Network Testing Utilizing Programmable Network Hardware

Kundel, Ralf; Siegmund, Fridolin; Hark, Rhaban; Rizk, Amr; Koldehofe, Boris

Published in:
IEEE Communications Magazine

DOI:
10.1109/MCOM.001.2100191

Publication date:
2022

Citation for published version (APA):
Kundel, R., Siegmund, F., Hark, R., Rizk, A., & Koldehofe, B. (2022). Network Testing Utilizing Programmable Network Hardware. *IEEE Communications Magazine*, *60*(2), 12-17. https://doi.org/10.1109/MCOM.001.2100191

Ralf Kundel, Fridolin Siegmund, Rhaban Hark, Amr Rizk, Boris Koldehofe. *Network Testing Utilizing Programmable Networking Hardware.* Accepted for publication in IEEE Communications Magazine, Network Softwarization and Management Series, presumably date of publication: 01/2022.

Network Testing Utilizing Programmable Network Hardware

Ralf Kundel, Fridolin Siegmund, Rhaban Hark, Amr Rizk, and Boris Koldehofe

***Abstract*—QoS requirements on modern network hardware, including switches and routers, require the ability to conduct precise measurements of the packet processing and forwarding of network elements. This requires tracing packet processing and detecting the loss of packets with high timing accuracy. Current approaches for network testing rely on special and purpose-built devices, which are costly and inflexible as these devices *cannot* be reconfigured to include new testing or monitoring functionality.**

In this article, we demonstrate the power behind novel programmable network switches to enable highly accurate and flexible testing and monitoring of network element functionality before and during deployment. While the cost of such switches is comparable to traditional commodity switches, their processing logic can be programmed to realize specific networking functionality. In the context of P4STA, an open source measurement framework previously presented by us, we show how the programmability of modern network switches helps to perform highly accurate and purpose-independent testing of network elements. In addition, we also highlight its ability to support reconfigurable monitoring tasks within the network after deployment.

I. Introduction

In the last decades, communication networks have become the backbone of nearly every digital service used in daily life. Networking hardware has experienced tremendous performance jumps caused by the Internet becoming a gigantic network with a total bandwidth of hundreds of terabits per second. The underlying networks that constitute the building blocks of this ecosystem (such as Internet Service Providers (ISPs), enterprise/residential networks and data centers) face steadily increasing and highly intertwined QoS/QoE requirements for very different classes of applications such as Voice over IP or automation. As a result, modern network elements must provide a data plane performance of multiple terabits per second while having forwarding delays of less than a microsecond and no unexpected packet loss.

Today's network elements typically constitute switches, routers but also complex network functions. Some network functions can be subject to regulatory requirements, which need to be thoroughly tested. For instance, a broadband network gateway in Telco access networks must fulfill a certain QoS level regarding packet loss and counting accuracy [1]. In this work, we will focus on the example of switches in 5G fronthaul networks as Device Under Test (DUT), connecting radio units with base-band units. For that, a very deterministic forwarding behavior and no packet loss are required to ensure a failure-free operation of the 5G network. In the aforementioned example of 5G fronthauls, *e.g.*, unfavorable queue management or incorrect prioritization can lead to packets not being delivered within the required timeframe or even losses and as a consequence to failures in the 5G network [2].

Meeting requirements of high-performance networking hardware typically requires a sophisticated understanding of the hardware ahead of its deployment. Analyzing a few samples at low resolution will in many cases not allow to find the causes. Instead, measurements must be sampled with a high resolution.

A. Background and Technological Fundamentals

To understand the importance of high resolution testing let us take a closer look at the four phases in the innovation cycle of networking hardware. Functional and performance testing is conducted in four phases of the innovation cycle of networking hardware as depicted in Figure 1. In the research and development phase (1), it is required to measure and understand existing networking hardware's behavior in detail and perform experiments with new prototypical hardware. Considering the previously mentioned performance requirements, these experiments must be performed with very high accuracy and at the same time at maximal load. In the subsequent quality management phase (2), the newly developed and produced products at the manufacturer require testing to validate the expected quality on a random sample of

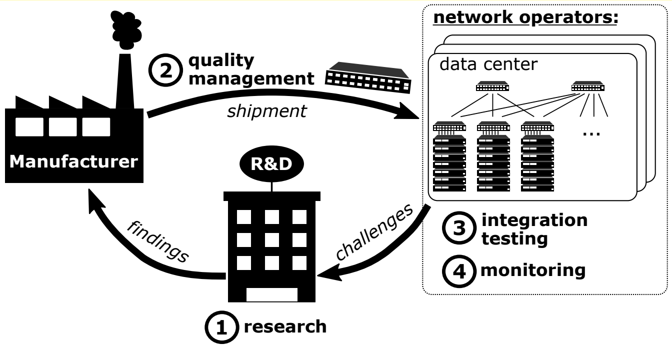


Fig. 1: Innovation cycle of networking hardware. Measurements of the hardware are required in all phases.

the production output. After shipping to the operator, typically a data center or Internet service provider, the hardware will be configured for deployment and the software updated. Again, the expected behavior for the *specific use case* must be validated (3). Finally, networks are monitored in operation in order to detect and correct abnormal behavior (4). Based on the monitoring observations, new challenges for R&D arise, which completes the innovation cycle. The obtained understanding of the networking hardware behavior is required by network operators, hardware vendors and researchers in order to improve current and future networks.

Concretely, the testing system must support a very high-throughput, *i.e.*, up to $100Gbit/s$, measuring latency with a few nanoseconds accuracy and detecting very small values of packet loss. Currently, it is necessary to use specialized tools for network testing to meet these requirements simultaneously. However, these commercial and closed-source tools require purpose-built hardware that is limited in flexibility and incurs high costs. This causes either a high effort and costs or leads to insufficient testing.

Paradigms for programming networks have shaped the development of communication networks in the last decade [3]. This programmability and by that the gained flexibility can be divided into two key aspects enforced by programmable hardware: The opening of the control plane interface, which is responsible for controlling the flow rules in the switch data plane, was the defining first step. One major milestone for this was the introduction of the OpenFlow protocol, which allowed opening up existing networking hardware supplied by major vendors. As control plane protocols do not allow to modify the fundamental behavior of data plane chips, reconfigurable hardware was a second major step towards network programmability [4]. While Field Programmable Gate Arrays (FPGAs) allow reconfiguring digital circuits on bit level since the 80s, they suffer from comparably low clock frequencies. Hence, they cannot provide the same performance as special-purpose chips (ASICs) for networking hardware. However, the introduction of programmable hardware specialized for networking purposes enabled reconfiguration with a performance comparable to that of non-configurable chips used in networking applications [5]. By that, many different applications, including but not limited to network measurements, can be realized on the same hardware platform. One primary driver of this technology is the programming language P4 [6], which provides a simple and powerful abstraction for data plane programming.

Modern networking hardware that is in operation today already provides such programmability and allows, together with software, to create flexible and high-performance test solutions that tackle the previously mentioned drawbacks of existing purpose-built approaches. Consequently, these off-the-shelf devices will constitute a fundamental building block for functional and performance network testing.

In our previous work, we initially presented P4STA [7], a load generation framework, open source available on GitHub [8]. One main advantage of open source tools over commercial products is their easy customizability and, consequently, increased flexibility.

The contributions of this article are: 1) we generalize the approach of load generation with programmable hardware and focus on the integration of *programmable networking hardware* and *open source software* in all phases of the innovation cycle; the presented evaluation results refer to scenarios belonging to the phases 1-3. 2) We point out how to create testing solution for networking hardware which provides *(i)* high testing performance and *(ii)* flexibility at *(iii)* comparably low costs. 3) Last, we present an approach for detecting packet reordering with sub-microsecond accuracy.

II. Utilizing Programmable Hardware for Network Testing

In the following, we describe how programmable hardware can be used for flexible and high-performance network measurements. The subsequently discussed approach can be used in all four testing scenarios of Figure 1 by reconfiguring existing hardware.

The P4STA framework tackles the previously named challenges in a disaggregated manner:

A. Disaggregated Network Load Generation and Packet Timestamping

The P4STA setup for disaggregated network measurement experiments, as shown in Figure 2, consists of several components:

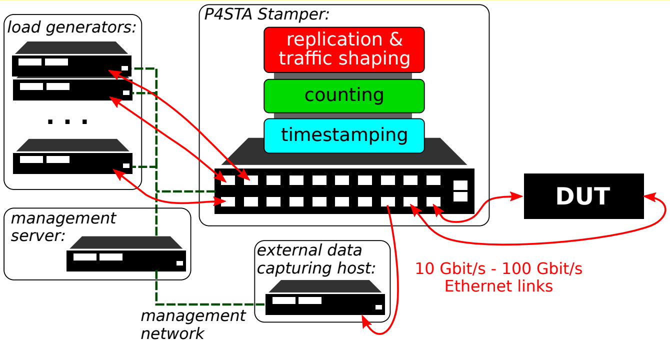

load generators:
P4STA Stamper:
replication & traffic shaping
counting
timestamping
DUT
management server:
external data capturing host:
10 Gbit/s - 100 Gbit/s Ethernet links
management network

Fig. 2: Measurement setup following the P4STA terminology relying on a P4-programmable Stamper.

DUT: The Device Under Test, typically the networking hardware to be measured, will be considered in the following as a black box. This means the internals of the DUT are not disclosed; however, the external behavior, *e.g.*, supported protocols or how packets are manipulated, are known in order to configure P4STA properly. We assume that the DUT provides $n \geq 1$ Ethernet ports, typically two, for receiving and sending packets.

Load generators: The load generators are standard servers with powerful network interface cards, as a software realization is very flexible and easy to implement.

In order to support challenging measurement scenarios, multiple load generators can be used in parallel. Hence, the entire state, computing demand and traffic load can be distributed. This allows each generator to increase its maximum rate and more sessions in parallel can be generated.

P4STA Stamper: The central component of P4STA is the Stamper. First, the Stamper is responsible for forwarding packets from the load generators to the DUT and vice versa. For that, the packet destination information, typically Ethernet or IP addresses, is used. Second, each packet to and from the DUT will be timestamped by the egress or ingress port, respectively. By taking the timestamps in hardware, a very high time accuracy compared to software-based approaches, having an inaccuracy of up to $1000ns$, can be achieved [9]. Third, the number of incoming and outgoing packets and bytes for each port will be counted in order to detect packet loss in hardware with very high accuracy.

Last, traffic replication and port shaping can be used for specifying the load on the DUT. Generating high loads in software is very challenging, especially with small packet sizes. For that, the replication feature can help to reach high packet rates.

External data capturing host: Modern networking switches cannot store large amounts of data within the data plane of the switch. Consequently, the data to be captured must be transmitted as a continuous stream to a sink where it can be stored. In P4STA, we define this sink as an external data capturing host connected via Ethernet to the Stamper. It receives copies of a subset or all packets returning from the DUT, duplicated by the Stamper. This copy can be either the entire packet or a header digest only. All metadata information, typically the timestamps of a packet, must be part of this copy.

Management server: Disaggregation always results in increased management effort and a possible lack of clarity for human administrators. In order to prevent this, all measurement processes are automated and are controlled by a centralized server. Typical operations such as deploying a configuration on the Stamper, starting a measurement or collecting the results can be simply controlled. This central application is accessible via an HTML-based user interface to facilitate usability and make measurement results visible. In addition, a CLI and API are provided for test case automation.

B. *Storing Measurement Data at Line Rate*

Network measurements with high load typically generate a tremendous amount of data to be stored. Assuming a rate of 10^8 packets per second and 16 bytes of measurement data to be stored for each packet results into $1600MB/s$. However, the internal memory of modern networking switches, which is composed of high-speed SRAM memory cells, only provides very little capacity in the range of hundreds of kilobytes. Hence, we note that not even the measurement data of the packets in-flight that are traversing the DUT can be stored in the data plane of the switch. This huge amount of data implies that monitoring in networks on all switches on a per-packet base can cause scalability issues. Indeed, this internal memory is very well suited for storing aggregated measurement data such as the total number of sent packets per port. In order to circumvent this memory limitation, this data can be stored directly within the packet. For that, two different options exist:

Payload: The first approach overrides existing bytes within the packet payload. For that, the section of the payload cannot be used by the application layer. Consequently, this approach is only helpful in the case of benchmark experiments of network elements without a productive application running in addition to the generated network load. This approach does not affect the size of the test packets.

Data header: An alternative approach is to add an additional packet header to the packet carrying the data as a piggyback. In our experiments, unused *TCP option* types were leveraged as they are ignored by all investigated networking hardware and software implementations. The

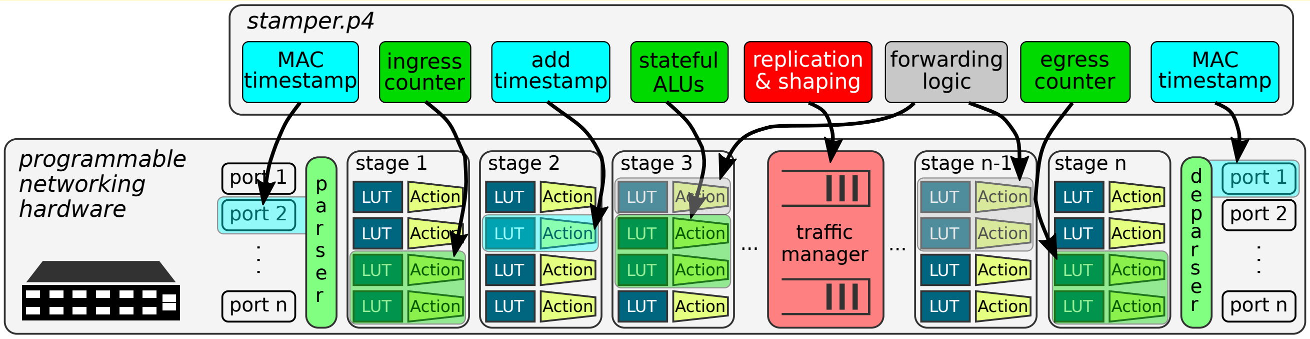


Fig. 3: Exemplary functional mapping of the Stamper p4-program on a programmable switch pipeline.

advantage of this approach is that any application on top of the transport layer can run within the load generators and will not be affected by the measurement. However, this increases the packet size, and by that, the Maximum Transmission Unit (MTU) of the DUT or load generator receiver may be exceeded.

In both cases, a new checksum of packets traversing the Stamper must be computed incrementally based on the old checksum and the modified packet data every time a packet data value is altered.

C. Stamper Realization with P4 Programmable Hardware

The data plane realization of the Stamper depends on the underlying hardware ranging from FPGAs to Smart Network Interface Cards (SmartNICs) or P4 programmable switches. In the following, we describe the main functional blocks and how to map them onto a P4 programmable Intel Tofino switch. Note that a SmartNIC platform is supported by P4STA as well.

As depicted in Figure 3, the data plane behavior of the Stamper is described in a P4-program called *stamper.p4* being mapped by a compiler on the switching hardware.

One key functionality of P4STA is timestamping. To achieve a measurement accuracy in the range of a few nanoseconds, timestamps are taken as late as possible before sending packets and as early as possible after receiving them for each packet in the data plane hardware. The ingress and egress Media Access Controller (MAC), which represents the piece of logic closest to the wire of modern switching hardware, can be used for that. The retrieved timestamp data is added to the packet as described in the previous subsection.

Further, all incoming and outgoing packets and bytes are counted for each port. The total number of timestamped packets, average latency and the total number of bytes is captured in the data plane by stateful ALUs, performing arithmetic operations based on their state and the current packet. Hence, possible losses between the Stamper and the external data capturing host can be detected.

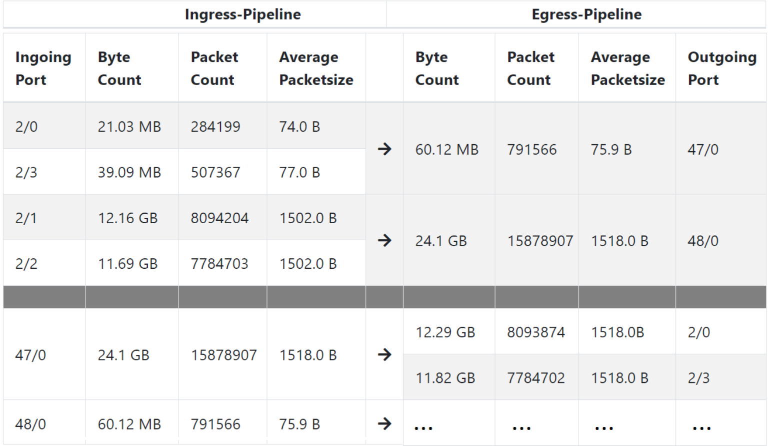
Finally, a table, *forwarding logic*, is applied for each packet to determine the output port and the destination MAC address can be updated. For each port, traffic replication and shaping can be applied. Replicating each packet from a single load generation server multiple times can achieve a very high load at small packet sizes. For example, an experimental $100 Mbit/s$ raw-socket-based packet source can be replicated 10^3 times to create a $100 Gbit/s$ load on the DUT with the same characteristics as before. This is possible by utilizing the multicast capabilities of the underlying P4 switch. By marking a status bit within the packet, duplicates can be detected and filtered out later on. Moreover, by shaping a DUT port of the Stamper, a burst-free packet flow with a specified rate can be realized.

To summarize, the Stamper implementation is responsible for retrieving the following information: 1) A timestamp for each packet before and after the packet as well as the 2) average, maximum and minimum latency over all packets and 3) packet loss counters are maintained. Realizing the previously mentioned functionality with the Intel Tofino networking switch leads to resource utilization of less than 10% of the most used resource such as SRAM and stateful ALUs.

D. High Performance Data and Packet Capturing

The external data capturing host is responsible for receiving and storing all measurement data attached to packets. For that, incoming packets are parsed and the information of interest is extracted and stored. Currently, we are facing two different implementation types, both having advantages and disadvantages:

Raw socket based capturing: Any application on a Linux-based system can open a raw socket on any network interface port to receive any packet that arrives on this port. On the one hand, this approach is quite simple. On the other hand, the performance of such sockets



Ingress-Pipeline					Egress-Pipeline			
Ingoing Port	Byte Count	Packet Count	Average Packetsize		Byte Count	Packet Count	Average Packetsize	Outgoing Port
2/0	21.03 MB	284199	74.0 B	→	60.12 MB	791566	75.9 B	47/0
2/3	39.09 MB	507367	77.0 B					
2/1	12.16 GB	8094204	1502.0 B	→	24.1 GB	15878907	1518.0 B	48/0
2/2	11.69 GB	7784703	1502.0 B					
47/0	24.1 GB	15878907	1518.0 B	→	12.29 GB	8093874	1518.0B	2/0
					11.82 GB	7784702	1518.0 B	2/3
48/0	60.12 MB	791566	75.9 B	→	...	...	...	...

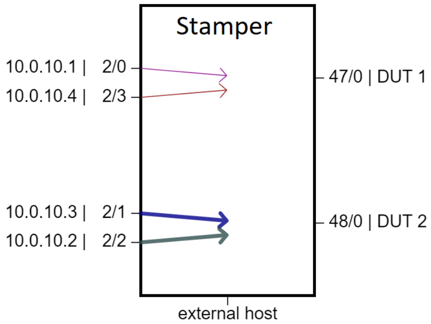


Fig. 4: User interface displaying the measurement data of the ingress and egress counter for all Stamper ports. Two load generators are connected to one DUT port each. The bold arrows on the right side indicate timestamped flows.

is comparably bad, and we observed that even under the best conditions, receiving rates stay below $1 Gbit/s$. Therefore, a downscale factor N can be configured in the Stamper, which avoids an overload of the external host. Then, only one out of N packets will be sent to the data sink. Unaffected by this, all counters and stateful ALUs in the Stamper behave as before.

High-performance poll mode driver: An alternative to the raw socket approach are user space drivers such as the DPDK framework [10]. These drivers allow much higher packet rates up to $100 Gbit/s$ even at comparably low packet sizes as the network interface card is entirely under control by the application. Further, the load can be distributed between several CPU cores for performance. However, such drivers require hardware support which may currently not be given in all network interface cards in production and the port must be unbound from the kernel networking stack. In addition, this approach can be used to capture a complete packet trace on the external host. Note that this can quickly exceed the main memory capacity and HDD writing speeds are too slow.

E. Analyzing the Measurement Results

After capturing raw measurement data, the last step is to process and analyze it. For that, we distinguish between per-packet data and aggregated information such as counter and stateful ALU values in the data plane of the Stamper. It is essential to quickly obtain preliminary results with minimal effort for both the aggregated data and the per-packet information time series. The screenshot of the P4STA user interface in Figure 4 depicts some of the aggregated information captured in the data plane of the Stamper. Concretely, it presents the number of received and transmitted packets and bytes for each port. Also, packet loss within the DUT and possible faulty behavior in the testbed setup can be easily detected. The ports of the Stamper on the right side, *e.g.*, `47/0`, map to the ports in the left-side table. Based on the retrieved data from the Stamper and external host, further information can be derived, including the latency over time, jitter, inter-packet arrival times, packet reordering and many more. These metrics can be computed automatically within the P4STA workflow. This is very important, especially for the large quantity of time series data, as the captured raw data is not human readable.

III. Experimental Results and Experiences

Network testing is crucial for validating and understanding high-performance networking hardware. The accuracy of this approach has been discussed in our preceding work [11] in detail. In the following, we present some results for a simple packet forwarding network function (DUT) realized on a SmartNIC. However, the presented measurement setup could be executed exactly in the same way for a switch to be validated for operating in a 5G front-haul network with strict forwarding requirements, *i.e.* no packet loss and reordering as well as a constant latency. We choose this DUT intentionally as its architecture, not being a straight pipeline, can theoretically cause packet loss, higher latency variation and packet reordering.

While performing the test, we created a constant bitrate-shaped test load of $7 Gbit/s$ and 1518 $byte$ packet size with the P4STA framework, which was applied to the DUT. Each packet was timestamped twice before and after the investigated network function, and the external host captured all measurement data. The analysis tool of P4STA allows the computation of the latency for each packet and additional metrics after the experiment.

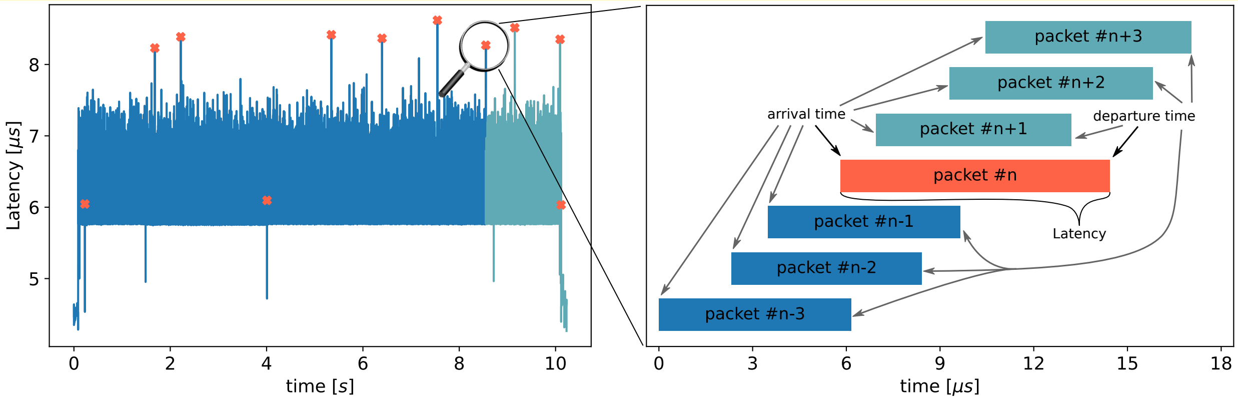


Fig. 5: Left: Latency time series of all packets for the device under test at 7 Gbit/s for each packet. Red crosses mark out of order packets. Right: Detail view on the arrival and departure time at the DUT for the marked out of order packet as well as preceding and subsequent packets.

The left plot in Figure 5 depicts the latency over time for all packets in one direction through the device under test with a time accuracy of nanoseconds. As the absolute latency and its range are in the order of a few microseconds, this high time accuracy is required. Other network functions, especially switches, can have latencies lower by a factor of up to ten and higher in the future [12] which requires even a higher time resolution.

The red crosses in the plot indicate out of order packets that were detected by P4STA. For example, as shown in detail on the right half of Figure 5, *packet n*, which was sent at around 8.5 *seconds* after the start, is delivered out of order as it was handed over to the network function between packets $n-1$ and $n+1$. However, it was sent out by the network function between *packet* $n+1$ and $n+2$. In total, we observed 11 out of order packets while the total number of packets was around 5.7 *million*. In this case, the reordering behavior can be explained by the internal architecture of the DUT. Transferring these results on the concrete example of a 5G fronthaul network, this reordering would cause invalid radio signals on the air interface. In general, such measurements enormously assist in understanding and validating the internal behavior of high-performance network functions.

The additional latency in this setup caused by the Stamper is negligible and depends on many factors such as the link speed and the used hardware. In the case of a Tofino-based Stamper and $10Gbit/s$ link speed, this additional latency is around $2\mu s$. Netronome SmartNIC-based Stamper implementations can cause an increase of up to $20\mu s$ at the same link speed. Higher link speeds result in lower latency of the Stamper due to its store and forward behavior.

IV. Related Work

Similar approaches for network testing have been discussed in related work before. Shabaz *et al.* proposed an open source load generator built upon the NetFPGA platform called Open Source Network Tester (OSNT) [13]. They benefit from the high time accuracy of programmable hardware compared to software-based approaches as well. However, no dynamic load patterns such as stateful TCP could be generated as they realized the load generation within the FPGA. Further, the maximum load is limited due to the number of ports and speed per port compared to programmable switches.

Another approach for fine-grained network monitoring is In-band Network Telemetry (INT), utilizing piggybacking of data attached to the packet traversing a network [14]. As soon as a packet enters an INT-capable network, the entry switch will add an INT header to the packet. Every traversing switch can add additional information to this header in a stacked way. Before the packet leaves the INT network, the last switch removes the INT header stack and sends it to an external collector, similar to the external host in our work. Compared to our approach, the overhead is much higher and load replication features could not be supported.

Last, we would like to mention the ERSPAN protocol, which allows to duplicate and encapsulate packets within the data plane for monitoring purposes. A timestamping accuracy similar to the results presented in this work can be achieved by leveraging the capability of hardware switches. Indeed, measuring the latency or jitter of numerous packets, e.g., a $100Gbit/s$ test run for $10s$, becomes very difficult as each packet must be duplicated before and after the DUT and the collected data must be assembled afterward by unique packet identifiers.

V. Conclusion

Network element testing and monitoring play an essential role in the innovation cycle of developing, manufacturing and deploying of networking hardware. The potential of existing programmable data plane hardware offers an alternative to specialized hardware for measuring and testing network element functionality. In this article, we demonstrated how programmable networking hardware can be utilized for network testing and monitoring using the open source P4STA measurement framework. By combining the benefits of software generation of test inputs and programmable hardware, the presented measurement framework is flexible enough to allow network testers to include their own test functionality while still achieving high measurement accuracy.

As data plane chip vendors are currently working on next generations of programmable hardware that promise improved programmability (such as stateful and complex operations) [15], we postulate that more advanced network testing functionality can be embedded directly in hardware. We encourage the research community to take advantage of the existing open source projects relying on programmable hardware. Since the initial publication of the P4STA framework, first vendors of testing hardware announced commercial products for network testing purposes based on programmable off-the-shelf networking hardware, on which also P4STA builds. For future work, the integration of other existing protocols, *e.g.*, ERSPAN and INT, shall be investigated.

Acknowledgment

This work has been supported by Deutsche Telekom through the Dynamic Networks 8 project, by the German Research Foundation (DFG) within the Collaborative Research Center MAKI as well as the project SPINE.

References

[1] R. Kundel, L. Nobach, J. Blendin, W. Maas, A. Zimber, H.-J. Kolbe, G. Schyguda, V. Gurevich, R. Hark, B. Koldehofe, and R. Steinmetz, “OpenBNG: Central office network functions on programmable data plane hardware,” *International Journal of Network Management*, 2021.

[2] C. Ranaweera, E. Wong, A. Nirmalathas, C. Jayasundara, and C. Lim, “5g c-ran with optical fronthaul: An analysis from a deployment perspective,” *Journal of Lightwave Technology*, vol. 36, no. 11, pp. 2059–2068, 2018.

[3] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey.” IEEE, 2014, pp. 14–76.

[4] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, “A survey on data plane programming with p4: Fundamentals, advances, and applied research,” 2021.

[5] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn,” in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. ACM, 2013, pp. 99–110.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, “P4: Programming protocol-independent packet processors,” vol. 44, no. 3. ACM, 2014, pp. 87–95.

[7] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, and B. Koldehofe, “P4STA: High performance packet timestamping with programmable packet processors,” in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium(NOMS)*. IEEE, 2020.

[8] “P4sta: High performance packet timestamping and load aggregation framework,” https://github.com/ralfkundel/P4STA, 2019.

[9] P. Orosz and T. Skopko, “Performance evaluation of a high precision software-based timestamping solution for network monitoring,” *International Journal on Advances in Software*, vol. 4, no. 1, 2011.

[10] Intel. (2014) Data plane development kit. Accessed last on: 28. May 2021. [Online]. Available: https://www.dpdk.org/

[11] R. Kundel, F. Siegmund, and B. Koldehofe, “How to measure the speed of light with programmable data plane hardware?” in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019, pp. 1–2.

[12] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, “It’s time for low latency,” in *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*. USENIX Association, May 2011.

[13] 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.

[14] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, “In-band network telemetry: A survey,” *Computer Networks*, vol. 186, p. 107763, 2021.

[15] A. Agrawal and C. Kim, “Intel tofino2 - a 12.9tbps p4-programmable ethernet switch,” in *2020 IEEE Hot Chips 32 Symposium (HCS)*, 2020, pp. 1–32.

Biographies

Ralf Kundel (ralf.kundel@kom.tu-darmstadt.de) obtained his B.Sc. and M.Sc. degree from Technical University of Darmstadt, Germany in 2015 and 2017 respectively. Since 2018 he is a Ph.D. student at the same university. In his research he focuses on quality of service aware network functions offloaded on programmable hardware. One main focus is on internet service providers access networks.

Fridolin Siegmund is a graduate student at Technical University of Darmstadt, Germany, where he received his bachelor in 2019 and is currently working on his master’s degree. His main research interest is in software defined networking.

Rhaban Hark obtained his Ph.D. in 2020 at Technical University of Darmstadt where he is currently heading the Adaptive Communication Systems Group. One main research interest is in monitoring software defined networks.

Amr Rizk received the doctoral degree (Dr.-Ing.) from the Leibniz Universität Hannover, Germany, in 2013. Since 2021 he is a professor at the department for computer science at the University of Duisburg-Essen, Germany. His interests include network performance evaluation and communication system applications.

Boris Koldehofe received the Ph.D. degree from the Chalmers University of Technology, Gothenburg, Sweden, in 2005. He is currently Professor of Computer Science at the University of Groningen. He has extensive research and teaching experience in the area of networked and distributed systems.