ACCURATE LIVE VISUALIZATION OF PACKET DELAYS, INTER-ARRIVAL TIMES, AND BURSTINESS FOR REAL-TIME NETWORKS

Alexej Grigorjew[1], Stefan Geißler[1], Maximilian Ziegler[1], Lukas Kilian Schumann[1], Philip Diederich[2], David Hock[3], Tobias Hoßfeld[1], Wolfgang Kellerer[2]

[1]University of Würzburg, Chair of Communication Networks, Würzburg, Germany
[2]Technical University of Munich, Chair of Communication Networks, Munich, Germany
[3]Infosim GmbH & Co. KG, Würzburg, Germany

Contact: alexej.grigorjew@uni-wuerzburg.de

ABSTRACT

In this demonstration, a live delay and traffic volume visualization is presented. It is created with consumer off-the-shelf components, provides sub-microsecond accuracy, and provides sufficient performance for line rate measurements without sampling. The tool can help to accelerate the development and operation of real-time networks.

1 Introduction

Accurate traffic measurements are an important source of information for the deployment and operation of real-time networks. They are required prior to regular operation for the latency assessment of networking devices, and for the assessment of traffic volume from end devices. Information about the maximum latency of switches in certain configurations and for certain types of traffic load is vital to reservation protocols, as the computation of the accumulated maximum latency relies on accurate upper delay bounds. In addition, upper bounds for the traffic volume of end devices are equally necessary, and at the same time, can be difficult to obtain, as both the operating system and the physical network interface can be a source of additional delay, which can increase the traffic's burstiness.

Further, a live monitoring tool can be useful during network operation. Networking devices are essentially black boxes, and slightly different traffic patterns can lead to different latency characteristics altogether. In scenarios where deterministic guarantees are required, it is vital to identify any changes in a timely manner. Similarly, misbehaving or malfunctioning devices that exceed their traffic limits should be quickly identified to prevent cascading impairments.

Finally, the above requirements for latency and traffic volume measurements must be met under certain constraints. The measurement accuracy must be sufficient for the intended range. With many modern switches featuring single-digit microsecond delays, the measurement should provide a sub-microsecond accuracy. More importantly, the measurement should be transparent and should not impair the actual operation of the network in any way. Lastly, the measurement methodology should be able to sustain measurements during peak load conditions.

In this abstract, a proof of concept demonstration is described that is able to fulfill these requirements, including a live visualization in a browser, with off-the-shelf hardware. It presents the general testbed setup in Section 2, the software setup in Section 3, and highlights some intended demonstration scenarios in Section 4.

2 Testbed Setup

The hardware setup is similar to our previous hardware timestamping testbed, which was briefly described in [1]. Figure 1 presents a general overview of the testbed setup. A traffic source sends packets towards a traffic sink, which go through a Device Under Test (DUT). The DUT can be a router, switch, or any kind of networking device whose delay might be of interest. Before and after passing the DUT, the packets are replicated by transparent network taps, and their duplicates are sent towards a measurement device. To achieve the required accuracy, it is vital to select

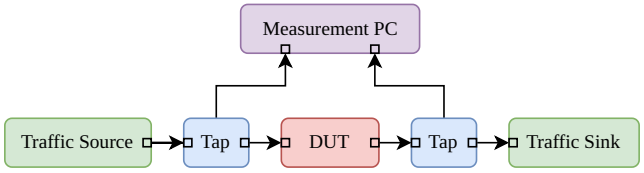

Figure 1: General setup for hardware-timestamped delay measurements of a Device Under Test (DUT).

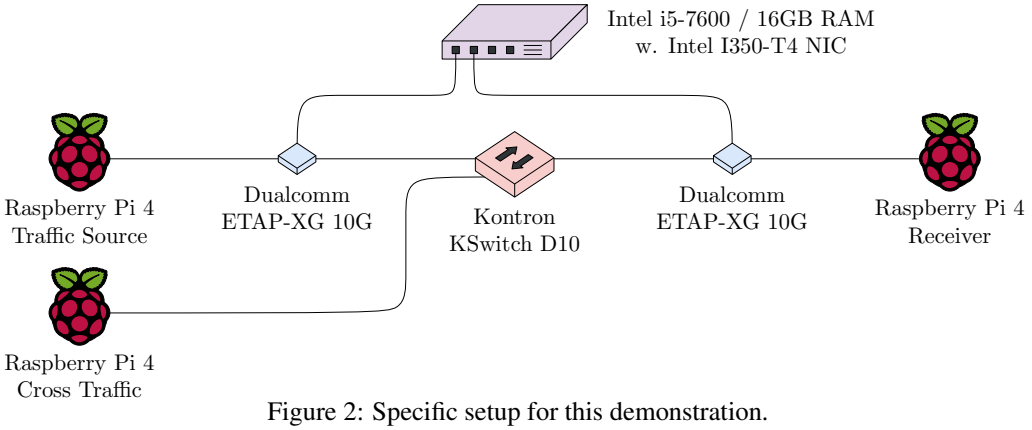

Figure 2: Specific setup for this demonstration.

network taps that provide the smallest possible delay variance, e.g., by operating as close to the physical layer as possible. The measurement computer is equipped with a multi-port Network Interface Card (NIC) that is capable of hardware timestamping for all incoming packets at line rate. More specifically, the NIC features a single controller for multiple ports, so the hardware timestamps of both used ports are based on the same reference clock.

This general setup is fairly independent of the exact hardware being used. The methodology provides the means for passive measurements, so both traffic source and traffic sink can be exchanged freely. The method can be applied to various link speeds with Ethernet copper wires (RJ45 interfaces) and optical cables, as long as suitable network taps and NICs are available. It has successfully been tested with both optical and copper wires using Dualcomm ETAP-XG 10G taps, and with $1\,\mathrm{Gbit/s}$ (Intel I350-T4 NIC) [1] and $10\,\mathrm{Gbit/s}$ (Intel X550-T2 NIC) [2] link speeds.

For this demonstration, the specific hardware choices are depicted in Figure 2. Traffic source, traffic sink, and cross traffic generator are represented by a Raspberry Pi 4, each. The traffic source is equipped with several Tinkerforge[1] sensors, and all three devices are capable of transmitting artificial traffic for these measurements. Two Dualcomm ETAP-XG 10G[2] taps with 10GBase-T SFP+ modules are used for traffic duplication. The DUT is represented by a Kontron KSwitch D10[3], which was specifically designed for low latency real-time networking and provides several interesting features, such as priority transmission selection and various shapers. For comparison, it can be exchanged with any standard Ethernet switch during the demonstration, such as a MikroTik hEX[4], which can provide basic routing functions at a very low cost. The measurements of the duplicated packets are conducted by a standard Intel i5-7600 machine equipped with 16GB RAM and an Intel I350-T4 NIC[5]. It is important to connect both cables from both taps to adjacent ports of the same controller, otherwise the underlying hardware timestamps are not comparable. The I350's clock has a frequency of $125\,\mathrm{MHz}$, which means that timestamps are as accurate as $\pm 8\,\mathrm{ns}$. This accuracy is sufficient to detect changes in delays caused by varying Ethernet cable lengths, which can be illustrated during the demonstration.

3 Software Setup

The software components in this measurement tool can be divided into three major categories, as depicted in Figure 3. First, the packets arriving at the network interfaces are compared to a pre-configured eBPF filter by the Linux kernel.

[1] https://www.tinkerforge.com/en/
[2] https://www.dualcomm.com/products/etap-xg-10g-network-tap
[3] https://www.kontron.com/en/products/kswitch-d10-mmt-series/p170637
[4] https://mikrotik.com/product/RB750Gr3
[5] https://www.intel.com/content/www/us/en/products/sku/84805/

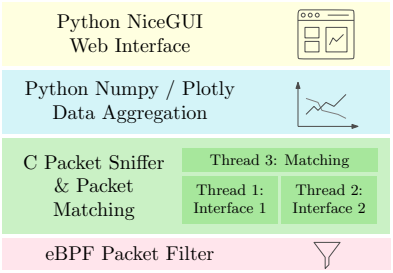

Figure 3: Software architecture behind the live delay visualization.

The filter rules are determined by the user via the python web interface. The filter is compiled by using the libpcap[6] library, and applied to a raw socket with a system call. This step allows to drop unwanted packets early and to achieve excellent performance even on older machines.

Next, the packets that pass the filter must actually be received by a program. This step is critical, as the packet reception technique strongly impacts the performance of the overall system. Traditional `recvmsg` system calls cannot provide sufficient performance for line rate packet capture. After comparing various packet reception techniques[7], for this demonstration, a shared memory buffer with `PACKET_MMAP`[8] and `PACKET_RX_RING` is used. In addition, the NIC must be instructed to apply hardware timestamping for all incoming packets with the correct system calls. For more details on the these system calls, please refer to the sources on GitHub[9]. These steps must be performed in parallel for both active interfaces of the measurement PC, each one is served by an individual thread. Further, the packets received by one interface are collected in a hashed data structure to be matched against the packets received by the other interface. Only after measuring the same packet on both interfaces, the delay of the DUT can be inferred by comparing their timestamps. Hashing and matching is performed by a third thread of the same C program.

Next, the pair of timestamps is passed to a higher level python program, along with a five-tuple that identifies the individual flow (`srcIp`, `dstIp`, `protocol`, `srcPort`, `dstPort`). This is done by a named Unix pipe. To increase performance, the python program polls the contents of the pipe in batches after pre-configured intervals. The python program aggregates the data for each individual flow, calculates the KPIs such as delay, inter-arrival times, and burstiness, and prepares graphical visualizations of these KPIs in plots. Further, the python script is the main executable of this tool and starts the packet sniffer and the web interface as sub processes.

Finally, the web interface uses the python NiceGUI[10] framework to present the visualizations to the user. Further, the web UI provides the means to select the displayed flows, and to configure certain aspects of the python aggregator, such as refresh interval and retained data for the time series. In detail, the measurement can be started and stopped, new flows can be detected, and the delay computation can be calibrated, e.g., to account for unequal cable lengths. A screenshot of the current interface is presented in Figure 4. Note that minor improvements are to be expected until the date of the conference.

4 Demo Scenarios

Next to the general use cases, including assessment of switching delay and traffic volume, as well as live monitoring of critical networking components during operation, the demonstrator can be used to illustrate various aspects of real-time networking with deterministic upper delay bounds. The following scenarios represent a non-exhaustive list.

(a) What is the accuracy of the delay measurement? For this test, a controlled additional delay can be introduced by increasing the cable length. Assuming the propagation speed through the copper wire is roughly 0.7 times the speed of light, this equals $0.21\,\mathrm{m/ns}$, or an extra $4.77\,\mathrm{ns}$ for each extra cable meter.
(b) How does the latency of the networking device change under varying traffic conditions? What happens when cross traffic fully saturates the outgoing link? What happens to an individual flow's delay when its priority is increased? For these purposes, a second web interface is built to control the behavior of both traffic sources.

[6] `https://www.tcpdump.org/`

[7] The results will be presented at WueWoWAS'23 (`https://lsinfo3.github.io/WueWoWAS2023/program/`) and cited here in the camera ready version.

[8] `https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt`

[9] Past repository: `https://github.com/lsinfo3/hwtstamp-snippets/` – improvements planned for camera ready version.

[10] `https://nicegui.io/`

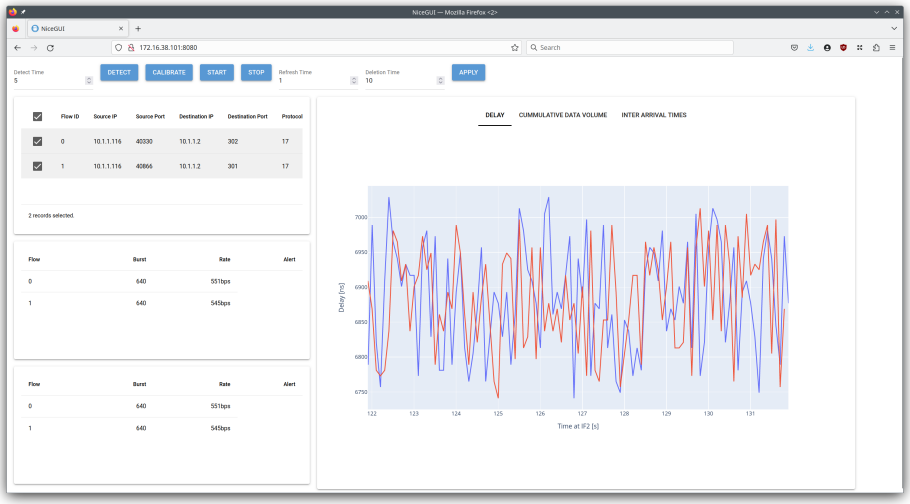

Figure 4: Screenshot of the delay monitoring web interface.

(c) How does the measured traffic volume compare to the configured values on the traffic sources? What is the additional overhead caused by the variance of the software implementation, and caused by the network interface? Does the burstiness increase from this variance?
(d) How does the variance introduced by the networking device affect the measured traffic volume after passing the switch? Does the additional variance increase the burstiness of the flows? How much traffic must be present in order to see a measurable effect?
(e) General measurements of the distributions of various KPIs are possible, most notably, inter-arrival time distributions, frame size distributions, and service time distributions. This data can be acquired quickly with our tool and can help during the development of queuing theory models for stochastic guarantees in the future.

5 Demo Requirements

For the demo, we bring all the necessary devices, as depicted in Figure 2. For the setup, a large enough table is required to comfortably lay out the testbed, i.e., a typical computer work space. In addition, we bring two power strips to plug in all of our devices, so two power outlets are necessary. The visualization will be displayed on a laptop screen.

References

[1] A. Grigorjew, P. Diederich, T. Hoßfeld, and W. Kellerer, "Affordable measurement setups for networking device latency with sub-microsecond accuracy," in *Würzburg Workshop on Next-Generation Communication Networks (WueWoWas'22)*, 2022, p. 5. DOI: 10.25972/OPUS-28075.

[2] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "How Low Can You Go? A Limbo Dance for Low-Latency Network Functions," en, *Journal of Network and Systems Management*, vol. 31, no. 1, p. 20, Dec. 2022. DOI: 10.1007/s10922-022-09710-3. [Online]. Available: https://doi.org/10.1007/s10922-022-09710-3 (visited on 05/12/2023).

Acknowledgments

This work was partly funded by the Deutsche Forschungsgesellschaft DFG (German Reasearch Agency) under Grant No 316878574, project SDN-App, and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, under grant number DIK0250/02, project KOSINU5.