thorough understanding of the measurement setup (e.g. busy period length, delay correlations, ...)
- **Minimum/Maximum Delay:** As in some cases not all packets can be forwarded to the external host, a minimum/maximum delay can be recorded in hardware.

The counters for the total number of stamped packets and bytes as well as the *total DUT delay* are implemented as $64bit$ counters in order to avoid overflows whereas a 48 bit register is sufficient for the minimum and maximum delay. All these values are read out after test execution automatically by the P4STA management server and are available for analysis.

E. *Load Generation, Aggregation and Shaping*

In general, any load generator can be used with P4STA as long as the timestamp format (header or payload) matches the Stamper implementation. Nevertheless, we assume that mainly software based load generators are used as they are inexpensive and flexible. Currently we integrate iPerf3 and MoonGen as load generators which are Linux kernel socket and DPDK based, respectively. Note that MoonGen also supports hardware timestamping within limitations of the used NIC, as discussed before. The integration of a generic load generator control interface in P4STA allows to simply transmit traffic and collect result at all load generators sources and sinks. A custom load generator can either be integrated into P4STA or started from outside.

Flows from multiple load generators can be aggregated, limited by the number of ports of the Stamper device. This is based on Layer 2/3 forwarding as described in Section III-A. Hence, using, e.g., 10 servers each with $10Gbit/s$ link speed, load can be aggregated on a single $100Gbit/s$ link. This is useful as current software load generators are not able to generate such a high traffic load at small packet sizes.

If the Stamper device has packet queues, as assumed in the generic $P4_{14}$ pipeline, these can be used to shape the data rate towards the DUT. Shaping is useful if the DUT should be tested with specific traffic patterns, e.g., under a load sweep, or if the maximum supported bandwidth by the DUT is limited to a value lower than the link bandwidth, e.g. due to added headers in the network function. This shaping can guarantee a constant load towards the DUT as long as the arriving rate of the load generators is higher than the shaped rate and the rate control at the load generators is packet loss agnostic.

F. Data Capturing

Timestamps are captured at the external host through seeking the predefined signature, e.g. the introduced TCP option `0x0f`. Note that further information such as packet reordering can easily be detected here as well. Additionally, tracing and storing all packets, including the timestamps, for later evaluation is possible. In P4STA two implementations of the external host are provided: (1) one Python based raw socket implementation which is able to capture packets up to a bandwidth of $1Gbit/s$. While this implementation is supported by any target with a network interface port, it can not capture packets at line rate of the DUT, which may be up to $100Gbit/s$. Hence, the Stamper can be configured to duplicate only every n_{th} packet towards the external host. In contrast, (2) a high performance DPDK version of the external host can be used instead, however this requires a DPDK compatible NIC.

G. Testbed Setup, Results and Reproducibility

Installing P4STA depends on the used target for the Stamper, load generators and external host. We provide installation scripts which configure all servers for kernel based external host implementation and `iPerf3` as load generator.

The workflow of P4STA is divided into four steps: (1) Testbed configuration, (2) configuration deployment to all involved devices, (3) test execution and (4) analysis of the results. In the first step, all ports, load generators and link bandwidths are defined by the user interface or configuration file. In the second step, this configuration is converted considering the selected Stamper target and is deployed to it. Third, either an integrated load generator can be started in the UI or any other load can be initiated manually and all results are

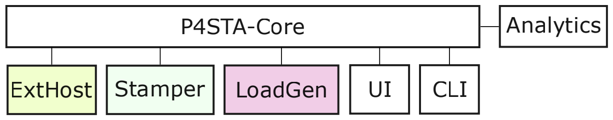

FIG. 4: Modular Software Components of P4STA. Modules can be replaced depending on the hardware to be used.

retrieved and permanently stored by the management server. Last, with P4STA Analytics we provide a tool-chain to evaluate the measured results and provide graphs, measured values and computed statistics. These graphs, including raw data, computed statistics and all required scripts for customization and recreations of the plots can be downloaded as zip archive file. Additionally, the testbed setup can be reconstructed with this download later, e.g. for reproducing research results.

H. Extendibility and Setup

The P4STA management component is implemented as a Python-based web server and core logic, which is divided into several parts of different functionality as depicted in Figure 4. While for the UI and CLI only one implementation is provided, the implementation and interfaces for the external host, Stamper and load generators can be extended for other targets or extended functionality. For example, the installation of the target *Barefoot Tofino* can be achieved by copying the corresponding sub-repository into the `StamperTargets` folder. The process of integrating further external host or load generator implementations is similar. As the source code for all modules is available, custom features can be integrated easily. For example, stamping of packets with a special header structure can be integrated by only updating the data plane description of the Stamper. For illustration, we tested a tunneling network function where the timestamp was written into the TCP options header inside the encapsulation payload of the packet without changes except for the P4 data plane program.

IV. EVALUATION

In this chapter we pursue two main goals: (1) show the correctness and accuracy of P4STA with a P4 programmable switch and (2) provide exemplary results for different network devices that illustrate the abilities of this framework.

A. Packet Timestamping with Nanosecond Precision

In this section, we will measure the accuracy of P4STA timestamping and packet loss detection with a programmable P4 switch based on the Barefoot Tofino ASIC (1st generation). To show the accuracy of the measurement system we conduct measurements of the latency of fiber-optic cables of known lengths. The used Stamper implementation is optimized for this target and uses IEEE-1588 timestamping capabilities of the ingress and egress MACs as illustrated in Figure 2.

We performed multiple tests with *MTP OM4* fibers, connected to two ports of the timestamper. We assume that the latency l depends on a constant offset t_c, caused by MAC, transceivers and other (unknown) functional units, and the length of the fiber len_f, i.e., $l = t_c + len_f \cdot c_{\text{fiber}}$ where c_{fiber} is

	actual length	avg. latency	std. dev.	packet loss
1m	1,06 m	107,830 ns	1,46 ns	0
2m	2,08 m	112,850 ns	1,61 ns	0
3m	3,18 m	118,336 ns	1,52 ns	0
10m	10,12 m	152,883 ns	1,61 ns	0

TABLE I: Measured latency and standard deviation for MTP OM4 multimode fibers with different lengths.

the speed of light in the optical fiber. The measurements were conducted as follows: First, we measure the latency for MTP OM4 multimode fibers with varying lengths in the range of $1m$ to $10m$. Each test was conducted for 10s, resulting in about 8 million measured packets of 1514 bytes for each run. All test runs are performed with the same transceivers and 10 Gbit/s load in 40GBase-SR4 link mode and MAC-timestamping is enabled. The resulting values are listed in Table I.

Using a linear regression, we estimate the values $t_c = 102.52ns$ and $c_{\text{fiber}} = 4.97ns/m$. The average absolute residuum is $0.018ns$ and the maximum error is $0.03ns$ (1m Fiber) with a minuscule confidence interval for multiple test runs. Note that the speed of light in an optical cable depends on the refractive index of the fiber and can be assumed to be between 67% and 69% of the speed of light in vacuum leading to a range of $4.83ns/m - 4.98ns/m$ [18], [19], which is consistent with our measurements. The value t_c can be computed once and subtracted from further measured results in order to obtain the DUT latency. While the average latency has a sub-nanosecond accuracy, the standard deviation is $1.6ns$ and the latency range is $12ns$ (delay$_{avg} \pm 6ns$). Thus, the delay of a single packet can only be measured with an error of $\pm 6ns$ but average delays have a (sub)nanosecond precision. The packet loss, computed by the counter values of all sent and received packets, is zero for all tests. As one measurement series consists of many measurements, we consider these results reproducible with high time accuracy and detectable packet loss as small as 1 per 1 million packets.

Recall that results of timestamping packets in the ingress and egress pipeline instead of in the MACs can be slightly worse. However, we observed results which are, with respect to a higher latency offset, very similar in time accuracy. For FPGAs a timestamp granularity of $1ns$ is not possible as their clock frequency is in the range of 100MHz to 350MHz and a single clock cycle takes $10ns - 3ns$. First experiments with FPGAs have shown an ingress timestamp accuracy within $20ns$ without using MAC timestamp capabilities.

B. Network Elements Evaluation

In this section we show and compare results of P4STA consisting of a P4 programmable Barefoot Tofino switch and Moongen as load generator. We use constant bit rate input without bursts and packet sizes of 1020 bytes (including timestamp) for measuring the following network elements with a link speed of $10Gbit/s$:

- **NetFPGA SUME** bridging two ports with a store-and-forward FIFO realized with internal memory.

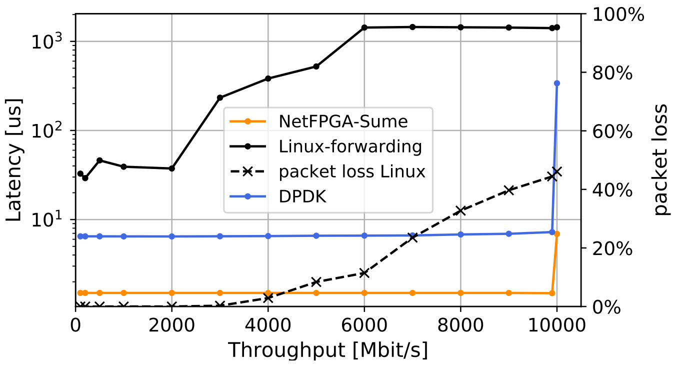

FIG. 5: Average latency of NetFPGA-Sume, Linux-server, DPDK forwarding (Intel 82599) and loss of Linux. The loss of all other DUT is always zero, except of the NetFPGA-Sume with 10 Gbit/s load (0.003% loss). Packet size: 1020 byte.

- **DPDK packet forwarding** based on the DPDK *basicfwd.c* sample application on a Dell PowerEdge R740 (2x Intel E5-2670v3, 128 GB) with an Intel 82599 NIC.
- **Linux kernel packet forwarding** based on L3 addresses and static configured routing tables.

We focus on selected measurement results for space reasons and to highlight the benefits of P4STA as illustrated in Table II for some selected metrics.

Figure 5 visualizes the average latency of the selected DUT dependent on the load. While the DPDK and the FPGA yield a constant latency, the forwarding delay of the Linux kernel starts increasing from a load of $2Gbit/s$ jumping from $40\mu s$ to $1.45ms$ in average. DPDK and NetFPGA only deviate from the constant latency when the load is identical to the link speed, i.e., in the step going from $9.9Gbit/s$ to $10Gbit/s$.

Figure 6a depicts the packet latency of the investigated DUT as a function of the packet input time. We observe that the latency of the NetFPGA shows a nonstationary behavior where it increases up to $7.4\mu s$ when increasing the load from $9.9Gbit/s$ to $10Gbit/s$. This latency corresponds to the size of the FIFO packet queue which connects the two ports, and shows a typical tail drop queue behavior which explains the packet loss in this setting. The figure shows that for an input rate of $9.9Gbit/s$ the packets can be forwarded without queueing. The figure also shows that the DPDK application shows a similar behavior as the FPGA for $10Gbit/s$. However, in DPDK the amount of buffered packets may increase during the test period up to a delay of $733\mu s$ as the memory size is not limited. Consequently, no packets are lost in this setting.

Figure 6b shows the tail latency for the DUT given different loads. Note the extreme jumps for the Linux kernel packet forwarding and ceiling due to memory allocation. Similarly, Figure 6c depicts how the Linux kernel packet forwarding behaves for 1020 byte packets under a load of $5Gbit/s$, that can be regarded the edge case between functional correctness and overload, and the overload case of $7Gbit/s$. The depicted result conforms with the values of the average absolute IPDV and standard deviation reported in Table II. Plots like this, measured and created with P4STA, make the evaluation and

	NetFPGA(9.9G)	NetFPGA(10G)	DPDK(9.9G)	DPDK(10G)	Linux(2G)	Linux(5G)	Linux(7G)
transmitted packets	11887798	12005428	11886608	12007588	2405708	6012055	8412592
lost packets	0	340	0	0	0	503711	1979288
loss	0%	0.003%	0%	0%	0%	8.38%	23.5%
average delay	1.49μs	6.89μs	7.23μs	338.89μs	37,42μs	522μs	1450μs
min delay	1.43μs	1.45μs	6.10μs	6.05μs	8.87μs	7.47μs	37.18μs
max delay	1.57μs	7.43μs	23.36μs	733.22μs	256.78μs	3220μs	3210μs
std. delay deviation	28.9ns	664ns	427.9ns	220630ns	17230ns	917400ns	98620ns

TABLE II: Measured results with P4STA and a P4 programmable switch for different devices under test and bandwidths. Rate Limiting was performed inside the Stamper. Average delay and loss was measured with registers inside the Stamper device, all other values are captured by the external host for every 100. packet. Packet size: 1020 byte. No offset-correction.

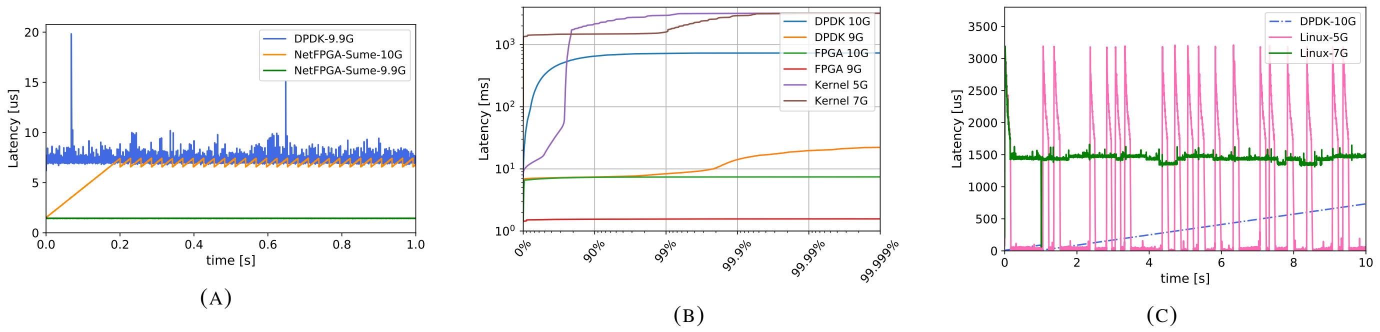

FIG. 6: (A) Latency time series of individual packets at NetFPGA-Sume for $9.9Gbit/s$ and $10Gbit/s$ load and a Linux server with DPDK packet forwarding under $9.9Gbit/s$ load. Packet size: 1020 byte. (B) Tail latency percentile of the investigated DUT. (C) Latency time series of individual packets at a server with DPDK capable NIC. Showing results for Linux kernel with $5Gbit/s$ and $7Gbit/s$ load and DPDK with $10Gbit/s$ for comparison. Packet size: 1020 byte.

understanding of network elements, e.g. FPGA based network functions with a latency of hundreds of ns, much easier.

C. Resource Utilization and Limitations

The current design of the Stamper for P4 targets has a dependency graph of 6 pipeline stages for the ingress and 1 for the egress pipeline. In case of Barefoot Tofino, this number of stages is needed as well. However, the resources of each stage have only a low utilization and further logic can be combined with P4STA on a Tofino architecture. Furthermore, if some functionality, e.g. L2/3 forwarding, is not needed, it can be removed from the code to reduce the resource utilization further. We made the observation that rate limiting and packet duplication are non-trivial for FPGAs and programmable switches as this influences the timing of the total pipeline and can affect the accuracy of timestamps in some cases.

D. Costs and Required Hardware

The P4STA framework can be used with multiple hardware timestamping devices that are not particularly built for this primary use case and can be used by research institutes for many other applications as well. In addition to the reusability of the hardware, the costs connected to P4STA are lower as such common devices are sold in higher volumes. In contrast a commercial load generator may cost above $100k$\$ if high loads and special protocols are required. Programmable SmartNICs are available for a few hundreds of dollars, high-performance network-FPGAs are available below 1500\$ and even programmable white-box switches are available for 7500\$[1].

V. Conclusion

In this paper we introduced a concept and a corresponding open source framework, denoted P4STA, for disaggregated load generation and packet timestamping at line rate with nanosecond accuracy. P4STA is designed for off-the-shelf programmable network devices, such as FPGAs, SmartNICs and P4 switches. We demonstrated its high accuracy in measuring latency in the range of nanoseconds in addition to highly accurate loss detection and load aggregation capabilities. Furthermore, we proposed an approach for in-packet timestamp-storage, shown on the example of TCP options. We note that in addition to being flexible, the costs of a P4 programmable SmartNIC or switch are much lower than for a special purpose built load generator. The P4STA framework is available as open source[2]. Currently supported targets are the reference software switch bmv2, Barefoot Tofino and Netronome SmartNICs. A reference implementation for the NetFPGA will follow soon. In future work we plan to support further hardware targets, as well as, the investigation of distributed measurements with IEEE1588 time synchronized Stamper devices and advanced evaluation tools.

Acknowledgment

This work has been supported by Deutsche Telekom through the Dynamic Networks 8 project, and by the German Research Foundation (DFG) as part of the projects C2,B4 within the Collaborative Research Center (CRC) 1053 MAKI and in parts by the DFG grant SPINE. Furthermore, we thank our colleagues and reviewers for their valuable feedback.

[1]Web-price for Aurora 710, July 2019

[2]https://github.com/ralfkundel/P4STA

References

[1] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Mckeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski, "Osnt: open source network tester," vol. 28, no. 5, Sep. 2014, pp. 6–12.

[2] P. Arlos and M. Fiedler, "A comparison of measurement accuracy for dag, tcpdump and windump," ResearchGate, 2016.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[4] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. ACM, 2015, pp. 275–287.

[5] A. Frömmgen, D. Stohr, B. Koldehofe, and A. Rizk, "Don't repeat yourself: seamless execution and analysis of extensive network experiments," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. ACM, 2018, pp. 20–26.

[6] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, pp. 225–238.

[7] A. Hernandez and E. Magana, "One-way delay measurement and characterization," in *International Conference on Networking and Services (ICNS'07)*. IEEE, 2007.

[8] Intel. (2014) Data plane development kit. [Online]. Available: https://www.dpdk.org/

[9] ——. (2018) Does intel xl710-qda2 support hw timestamping. [Online]. Available: https://forums.intel.com/s/question/0D50P000049xNZ1SAM/does-intel-xl710qda2-support-hw-timestamping?language=de

[10] Juniper. (2016) Warp17: The stateful traffic generator. [Online]. Available: https://github.com/Juniper/warp17

[11] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker. (2015) In-band network telemetry via programmable dataplanes. [Online]. Available: https://github.com/p4lang/papers/tree/master/int-demo

[12] J. Luo, J. Jin, and F. Shan, "Standardization of low-latency tcp with explicit congestion notification: A survey," *IEEE Internet Computing*, vol. 21, no. 1, pp. 48–55, 2017.

[13] J. Micheel, S. Donnelly, and I. Graham, "Precision timestamping of network packets," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, ser. IMW '01. ACM, 2001, pp. 273–277.

[14] A. Morton, "Considerations for benchmarking virtual network functions and their infrastructure," Internet Engineering Task Force, Request for Comments 8172, 2017.

[15] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, "Measurement and analysis of single-hop delay on an ip backbone network," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, pp. 908–921, 2003.

[16] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open source opc ua pubsub over tsn for realtime industrial communication," in *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2018, pp. 1087–1090.

[17] I. Plexxi. (2016) Latency in ethernet switches. [Online]. Available: http://www.plexxi.com/wp-content/uploads/2016/01/Latency-in-Ethernet-Switches.pdf

[18] M. Primorac, E. Bugnion, and K. Argyraki, "How to measure the killer microsecond," in *Proceedings of the Workshop on Kernel-Bypass Networks*, ser. KBNets '17. ACM, 2017, pp. 37–42.

[19] Quora. (2015) What is precisely the speed of light in fiber optics? [Online]. Available: https://www.quora.com/What-is-precisely-the-speed-of-light-in-fiber-optics

[20] A. F. Voellm. (2013) Compute engine load balancing hits 1 million requests per second! [Online]. Available: https://cloudplatform.googleblog.com/2013/11/compute-engine-load-balancing-hits-1-million-requests-per-second.html

[21] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proceedings of the Internet Measurement Conference*, 2017, pp. 78–85.