Revisiting Benchmarking Methodology for Interconnect Devices

Daniel Raumer, Sebastian Gallemüller, Florian Wohlfart, Paul Emmerich, Patrick Werneck, and Georg Carle
Technical University of Munich, Garching b. München, Germany
{raumer|gallenmu|wohlfart|emmericp|werneck|carle}@in.tum.de

ABSTRACT

Ever growing demand for network bandwidth makes computer networks an area of constant development and fast adjustments. The steady change makes good performance assessments equally necessary and challenging. This development motivated us to revisit the established benchmarking methodology. We provide an overview of the state-of-the-art in router benchmarking, the currently available benchmarking tools, and challenges for benchmarks. A discussion of benchmarking results for three different devices (routers based on Linux and FreeBSD, and a MikroTik router) reveal different properties currently not covered by standardized benchmarks. We conclude by adding tests to the common benchmarking methodology reflecting these properties to make the results more valuable.

The prototype software implementation of our own benchmarking tool and its measurement reports are publicly available [0].

CCS Concepts

•Networks → Network measurement;

Keywords

Benchmarking methodology, RFC 2544

1. INTRODUCTION

Networks face increasing traffic caused by a growing number of connected devices and data intensive applications. To keep up with this demand, computer networks develop fast and change constantly. In recent years, we have seen trends such as the increase of general purpose hardware in networking and the transition from single-purpose machines to devices with multiple capabilities. These changes require reconsidering the traditional ways to assess the performance of networking equipment. Besides specifications and promises by the vendors, independent benchmarks are an important way to rate and compare the performance of different devices. RFC 2544 is a standardized test to benchmark a wide range of networking appliances. Although finalized in 1999, it is still of relevance today as it is the foundation for many benchmarks in the area.

In this paper, we revisit benchmarking methodology for interconnect devices. We conduct several measurements and show the properties of different interconnect devices. Moreover, we present challenges and suggestions how to reflect those effects with relevant benchmarks.

We provide a short history of benchmarking methodology and tools that are fundamental to current best practice in Section 2. Further, we present our own benchmarking solution based on inexpensive commercial-of-the-shelf (COTS) hardware and measurement results from state-of-the-art and above-standard benchmarking discussing their expressiveness and shortcomings in Section 3. After that, we suggest changes to the currently established benchmarking methodology in Section 4.

2. METHODOLOGY

In the following, we look at approaches to benchmark interconnect devices, investigating relevant standards and solutions. We discuss shortcomings and derive our requirements for an improved benchmarking tool.

2.1 Relevant Benchmarking Standards

The IETF started its standardization activities on benchmarking of network devices with the foundation of the benchmarking methodology working group (bmwg) in 1989 [5]. Performance indicators were defined in RFC 1242 [8] in 1991. In 1999, the bmwg published RFC 2544 [9] which has developed into the de facto standard for router benchmarking [37]. Since then, the bmwg has provided further documents that define additions for IPv6 benchmarking [30], benchmarks to determine the "reset time" [3], and other benchmarking tests for recovery [3] and FIB-dependent (Forwarding Information Base) performance [38]. The bmwg initiated drafts extending FIB-dependent performance measurements in 2005 or standardizing IPsec performance measurements in 2009 which were both never published as RFCs. Not all issues discussed inside the bmwg are released as an RFC, e.g., the extension of FIB-dependent performance tests to take the router internal instantiating into account in 2005 or the discussions about IPsec performance in 2009 never left draft level. Documents from other organizations, such as the ITU-T Y.1564 [22] (EtherSAM) or the MEF 14 [25], give additional comments or partially redefine the benchmarks of

ANRW '16, July 16, 2016, Berlin, Germany
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4443-2/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2959424.2959430

RFC 2544. The following key performance indicators (KPI) are the most important defined by RFC 2544:

- **Throughput:** The highest rate that the device under test (DuT) can serve without loss.
- **Back-to-Back frame burst size:** The longest duration (in frames) the DuT forwards bursts without loss.
- **Frame loss rate:** The percentage of dropped frames under a given load.
- **Latency:** The average duration a packet needs to be processed within the DuT.

After the most basic test run, the test should be repeated under each available condition separately including different frame sizes, bursty traffic, and number of rule entries. If the number of conditions or combination of conditions is feasible, the tests may also be performed while successively adding conditions. Thus, other metrics, e.g., for the FIB-dependent performance comparison [38] can be derived.

As RFC 2544 is the de facto standard for benchmarking networking devices, many vendors release RFC 2544-compliant measurements to promote their products [28, 11].

2.2 Available Benchmarking Solutions

The available tools for RFC 2544 conformant tests can be divided into two groups: Hardware-based devices, a powerful but costly approach, and software-based solutions with high flexibility, lower costs but moderate traffic rates and less precision.

Hardware-based benchmarking devices can accurately control the sending rate and can perform precise latency measurements. Such devices are available from different vendors such as Ixia [2], Spirent [36], and Xena [40] providing a collection of predefined benchmarks, which are either complex to adapt, or cannot be extended. Although they provide well-defined and reproducible performance tests, the high costs prevent widespread utilization [7].

In 2007, Bolla and Bruschi [6] published an in-depth performance study of software-based routers, including an RFC 2544 test with such a hardware-based benchmarking device and additional router internal performance counters for their analysis. Although the insights from internally gathered performance statistics may be helpful, the use of these counters may influence the performance or they may not be accessible for proprietary devices.

A less expensive and more flexible hardware benchmarking device is NetFPGA which is an open source FPGA-based network card that can be used for implementing device benchmarks [26]. NetFPGA-based traffic generators [12, 20] provide accurate inter-packet delays and capabilities to measure precise latencies. Rotsos et al. [33] use a NetFPGA for OFLOPS – an evaluation framework for OpenFlow devices – and are able to provide latency measurements in the sub-millisecond range.

Basic **software-based** packet generators can be used for performing RFC 2544 conform tests. These tools rely on cheap commodity hardware but suffer from comparatively low performance and inaccuracies caused by COTS systems' architectures [7]. In fact, most software packet generators can only handle packet rates that are not sufficient to saturate a single 10 GbE link with minimum-sized packets. This is valid for most traffic generators that emulate realistic traffic, e.g., Harpoon [35], but also for network benchmarking tools such as `iperf` [21] that promise to measure network bandwidth, jitter, packet loss, etc. Even commercial solutions like Candela LANforge fail to achieve such packet rates [10].

High-speed packet processing software does not rely on the OS network functionality but replaces it with specialized frameworks for efficient packet transmission. Examples are zsend on PF_RING ZC [27], packetblaster on Snabb [34], Pktgen-dpdk [29] on DPDK [1], or pkt-gen on netmap [31]. These tools are able to send simple packets or replay a PCAP file at millions of packets per second. However, none of these solutions is able to generate accurate inter-packet delays and to measure precise latencies - a fact that leads to uncontrolled micro-bursts and jitter hence influencing the results [7, 14, 12].

2.3 Benchmarking Requirements

Device benchmarks need to be *(i) valid*, i.e., the benchmark outcome reflects real-world device behavior, *(ii) reproducible*, i.e., other organizations can reproduce and verify published benchmarks, and *(iii) comparable*, i.e., we can directly compare the benchmarks executed by different organizations. With packet processing becoming more software-based, current benchmarking methodologies cannot satisfy these requirements. We identify three key shortcomings of existing benchmarking solutions that are challenging to implement.

Validity: Existing benchmarking approaches often rely on simplistic traffic patterns that do not represent realistic use cases of network interconnect devices. Even the most basic interconnect devices such as switches or routers show non-trivial worst-case performance (depending on the traffic applied) when implemented in software. First, the inter arrival times between incoming packets influence the batching behavior [15]. Second, the ordering and diversity of incoming packets stresses the cache and thus impacts packet processing performance. More complex interconnect devices exhibit completely non-trivial performance properties, e.g., Snabb [34] – a framework for implementing network functions – relies on just-in-time compilation leading to unpredictable side-effects. Therefore, test traffic for valid benchmarking requires packet diversity and needs to consider relevant performance limiting factors present in a multitude of devices. The relevant factors have to be reconsidered and updated as the tested systems evolve.

Reproducibility: Due to the increasing number of options to configure and implement tested devices, benchmarking needs to document the DuT configuration. Software-based packet processing is not an isolated component, it requires an operating system, device drivers, and an execution platform, all of them offering plenty of choices for configuration. Choices include settings directly related to the tested functionality (e.g., routing table), to the operating system or device drivers (e.g., scheduling, CPU core binding, network stack version or fast packet processing framework), and to the underlying platform. The choice of an execution platform offers many degrees of freedom, with numerous choices for CPU (differing in instruction set, optimizations, clock rate, number of cores, cache hierarchy, etc.), network interface card (offloading features), system buses, RAM, and virtualization solutions. Even many proprietary off-the-shelf devices, that come bundled with specific hardware, allow updating components and installing add-ons to extend the functionality of a device. Minor changes to device configuration, for instance, the batch size [18] can have a large impact on performance. Therefore, a mere specification of the tested topology and test traffic is not sufficient to reproduce results. To enable others to verify benchmarking results of software-based packet processing applications, the tested device must be documented in detail including all applied settings, the operating system, virtualization if applicable, and a description of the underlying hardware. Ideally, the description of a tested device is given in form of an OS image and scripts that automate all necessary steps to configure the device into the precise state which was benchmarked. Automation also minimizes configuration errors. If the benchmark requires multiple test cycles of a reconfigured tested device (for instance, exchanging the routing table), the load generator and device reconfiguration should be coupled and automated to further narrow the chance of misconfiguration.

Comparability: As the diversity of hardware platforms is almost unlimited, virtually every published benchmark of software-based packet processing applications uses a different combination of hardware, rendering the benchmarks incomparable. While the CPU is the prevalent performance-limiting factor in software-based packet processing, other components such as the network interface cards and PCI buses also need to be considered [17]. Even on the same hardware, comparative studies can be challenging as the required software components also need to be configured in a comparable manner [18]. We consider the definition of a standardized benchmarking hardware platform unfeasible. Therefore, we need to look for abstractions of the packet processing performance from the underlying hardware. This means determining and describing only those hardware properties that have a significant impact on the resulting packet processing performance. Modeling the packet processing performance is a topic of ongoing research.

These problems in device benchmarking methodology also affect benchmarking solutions. In addition to their traditional requirements of fast and accurate packet generation and measurement, benchmarking solutions should also be able to generate application-specific traffic and handle the configuration of the tested device.

3. SELECTED MEASUREMENTS

In the following, we show different device behavior and how to measure it. Therefore, we perform RFC 2544 benchmarks of three DuTs: Linux routing, FreeBSD routing, and an off-the-shelf MikroTik router. Although there have been numerous efforts to improve performance, we rather selected *representative DuTs*, than highly optimized and specialized software which we consider unsuitable for argumentation about benchmarking methodology. All three DuTs process packets in software, which was not common when RFC 2544 was defined. For the discussion of special effects, we also revert to other results that have been published and reviewed as part of studies on high throughput [13] and traffic generation [14].

The test setups consist of two separate machines – one for benchmarking and one DuT (see Figure 1). The egress ports are connected to the ingress ports of the other device respectively. For testing the DuT forwards traffic from its ingress port to its egress port which allows the benchmarking device to determine the DuT's precise forwarding capabilities.

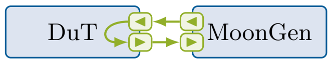


Figure 1: Setup

3.1 RFC 2544 Software Implementation

Benchmarking data are generated with our own software solution for automated RFC 2544 benchmarking that supports high traffic rates, precise latency measurements, and the generation of different traffic patterns. It is based on MoonGen [14] which is easy to extend due to its modular architecture. Prior to each benchmarking test the MoonGen RFC 2544 module automatically configures the DuT according to the guidelines from RFC 2544. All performance benchmarking tests run without altering the configuration other than specified by the benchmarking tool itself. Automatic configuration creates reproducible test results and prevents manipulations that could enhance the test results.

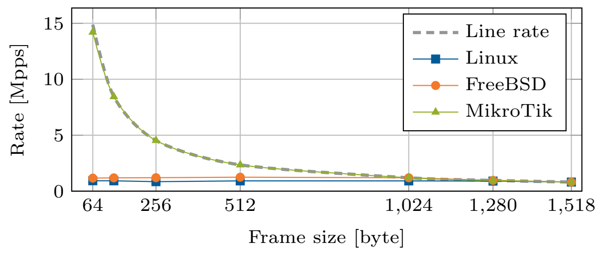
In the following, we only point to measurement results that show special effects. We provide the prototypic benchmark tool and the full benchmarking reports generated by our framework on our website [0].

3.2 Results of RFC 2544 Benchmarks

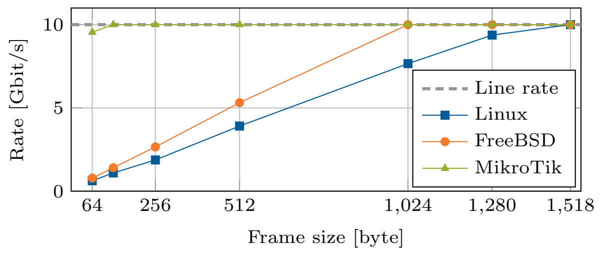
Throughput tests determine the maximum throughput in Mpps and Mbit/s. For software-based packet processing systems the throughput is either limited by link capacity or by the processing unit. The memory bandwidth is usually not limiting [17]. Packet processing costs mainly depend on the number of packets (i.e., the number of headers to process) rather than packet size. Therefore, the processing unit typically limits throughput when many small-sized packets are to be processed, the link capacity limits throughput for larger-sized packets. Figure 2 shows the results of the throughput benchmark for different frame sizes. The throughput measured in bit per second on the wire (Fig. 2b) shows linear behavior while the goodput which is not shown here further profits from the decreased per byte overhead of increased data units.

Note that this is not a fair comparison – and not meant to be – between Linux and FreeBSD since Linux was configured with a firewall and FreeBSD without. We previously benchmarked the same Linux system without firewall rules and achieved 1.58 Mpps [17]. Nevertheless, the unfair comparison fully complies with the guidelines, as we did not deactivate any features while we violated the guidelines in the fair case.

RFC 2544 demands to perform the tests in a manner that all processing paths are covered. Therefore, the comparison between MikroTik and the other devices is flawed due to the used traffic pattern and default device configuration: The MikroTik router reports the utilization of all 36 cores via SNMP. The processing load is independent of the number of flows, indicating that per-packet load balancing is used when a single large flow is applied. Linux and FreeBSD routers only use one core as they load balance via hashing over layer 3 and 4 addresses. RFC 2544 suggests to use 256 different flows in a second test run after establishing a baseline with a single flow but internal load balancing mechanisms still may require explicit activation, be undocumented, or unknown. For our DuTs we tested the effects of multiple flows: Linux scales linearly with the number of flows up to



(a) Packets per second



(b) Bit per second

Figure 2: Throughput

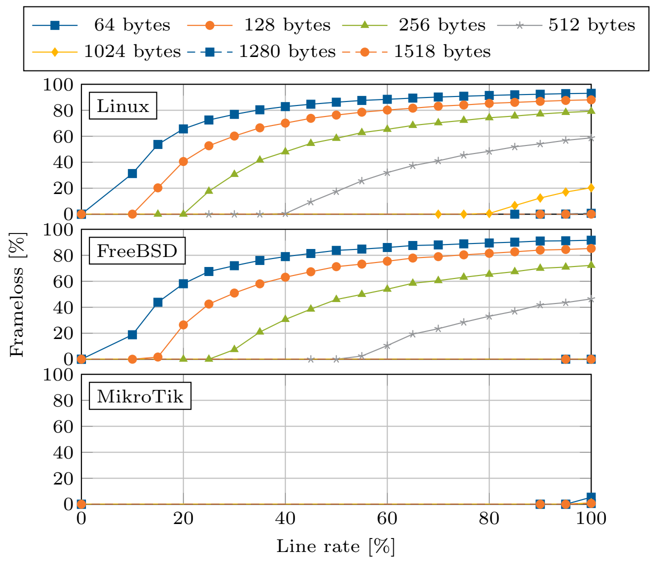


Figure 3: Frame loss percentage

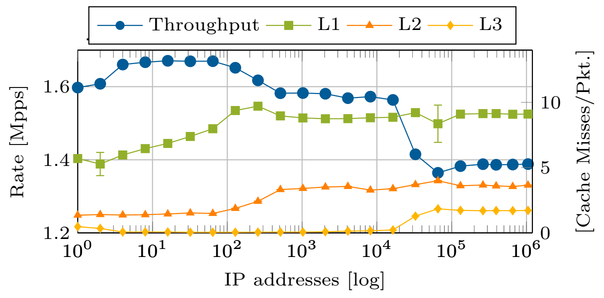


Figure 4: FIB-dependent performance of a Linux router

the four available CPU cores, FreeBSD only increased from 1.3 Mpps to 2.9 Mpps. The number of flows had no effect on the MikroTik throughput, providing further evidence of per-packet load balancing.

RFC 2544 conformant *latencies* are measured under the previously determined maximum load without frame loss. Our benchmark found that FreeBSD has a lower latency compared to Linux that is explained by the polling and batch-processing technique used in Linux. Linux in opposite achieves a higher throughput with the same configuration.

The MikroTik router achieves an average forwarding latency of 60 μs with 64-byte packets at 95% line rate. However, MikroTik achieves line rate with most packet sizes and the RFC 2544 compliant latency measurements for bigger packets were therefore executed at full line rate. Fill levels of buffers can never decrease with a line rate load as the incoming traffic is the same rate as the maximum rate by which the buffer can be drained. For software routers whose network card or CPU is in a sleep state, their buffers are filled up to some degree while the system is processing the first packets of a stream. The resulting latency is then randomly depending on the warm-up characteristics and varies between independent test runs, but not within a single unbroken stream. This effect is visible with all latencies measured at line rate for all three test devices.

The *frame loss rate* states the percentage of lost frames. Figure 3 displays the results of the frame loss benchmark for our DuTs. Frame losses occur as soon as the devices are overloaded. The actual throughput may still increase or decrease beyond the maximum throughput point. For the three devices here, such a behavior does not occur as traffic is dropped at the network card due to the lack of memory descriptors. This does not influence the bottleneck component, the CPU core. In general, effects due to increasing the load beyond the maximum throughput are atypical and the test not generally relevant. However, these effects can be measured for virtual machine scenarios [16].

For certain devices tests like the RFC 2544 frame loss rate test may be omitted. Following this argumentation, we have to add tests for certain device properties. Figure 4 shows how routing table size influences the maximum throughput of our Linux router without a configured firewall. The cache misses that we added for explanation have been determined with the Linux tool `perf stat` in 10 30-second runs for each packet rate and FIB size. We plotted the 95% confidence interval where the error bars were bigger than the mark indicating the average. The throughput is neither constant like with routers that store their routing table in content-addressable memory (CAM), nor decreasing according to the algorithmic complexity of the lookup data structure in Linux. In Figure 4, the throughput decrease is directly related to the increase of cache misses on each cache level. The lower rates for 1 and 2 flows are caused by postprocessing routines and not by the FIB.

3.3 Meaningful KPIs for Latency

In comparison to throughput, latency varies even under unchanged conditions (like a benchmark). Therefore, we went beyond the requirements of RFC 2544 and took 10 000 measurements while the RFC requires only 20. This large

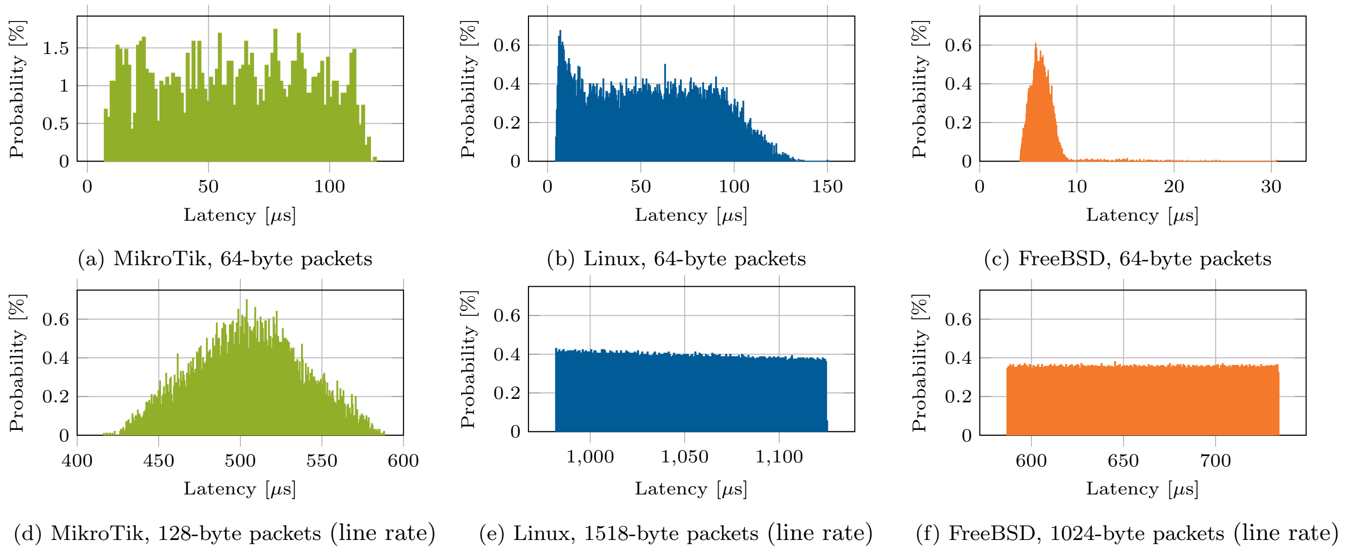


Figure 5: Forwarding latency of MikroTik, Linux, and FreeBSD

number is used to visualize the results in histograms and to demonstrate why latency should not just be reported as an average value. We prefer to use histograms over CDFs as we think that these allow for a easier recognition of long tail distributions. The histograms aid our discussion where the average latency can be misleading.

The first one is a long-tail distribution that is often encountered in software systems, especially ones that are virtualized [39]. Both our FreeBSD and Linux results exhibit such distributions as shown in Figure 5b and 5c. The highest 99.9th percentile in relation to the average was observed for 128-byte packets. Linux has a 99.9th percentile of 142 μs (22 times the average) and FreeBSD 38 μs (5.5 times the average).

The second example are multimodal distributions that can happen if packets pass through different pathways in a system. One example where this occurs is when packets from multiple incoming ports are forwarded to a single outgoing port. We have encountered the phenomenon of a bimodal latency distribution in previous work while benchmarking an AS5712-54X switch running PicOS with OpenFlow (Fig.3 in [13]). The switch forwarded most of the packets in either ∼0.9 μs or ∼3.6 μs. Both the average and median latency are meaningless in such a case.

3.4 Measuring Latency at the Right Load

RFC 2544 requires measuring the latency at the maximum load of the DuT determined earlier. Such a load is not a realistic scenario for a DuT as any forwarding device running under full load is overdue for a replacement. For example, the ITU-T_Y.1564 [22] defines latency measurements at the committed information rate (CIR), the bandwidth at which a device claims to hold certain performance guarantees which may be below the maximum throughput that the device can achieve.

One argument for latency measurements at the maximum packet rate is that such a measurement represents the absolute worst case for the latency. However, this is false. Devices may actually perform worse at lower packet rates due to power saving features. One example is the `ixgbe` driver in combination with Linux which can exhibit the worst performance at medium packet rates in the default configuration as the driver heavily throttles interrupts. We have discussed this effect in detail in a previous publication [15].

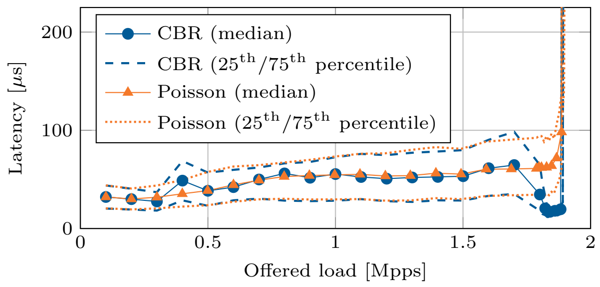


Figure 6: Latency response of a Linux packet forwarder under CBR and Poisson traffic [14]

The second problem with this methodology affects devices that achieve full line rate (cf. Section 3.2). The impossibility to drain filled buffers is particularly problematic with software routers. Any processing delay can fill up the buffers. Such a delay might be imposed by initial sleep states of the network card or the processing unit, short interruptions due to scheduling, or when the outgoing port is slightly slower than the incoming port, e.g., due to clock drift between independent ports. The latency is then increased by the serving time of the preceding packets. A packet rate that is slightly lower than full line rate solves this measurement artifact.

For example, the MikroTik software router achieves an average forwarding latency of 60 μs with 64-byte packets at 95% line rate in Figure 5a. However, as soon as the packet rate is increased to full line rate with a frame size of 128 bytes, the average latency jumps to 500 μs in Figure 5d. Similar effects can be seen for Linux and FreeBSD in Figure 5 that show a uniform distribution due to batch processing. We suppose that the different distribution of the MikroTik router is an effect of the per-packet load balancing across the 36 cores via the on-chip mesh network.

3.5 Traffic Patterns Matter

One important aspect of the test traffic is the traffic pattern, i.e., the distribution of the inter-packet gaps in the test traffic. This distribution affects the buffers of the DuT and therefore the latency. As benchmarks have to be reproducible, a precise definition and exact reproducibility of the generated traffic is required. Therefore, RFC 2544 calls for constant bit-rate (CBR) by default and recommends to run "some" of the tests with bursty traffic as well [9]. It does not specify which of the tests should also be run with bursty traffic.

CBR is an unrealistic type of traffic and so is bursty traffic. Real traffic follows more complicated distributions: over long timescales (hours to days) the traffic exhibits a self-similar pattern [23]. Such traffic can be approximated over short time scales with a Poisson process [32] which is simpler to implement than the aforementioned patterns.

We compared the impact of traffic patterns on the behavior of Linux in previous work [14]. Figure 6 shows the latency of Linux under increasing load, providing more insight than an RFC 2544 conformant test. The latency differs with CBR and with Poisson traffic before the system overloads. We have analyzed this behavior which is an artifact of the interaction between network interrupt handling in Linux and the `ixgbe` driver [15]. This effect disappears when using Poisson traffic, so it likely does not appear in the real world, but only when being tested under unrealistic circumstances.

4. IMPROVING BENCHMARKS

Based on the identified shortcomings of current benchmarking methodologies (Section 2.3), the shown measurements (Section 3), and previous research [16, 14, 17, 18, 4], we derive specific recommendations to improve interconnect device benchmarking. Our proposals aim to extend RFC 2544 and similar guidelines to improve the validity and reproducibility of benchmarking. Improving the comparability is out of scope for this study.

We propose to enhance current best practices for device benchmarking in three ways: *(i)* extended latency reporting, *(ii)* additional test traffic patterns, *(iii)* a fixed set of tests per device class, and *(iv)* automated configuration. These proposals are specifically important for software devices, however also apply to hardware devices.

Meaningful Latency As we have argued in Section 3.3, the average latency should not be the only metric for latency. Latency can be compared visually through a histogram or cumulative distribution function (CDF). However, a complex reporting format is not preferred for a benchmark as this report cannot be translated into a few KPI numbers [19]. We propose to add a further KPIs for latency: 25th, 50th, 75th, 95th, 99th, and the 99.9th percentile enrich the report with information about the latency distribution and still keep reporting comparable.

Test Traffic Patterns Section 3.5 shows that the traffic pattern, consisting of packet inter arrival time and packet contents, has an influence on the behavior of a DuT. We propose to extend tests based on constant-bitrate traffic to also run with Poisson traffic as it approximates real world traffic more closely than CBR traffic but can still be generated by hardware and software load generators. For meaningful performance comparison, the traffic should be based on multiple flows to test multi-core scaling of software forwarding devices. We propose to conduct throughput benchmarks with both a single flow and randomized traffic and report results for both experiments. This provides insight into the behavior under different traffic and is an important metric for multi-core scaling of software forwarding devices.

Functionality-dependent tests Benchmarks need to be adapted to reflect suitable traffic and device configurations for a tested functionality such as switching, routing, or packet filtering. In this study, we showcase routing as one class of network interconnect functionality. The key processing step for software routers is the lookup in the FIB, which is required for each packet. Software routers have characteristic FIB lookup times, depending on the underlying data structures that do not always follow a constant, linear, or logarithmic distribution. While RFC 3222 [38] defines terminology for benchmarking software routers, specific guidelines for router benchmarking never made it beyond draft status in the IETF. We propose, in dependence of the existing metrics for switch forwarding tables [33, 24], to include FIB-dependent performance benchmarking results.

Automation Automating the configuration (and reconfiguration between test cycles) of tested devices removes error-prone manual device configuration and simplifies the reproducibility of benchmarks. We, therefore, propose that benchmarks should come with a setup script that establishes the benchmarked state in the tested device. In case of software-based packet processing we furthermore require the OS image and a detailed specification of the underlying hardware.

5. CONCLUSION

We provide arguments for reconsidering device benchmarking methodology for routers that has been valid for more than a decade. Our discussion shows that more detailed latency measurements and more realistic traffic patterns can describe device behavior more realistically. Further, functionality specific tests also can add valuable information. Documenting and automating configuration of test devices ensures reproducibility and minimizes misconfiguration.

Although the proposed changes make benchmark tools more complex, a solution purely relying on inexpensive commodity hardware is possible. As part of ongoing work, we provide a preliminary RFC 2544 benchmarking tool based on the packet generator MoonGen including the extended latency measurements. Furthermore, we release the full benchmark reports of the three investigated routers [0].

Acknowledgment

The authors would like to thank Marcel Sütterlin, Matthias Holdorf, Dominik Scholz, Alexander Mildner, and Simon Gallitscher. This paper is supported by the BMBF projects SURF (16KIS0145) and SENDATE-Planets (KIS2ITS001).

MoonGen RFC 2544 Benchmark Reports

[0] http://net.in.tum.de/pub/router-benchmarking/

6. REFERENCES

[1] DPDK. http://dpdk.org/. (visited 16/20/05).

[2] Ixia IxNetwork QuickTest, 2014.

[3] R. Asati, C. Pignataro, F. Calabria, and C. Olvera. Device Reset Characterization. RFC 6201, 2011.

[4] A. Beifuß, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle. A Study of Networking Software Induced Latency. In *NetSys*, Cottbus, Germany, 2015.

[5] BMWG history. https://datatracker.ietf.org/doc/charter-ietf-bmwg/history/. (visited 16/20/05).

[6] R. Bolla and R. Bruschi. Linux Software Router: Data Plane Optimization and Performance Evaluation. *JNW*, 2(3), 2007.

[7] A. Botta, A. Dainotti, and A. Pescapé. Do You Trust Your Software-Based Traffic Generator? *IEEE ComMag*, 48(9), 2010.

[8] S. Bradner. Benchmarking Terminology for Network Interconnection Devices. RFC 1242, 1991.

[9] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. RFC 2544, 1999.

[10] Candela. LANforge RFC Support. http://www.candelatech.com/rfcs.php. (visited 16/20/05).

[11] Cisco. Cisco Nexus 3548 Switch Performance Validation, 2012.

[12] G. A. Covington, G. Gibb, J. W. Lockwood, and N. McKeown. A Packet Generator on the NetFPGA Platform. In *IEEE FCCM*, Napa, USA, 2009.

[13] P. Emmerich, S. Gallenmüller, and G. Carle. FLOWer – Device Benchmarking Beyond 100 Gbit/s. Vienna, Austria, 2016.

[14] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *ACM SIGCOMM IMC*, Tokyo, Japan, 2015.

[15] P. Emmerich, D. Raumer, A. Beifuß, L. Erlacher, F. Wohlfart, T. M. Runge, S. Gallenmüller, and G. Carle. Optimizing Latency and CPU Load in Packet Processing Systems. In *SPECTS*, Chicago, USA, 2015.

[16] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. Performance Characteristics of Virtual Switching. In *IEEE CloudNet*, Luxembourg, 2014.

[17] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems. In *ICN*, Barcelona, Spain, 2015.

[18] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle. Comparison of Frameworks for High-Performance Packet IO. In *ACM/IEEE ANCS*, Santa Clara, USA, 2015.

[19] M. Georgescu, A. Morton, S. Fernandes, and P. Emmerich. Mean vs Median. Discussion on the IETF BMWG mailing list, archive available online at https://mailarchive.ietf.org/arch/, Nov. 2015.

[20] M. Ghobadi, G. Salmon, Y. Ganjali, M. Labrecque, and J. G. Steffan. Caliper: Precise and responsive traffic generator. In *IEEE HOTI*, Santa Clara, USA, 2012.

[21] iPerf. http://iperf.fr/. (visited 16/20/05).

[22] ITU-T. Y.1564: Ethernet service activation test methodology Recommendation, 2011.

[23] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans.Netw.*, 2(1), 1994.

[24] R. Mandeville and J. Perser. Benchmarking Methodology for LAN Switching Devices, Address caching capacity. RFC 2889, 2000.

[25] Metro Ethernet Forum. MEF 14: Abstract Test Suite for Traffic Management Phase 1, 2005.

[26] NetFPGA. http://netfpga.org/. (visited 16/20/05).

[27] ntop. PF_RING ZC (Zero Copy). http://www.ntop.org/products/pf_ring/pf_ring-zc-zero-copy/. (visited 16/16/05).

[28] ntop. Whitepaper: RFC-2544 performance test - PF_Ring-DNA VS Standard network Driver, 2012.

[29] Pktgen-DPDK. http://github.com/Pktgen/Pktgen-DPDK/.

[30] C. Popoviciu, A. Hamza, G. V. de Velde, and D. Dugatkin. IPv6 Benchmarking Methodology for Network Interconnect Devices. RFC 5180, 2008.

[31] L. Rizzo. netmap: A Novel Framework for Fast Packet I/O. In *USENIX ATC*, Boston, USA, 2012.

[32] T. G. Robertazzi. *Computer networks and systems: queueing theory and performance evaluation, 3rd edtion, Chapter 7.6: Self-Similar Traffic*. Springer, 2000.

[33] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. Oflops: An Open Framework for OpenFlow Switch Evaluation. In *PAM*, Vienna, Austria, 2012.

[34] SnabbCo. Snabb. https://github.com/SnabbCo/snabb. (visited 16/13/05).

[35] J. Sommers and P. Barford. Self-Configuring Network Traffic Generation. In *ACM SIGCOMM IMC*, Taormina, Italy, 2004.

[36] Spirent. Whitepaper: Spirent Testcenter RFC 2544 Benchmarking Test Package, 2010.

[37] The European Advanced Networking Test Center (EANTC). Whitepaper: Huawei technologies service activation using rfc 2544 tests, 2008.

[38] G. Trotter. Terminology for Forwarding Information Base (FIB) based Router Performance. RFC 3222, 2001.

[39] J. Whiteaker, F. Schneider, and R. Teixeira. Explaining packet delays under virtualization. *ACM SIGCOMM CCR*, 41, 2011.

[40] Xena Networks. Whitepaper: Xena2544 RFC2544 testing made easy, 2015.