Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems

Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle

Technische Universität München, Department of Computer Science, Network Architectures and Services
Boltzmannstr. 3, 85748 Garching bei München, Germany
{emmericp|raumer|wohlfart|carle}@net.in.tum.de

***Abstract*—Due to grown capabilities of commodity hardware for packet processing and the high flexibility of software, the use of those systems as alternatives to expensive dedicated networking devices has gained momentum. However, the performance of such PC-based software systems is still low when compared to specialized hardware. In this paper, we analyze the performance of several packet forwarding systems and identify bottlenecks by using profiling techniques. We show that the packet IO in the operating system's network stack is a significant bottleneck and that a six-fold performance increase can be achieved with user space networking frameworks like Intel DPDK.**

Keywords–Linux Router; Intel DPDK; Performance Evaluation; Measurement.

I. Introduction

Software routers and switches which are based on commodity hardware provide high flexibility. The user can combine modules without paying for unnecessary features. Software switches hosted on a single server allow for switching between virtual machines above the physical limit of its 10 GbE network adapters [1]. Additionally, almost any middle box behavior can be added to a software switch. Whole operating systems like the Vyatta Open-Firmware-Router [2] which focus on packet processing on commodity hardware have been created as part of new business models demonstrating the marketability of software routers and switches.

We analyze the performance of Linux IP forwarding, Linux bridge, Open vSwitch (OvS) [3], DPDK L2FWD (a forwarding application based on the user space packet processing system DPDK [4]), and DPDK vSwitch [5], a port of OvS that uses DPDK. We focus our measurements on OvS because it is the latest and fastest forwarding method based on the Linux network stack and the existence of the DPDK port allows for a direct performance comparison of the Linux network stack with DPDK. Thus we can show where potentially unnecessary bottlenecks in kernel-based packet processing systems are.

The throughput of DPDK-based software is significantly faster than the kernel forwarding techniques. We use profiling techniques to understand why the kernel applications are slower. We analyze hardware bottlenecks like effects of the CPU cache and software bottlenecks in the applications and the kernel. Based on these results we conclude that the most important bottleneck is receiving and sending packets in the network stack and that a six-fold performance improvement for OvS can be achieved by replacing the I/O technique with DPDK or a similar framework like Netmap [6] or PF_RING DNA [7].

We begin with a description of our test setup in Section II. Section III presents the results of our throughput tests. Section IV discusses potential hardware bottlenecks and Section V software bottlenecks. We discuss related work in Section VI and conclude with an outlook.

II. Test Methodology

Our test setup in Fig. 1 is based on recommendations by RFC 2544 [8].

A. Hardware Setup

Servers A and B are used as load generators and packet counters, the DuT (Device under Test) runs the software under test. For black-box tests, we must not introduce any overhead on the DuT through measurements. So we measure the offered load and the throughput on A and B. The DuT runs the Linux tool `perf` for white-box tests; this overhead reduces the maximum throughput by $\sim 1\%$.

The DuT uses an Intel X520-SR2 dual 10 GbE network interface card (NIC), the two other servers are equipped with X520-SR1 single port NICs. These NICs are based on the Intel 82599 Ethernet controller. All servers use 3.3 GHz Intel Xeon E3-1230 V2 CPUs. We also tested a second setup in which we replaced the X520 NICs with newer Intel X540 NICs to test effects of different hardware. We disabled Hyper-Threading, Turbo Boost, and power saving features that scale the frequency with the CPU load because we observed measurement artifacts with these features.

B. Software Setup

As generation of 64 B packets at 10 GbE line rate is a task that many existing packet generators are incapable of, we used a modified version of the `pfsend` packet generator from the PF_RING DNA [7] software repository [9]. This packet generator is able to produce UDP packets at the 10 GbE line rate of 14.88 Mpps with only a single core. Except for the DPDK forwarding test, all tests just used unidirectional traffic because the line rate was not the bottleneck.

We restrict the tests to a single flow and CPU core, because we observed linear scaling with the number of available CPU

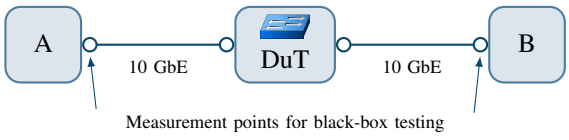

Figure 1. Server setup

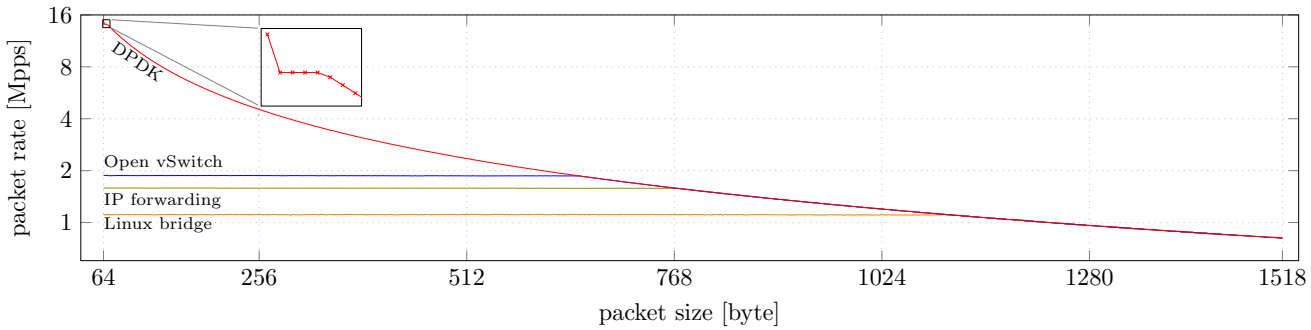

Figure 2. Packet size vs. throughput (logarithmic y-axis)

cores in previous work [10]. This focus on packet processing per core allows us to make claims for systems with different numbers of cores.

We used our own packet counter that relies on the statistics registers of our NICs which are accessible via `ethtool`. The packet rate is calculated by snapshotting the NIC counters periodically.

The DuT runs the Debian-based live Linux distribution Grml with a 3.7 kernel, the `ixgbe` 3.14.5 NIC driver with interrupts statically assigned to CPU cores, OvS 2.0.0, DPDK vSwitch 0.10 (based on OvS 2.0.0), and Intel DPDK 1.6.0.

C. Presentation of Results

All throughput measurements were run for 30 seconds and the packet rate was sampled every 100 ms. Graphs show the average measurement. The standard deviation was below 0.2% for all throughput measurements. We therefore omitted error bars in these. Profiling measurements were restricted to the core on which the processing task was pinned and were run for five minutes per test to get accurate results. Measurements showing significant noise were plotted with 95% confidence intervals (cf. Fig. 4).

III. Forwarding Performance

Table I compares the data plane performance of OvS, Linux IP forwarding, Linux bridge, DPDK vSwitch, and DPDK L2FWD with minimally sized packets. The packet size is irrelevant in almost all scenarios as shown in Fig. 2.

TABLE I. Data plane performance comparison

Application	Throughput [Mpps]
DPDK L2FWD bidir X540	29.76
DPDK L2FWD bidir X520	24.06
DPDK L2FWD unidir X520	14.88
DPDK vSwitch	11.31
Open vSwitch	1.88
Linux IP forwarding	1.58
Linux bridge	1.11

The OvS kernel module is able to process packets faster than the Linux kernel forwarding. The Linux kernel code for routing has received steady optimizations while the bridging code was last modified with kernel 2.6. OvS proved to be the fastest packet forwarding technique using the Linux network stack.

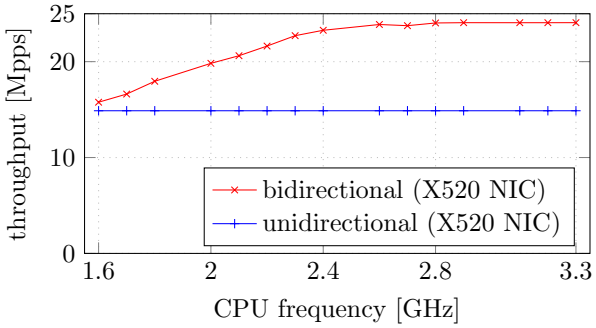

Figure 3. L2FWD at different clock rates

The DPDK applications do not use the Linux network stack and are significantly faster. The DPDK port of OvS showed a six-fold performance increase compared to the kernel version.

A. User Space Packet Processing

Approaches like Intel DPDK [4], Netmap [6], and PF_RING DNA [7] replace the network stack with a user space application to avoid overhead.

DPDK L2FWD only forwards packets between two statically configured network interfaces without consulting a routing or flow table. It can therefore be seen as an upper bound for the possible throughput. We focus our measurements on DPDK here but also observed similar results with forwarding applications based on Netmap and PF_RING.

We adjusted the CPU clock frequency to measure the required processing power per packet. DPDK L2FWD managed to forward 14.88 Mpps even with the lowest possible frequency, we therefore added a bidirectional test for this application. Fig. 3 shows the throughput with all clock frequencies supported by our CPU. The same test with similar results for the performance of Netmap is presented by Rizzo in [6].

The bidirectional test initially only achieved a throughput of 24.06 Mpps instead of line rate with the maximum clock frequency. We then tried to use two cores for this test but this resulted in the same performance which indicates a hardware limit in the NIC. We therefore replaced the X520 NIC with a newer X540 NIC on all servers to investigate further. The X540 was able to forward 29.76 Mpps, i.e., line rate, with DPDK on a single CPU core.

The DPDK L2FWD application initially only managed to forward 13.8 Mpps in the single direction test at the maximum

CPU frequency, a similar result can be found in [11]. Reducing the CPU frequency increased the throughput to the expected value of 14.88 Mpps. Our investigation of this anomaly revealed that the lack of any processing combined with the fast CPU caused DPDK to poll the NIC too often. DPDK does not use interrupts, it utilizes a busy wait loop that polls the NIC until at least one packet is returned. This resulted in a high poll rate which affected the throughput. We limited the poll rate to 500,000 poll operations per second (i.e., a batch size of about 30 packets) and achieved line rate in the unidirectional test with all frequencies. This effect was only observed with the X520 NIC, tests with X540 NICs did not show this anomaly.

IV. Hardware Bottlenecks

We use OvS as an example and follow a packet's path through it and examine each component for potential bottlenecks. A packet arrives at the input network interface and is transferred via DMA with Intel's Direct Cache Access (DCA) technology [12] to the L3 cache. It is then processed and modified by OvS on the CPU based on a flow table in the OvS kernel module called the *datapath*. Packets that do not match any rule in the datapath are forwarded to a user space process, which then installs a rule in the kernel module for subsequent packets of this flow, this processing path is called the slow path. These rules use an idle timeout so that only actively used rules are kept in the kernel module. Afterwards the packet is transferred to the outgoing network interface via DMA/DCA.

Other forwarding systems beside OvS use the same packet flow except for the processing step. The following potential bottlenecks are present in the packet processing path.

A. Network Bandwidth

DPDK L2FWD hit the limit of 14.88 Mpps in the unidirectional test, but not in the bidirectional test (cf. Table I). All other measured programs were far below this limit. It is therefore not a relevant bottleneck for our tests with a single CPU core.

B. NIC Processing Capacity

The data sheet of the Intel 82559 chip does not mention any limits to the packet rate [13]. However, we have encountered such a limit at 24 Mpps. We verified that the processing is limited by the number of packets per second and not the total bandwidth by testing with larger packet sizes. The NIC is able to handle line rate with packets larger than 100 Byte. The newer X540 chip does not have this limit.

C. PCIe Bandwidth

Both the X520 and X540 NICs we used are attached via a PCIe 2.0 x8 link with a net bandwidth of 32 GBit/s per direction, far more than the 20 GBit/s of the two network ports. Our CPU unfortunately does not support performance counters to measure this bandwidth. However, this limit is not relevant since the X540 NIC is able to sustain full line rate with 64 Byte packets on both ports.

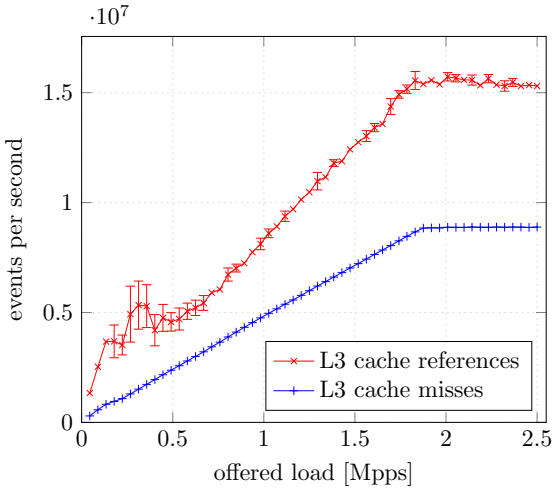

Figure 4. L3 cache statistics (OvS)

D. Memory Bandwidth

Each packet is written to and read from the main memory at least once. The CPU in our test server offers a memory bandwidth of 200 GBit/s. This is therefore not a bottleneck for 10 GBit networks but needs to be considered when moving to 100 GBit Ethernet.

E. CPU Cache Size

The overall cache size can be a bottleneck if it is insufficient for the state of the application.

We use OvS as example here, but the results are also applicable to other forwarding applications, which use a flow table or a routing table. Fig. 4 shows the number of L3 cache references and misses in the OvS forwarding scenario with only one flow. Both grow linearly with the number of processed packets per second, the miss ratio stays constant. Slight deviations in the lower packet rates are due to the dynamic interrupt rate throttling by the ixgbe driver. The other cache levels show similar results.

The total number of accesses and misses per second is in the order of 10^7. This translates to a cache and memory bandwidth of less than 8 GBit per second when multiplied with the CPU cache line size of 64 Bytes. This is an uncritically low bandwidth requirement that can easily be satisfied [14].

A high number of actively used flow table entries, which are 576 Bytes each, in the OvS forwarding scenario can exhaust the cache. The L1 cache fits 56 entries, the L2 cache 455, and the L3 Cache about 14 500 without taking space requirements for packets or any other required data into account.

Fig. 5 shows the number of active flows vs. cache misses and throughput. The first two caches quickly fill up and cause a slight drop in the performance from 1.87 Mpps with one flow (slightly lower than the result from Table I due to active profiling) to 1.76 Mpps with 2000 flows.

Tests that exhaust the L3 cache require more than 14 500 flows, but testing with such a large amount of flow table entries was not feasible due to exponential growth of the time required to add flows. This exponential growth of flow table

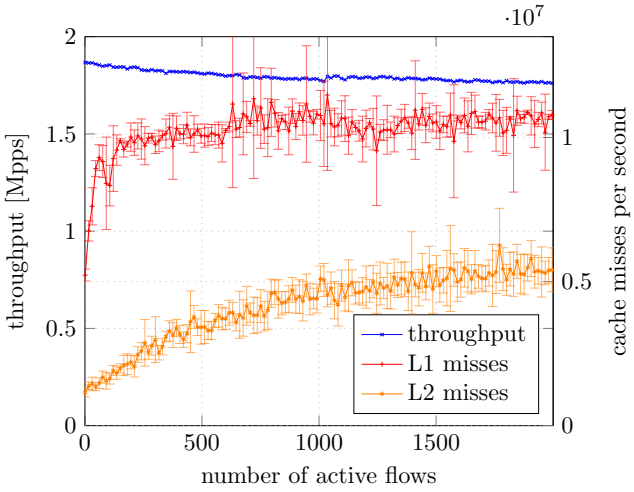

Figure 5. Flow table size vs. cache misses (OvS)

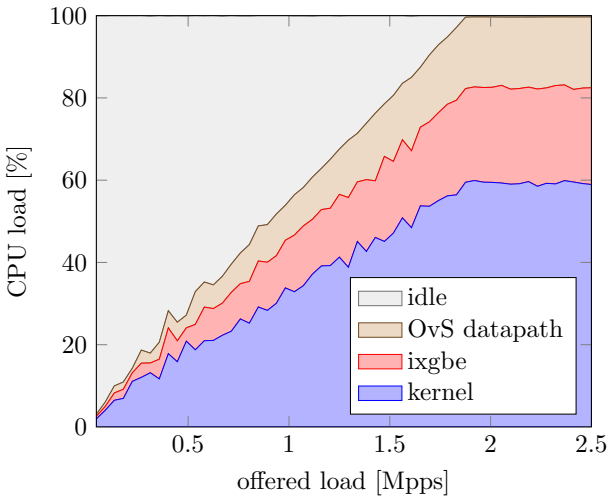

Figure 6. CPU usage per kernel module (OvS)

modification operations in OvS is described in more detail by Rotsos et al. [15]. Note that this is not a bottleneck in a real-world application because OvS only needs a constant amount of flow table entries (exact figure depends on the installed OpenFlow rules) per attached device due to wild card support.

Minor performance improvements can be achieved by reducing the footprint of flow or routing table entries. But this is not a major bottleneck and does not explain the performance gap to DPDK-based applications.

F. CPU Cache Line Length

The cache line length can also affect the throughput due to access latencies and bad alignment when using packet sizes that are not a multiple of the cache line length of 64 Bytes [14]. To investigate we performed a test series of maximum throughput experiments and we increased the packet size by 1 Byte per test (cf. Fig. 2).

The DPDK L2FWD throughput test showed a very slight deviation from the line rate for packet sizes between 65 and 68 Bytes, which were processed with only 14.12 Mpps instead of the expected 14.71 to 14.20 Mpps (line rate). Larger packets are limited by the line rate. We assume this is caused by packets that need slightly more than a single cache line. Rizzo measured this effect with Netmap in more detail in [6].

The maximum throughput curves of the conventional packet processing systems have no inflection points except when the network link bandwidth sets in. We conclude that there are no adverse effects if the packet sizes do not match the cache line size for the kernel-based systems which we are trying to improve.

G. CPU Time

We measured the throughput of all packet forwarding programs with different CPU frequencies to analyze the impact of raw processing power. All applications scaled linearly with the CPU frequency except for the DPDK L2FWD application (cf. Section III-A). This means that the only relevant bottleneck for the kernel-based forwarding applications is the software.

V. Software Bottlenecks

We ran the Linux profiling tool `perf` to analyze the relative time required per function and combined this with the CPU cycle counter to compute the wall-clock time of each function.

A. CPU Utilization per Processing Step

Fig. 6 shows the CPU time aggregated per kernel module in the OvS forwarding scenario under increasing load. It shows the self-time spent in the respective kernel module, i.e., without taking the call stack into account. The load of all involved modules increases linearly and the first drops were observed once the CPU load hit 100%. This example uses OvS because its architecture as a separate kernel module allows for a simple split by functionality, the other kernel-based forwarding applications show a similar behavior. DPDK utilizes a busy wait loop and the CPU load is always 100% independent of the offered load.

The OvS datapath module is responsible for the forwarding decision, the other modules handle all other tasks: receiving packets, sending packets, and buffer management. Only about 18% of the CPU time is spent in the OvS kernel module, so the software bottleneck is packet I/O and not processing.

The datapath module calls back into other kernel modules during the processing at one point to acquire and release a spin lock which should be attributed to OvS even though the time is spent in another kernel module. Measuring this requires a kernel that is compiled without the `fomit-frame-pointers` compiler optimization to generate backtraces based on the stack from a sample. This change resulted in a drop of the throughput by $\sim 15\%$. This additional overhead is distributed approximately uniformly across the code which we confirmed by comparing the self-times with and without the compiler optimization. The option allows for a more detailed and correct look at the CPU time distribution: about 4% of the total time is spent in the spin lock with OvS as caller, so 22% of the time is spent processing the packet and 78% are for receiving and sending. All of the following measurements were run without this compiler optimization.

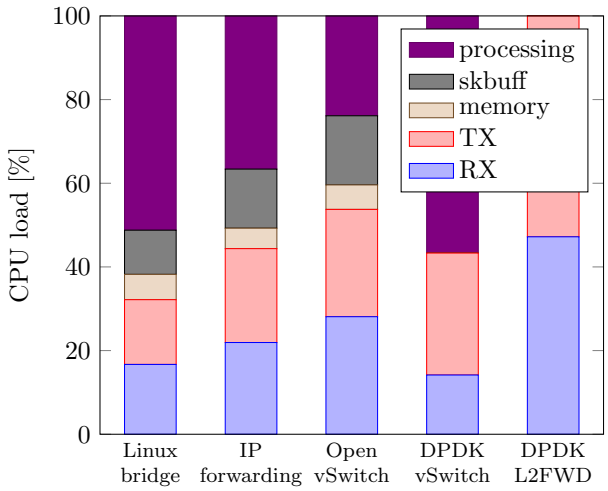

Figure 7. CPU usage per task under full load

Fig. 7 shows the relative required time for the required steps of the forwarding methods: `Processing` are all functions that are associated with reaching a forwarding decision, so it is 0% for the DPDK L2FWD application which only transfers packets between two interfaces. `Memory` represents the time required to allocate and free the memory for the packets, but without any initialization functions that are specific to the kernel's skbuff data structure. `Skbuff` contains all functions that initialize, release, or change an skbuff (kernel packet buffer descriptor) without the memory management functions. `RX` includes all functions that handle receiving a packet in the network stack and driver. It also includes time required to handle interrupts like saving and restoring the context or raising the software interrupt to call into `softirqd` for further processing. `TX` includes all functions from the point where the application calls into the network stack's `dev_queue_xmit` transmit function. It includes any task in the network stack that is required to send a packet, most notably the handling of the queuing discipline in the kernel.

Table II shows the same data as Fig. 7 in CPU cycles per packet. The Linux bridge clones the skbuff descriptor unnecessarily which leads to higher memory management and skbuff times. The differences between the I/O tasks for the DPDK applications are due to different batch sizes which were chosen for optimal overall performance with the applications.

TABLE II. CPU CYCLES PER PACKET AND PROCESSING STEP

Application	RX	TX	Mem.	skb	Process.
Linux bridge	491	456	178	311	1508
IP forward	453	463	101	292	757
Open vSwitch	490	447	102	288	416
DPDK vSwitch	41	85	0	0	165
DPDK L2FWD	55	62	0	0	0

The linux kernel needs about 1300 cycles on average to receive and send a single packet including memory management and overhead from the data structures. DPDK performs the same tasks with only 120 cycles, so the network stack is an important bottleneck for packet forwarding systems.

B. Overheads in the Kernel

The Linux network stack is a general-purpose network stack and exhibits unnecessary overheads when used with a specialized application like a packet forwarding system. One example for such an overhead in the kernel is the send path which requires a spin lock to access the queuing discipline of the device (configured as the default fifo queue). This lock takes about 8% of the total wall-clock time in the OvS example which impacts the throughput. Profiling shows that this lock is never contended because there are no other network tasks which could acquire the same lock. However, the kernel needs to handle the generic case which might be configured differently, so the spin lock still needs to be acquired and released at multiple locations, this manifests as overhead.

Another example is the skbuff data structure. It needs to handle all possible cases for the network stack and requires an extensive constructor and destructor. A simple buffer is sufficient for a software switch or router.

A switch can allocate a fixed amount of buffers on startup and directly reuse them after finishing a batch of packets. The kernel's network stack uses a memory pool from which buffers are acquired before retrieving packets and returned to after sending packets. In the general case, this is necessary because packets might still be needed after processing them. However, a software switch can always forward all packets directly. So the same buffers can be overwritten with new packets immediately after sending a batch. Memory management is therefore also unnecessary.

C. Open vSwitch vs. DPDK vSwitch

DPDK vSwitch shows a six-fold increase in throughput over OvS, this discrepancy can not explained by only the packet I/O. The processing logic is also optimized, DPDK vSwitch only takes 165 cycles to reach a forwarding decision for a packet while the original OvS requires 416 cycles.

DPDK vSwitch replaces the whole flow table with a highly optimized version. Some of these optimizations, like an optimized hash function that utilizes the CPU's CRC32 instruction, could be ported back to the original OvS. This would improve the performance slightly, but the network stack is still the main bottleneck.

VI. Related Work

Bolla and Bruschi recognized the trend for optimization already in 2007 and presented a detailed study of packet forwarding in Linux by applying RFC 2544 and internal measurements with profiling [16]. They measured a higher CPU load of packet I/O tasks in test similar to the one described in Section V here. This indicates that the overhead was reduced since kernel version 2.6.16 which they used. Later a study of performance influences of multi-core PC systems under different workloads [17] was published. These performance studies have been shown and extended, e.g., in [10]. We only discuss bottlenecks of software routers here, further measurements on Open vSwitch throughput and latency in different scenarios can be found in [18] and [19].

Another important aspect beside the throughput is the latency of a forwarding system. A notable example of latency

measurements is the OFLOPS OpenFlow benchmarking framework by Rotsos et al. which uses OvS as an example, they also describe challenges with measuring latency on commodity hardware [15]. Discussion of latency in software routers can also be found in [20] and [21]. We will extend this study to include latency measurements based on our packet generator MoonGen [22] in future work.

Other user space packet processing frameworks beside Intel's DPDK with similar performance characteristics are Netmap by Rizzo who presents similar measurements to our tests from Section III-A for his framework [6]. This shows that DPDK and Netmap have similar performance characteristics and our results are transferable to Netmap. Deri presents PF_RING DNA in [7] which was originally written for a fast packet capture application called nCap. We expect similar performance gains when using this framework.

DPDK was also used to evaluate the performance of a PC-based OpenFlow switch by Pongracz et al. [11] but without comparing it to a conventional packet I/O system like the Linux kernel.

Rizzo et al. ported the networking library libpcap to Netmap and use it to improve applications transparently with a user space version of OvS as one example [23]. They only state the performance improvements for various applications but do not measure software bottlenecks explicitly.

VII. Conclusions and Outlook

We identified and measured different bottlenecks and conclude that the main bottleneck for PC-based packet forwarding systems is the software due to overhead in the kernel's network stack. We measured that receiving and sending packets with a user space packet processing framework like DPDK is 12 times faster than using the Linux kernel to do the same task. Existing software switches like OvS can show significant performance improvements by adopting a modern packet I/O framework even without modifying the processing logic. DPDK vSwitch also optimizes the processing logic and shows a six-fold increase of the total performance compared to OvS.

There have been discussions to include the Netmap framework in the Linux kernel [24] where it could supplement the current network API while maintaining backwards compatibility with older drivers and applications. Using such a framework requires special drivers and a complicated setup procedure at the moment. Direct support from the Linux kernel is an important step for the mainstream adaption of such frameworks in the next generation of software switches and routers.

Acknowledgments

This research has been supported by the DFG as part of the MEMPHIS project (CA 595/5-2), the KIC EIT ICT Labs on SDN, and the BMBF under EUREKA-Project SASER (01BP12300A).

References

[1] L. Rizzo and G. Lettieri, "VALE, a Switched Ethernet for Virtual Machines," in Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies. ACM, 2012, pp. 61–72.

[2] "Vyatta," http://www.brocade.com/products/all/network-functions-virtualization/product-details/5400-vrouter/index.page, last visited 2015-03-08.

[3] "Open vSwitch," http://openvswitch.org, last visited 2015-03-08.

[4] "Intel DPDK: Data Plane Development Kit," http://dpdk.org/, Intel, last visited 2015-03-08.

[5] "Intel DPDK vSwitch," https://github.com/01org/dpdk-ovs, Intel Corporation, last visited 2015-03-08.

[6] L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," in 2012 USENIX Annual Technical Conference (USENIX ATC 12). Boston, MA: USENIX, 2012, pp. 101–112.

[7] L. Deri, "nCap: Wire-speed Packet Capture and Transmission," in IEEE Workshop on End-to-End Monitoring Techniques and Services, 2005, pp. 47–55.

[8] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544 (Informational), Internet Engineering Task Force, March 1999.

[9] "Ntop," http://www.ntop.org, last visited 2015-03-08.

[10] T. Meyer, F. Wohlfart, D. Raumer, B. Wolfinger, and G. Carle, "Validated Model-Based Prediction of Multi-Core Software Router Performance," Praxis der Informationsverarbeitung und Kommunikation (PIK), April 2014.

[11] G. Pongracz, L. Molnar, and Z. L. Kis, "Removing Roadblocks from SDN: OpenFlow Software Switch Performance on Intel DPDK," Second European Workshop on Software Defined Networks (EWSDN'13), 2013, pp. 62–67.

[12] R. Huggahalli, R. Iyer, and S. Tetrick, "Direct Cache Access for High Bandwidth Network I/O," in Proceedings of the 32nd Annual International Symposium on Computer Architecture, 2005, pp. 50–59.

[13] "Intel 82599 10 GbE Controller Datasheet Rev. 2.76," Intel, 2012, Santa Clara, USA.

[14] "Intel® 64 and IA-32 Architectures Optimization Reference Manual," Intel, 2014.

[15] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An Open Framework for OpenFlow Switch Evaluation," in Passive and Active Measurement. Springer, 2012, pp. 85–95.

[16] R. Bolla and R. Bruschi, "Linux Software Router: Data Plane Optimization and Performance Evaluation," Journal of Networks, vol. 2, no. 3, June 2007, pp. 6–17.

[17] M. Dobrescu, K. Argyraki, and S. Ratnasamy, "Toward Predictable Performance in Software Packet-Processing Platforms," in USENIX Conference on Networked Systems Design and Implementation (NSDI), April 2012.

[18] 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 2nd International Conference on Networked Systems 2015 (accepted), Cottbus, Germany, 2015.

[19] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance Characteristics of Virtual Switching," in 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), Luxembourg, 2014.

[20] L. Angrisani, G. Ventre, L. Peluso, and A. Tedesco, "Measurement of Processing and Queuing Delays Introduced by an Open-Source Router in a Single-Hop Network," IEEE Transactions on Instrumentation and Measurement, vol. 55, no. 4, 2006, pp. 1065–1076.

[21] S. Larsen, P. Sarangam, R. Huggahalli, and S. Kulkarni, "Architectural Breakdown of End-to-End Latency in a TCP/IP Network," International Journal of Parallel Programming, vol. 37, no. 6, 2009, pp. 556–571.

[22] P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," ArXiv e-prints, Oct. 2014. [Online]. Available: http://adsabs.harvard.edu/abs/2014arXiv1410.3322E

[23] L. Rizzo, M. Carbone, and G. Catalli, "Transparent Acceleration of Software Packet Forwarding using Netmap," in INFOCOM, 2012 Proceedings IEEE. IEEE, 2012, pp. 2471–2479.

[24] S. Hemminger, "netmap: infrastructure (in staging)," http://lwn.net/Articles/548077/, last visited 2015-03-08.