A Study of Network Stack Latency for Game Servers

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

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

***Abstract*—Latency has a high impact on the user satisfaction in real-time multiplayer games. The network between the client and server is, in most cases, the main cause for latency and out of the scope of the game's developer. But the developer should avoid introducing unnecessary delays within his responsibility; i.e. with respect to development and operating costs he should wisely choose a network stack and deployment model for the server in order to reduce latency. In this paper we present latency measurements of the Linux network stack and effects of virtualization. We show that a worst-case scenario can cause a latency in the millisecond range when running a game server. We discuss how latencies can be reduced by using the packet processing framework DPDK to replace the operating system's network stack.**

***Keywords*—latency, measurement, virtualization, Intel DPDK**

I. Introduction

Real-time multiplayer games require packet processing with low latency in order to provide a good gaming experience. The biggest share of delay often comes from the network connection between the server and the player and is out of scope for game developers. This network latency can be in the order of tens of milliseconds for Internet connections. Therefore a micro-optimization of network stack latency is often not considered worth the effort by developers. In this paper, we show that the network stack can cause a latency in the millisecond range on the server in edge cases. We show how to avoid these cases and what should be considered when designing a game server.

Another important scenario are players connected via local area network (LAN) to the game server. This is especially relevant for electronic sports events that allow players to compete in real-time games with minimal latency. The network latency here is minimal and the impact of the network stack's latency therefore of greater importance. We show that the latency caused by the network stack can be in the range of hundreds of microseconds under normal operating conditions. This is a significant share of the total latency in this scenario. The server's developers must take the network stack into account when designing a game server for this use case.

In this paper, we use a server application that effectively emulates a game server and measure its latency under different conditions. Excessive use of buffers can cause large latencies due to a problem known as buffer bloat [1]. We discuss how this problem applies to game servers and how a worst-case scenario, e.g. by a malicious or bugged client, can increase the latency by several milliseconds and how to avoid this worst-case latency. We also investigate the effects of deploying a game server on a virtual machine (VM) in a cloud environment. We then discuss advantages of game servers that bypass the operating system's network stack in order to provide maximum throughput performance and minimal latency.

978-1-4799-6882-4/14/$31.00 © 2014 IEEE

This paper is structured as follows: In Section II we look at the theoretical background of packet processing on servers with an emphasis on latency. Section III presents related work. We describe our methodology and test setup in Section IV. Our measurements and latency comparisons are presented in Section V. We then conclude with a summary of our results in Section VI.

II. Packet Processing in Linux

Off-the-shelf hardware has received features to cope with the growing number of cores and network speed. For example, modern network interface cards (NICs) can copy incoming packets directly to the CPU's cache instead of going through the memory. This improves both the throughput and the latency of servers [2]. This technique is called direct cache access (DCA), an extension of direct memory access (DMA).

The network stack has also been improved to take advantage of these new paradigms in Linux: The New API (NAPI), introduced in Linux kernel 2.4.20, is designed to mitigate performance problems with high packet rates. It first introduced throttling of interrupt rates and polling the NIC for new packets to reduce the large per-packet overhead of interrupts. This optimization is a trade-off between throughput and latency which is increased by the reduced number of interrupts. [3]

We follow a packet's path through a modern Linux-based game server and analyze the impact on the latency of each component and processing step.

A. Schematic View of Network Stacks for Game Servers

Figure 1 schematically visualizes the processing path of packet reception and the hard- and software that is involved.

Packets arriving at the ingoing interface are stored in the *Rx NIC buffer*. From there they are immediately transferred to the CPU's cache via DCA over the PCIe bus. This transfer is controlled by the DMA engine on the NIC and does not require intervention by the CPU. The CPU only needs to transfer DMA descriptors, pointers to memory locations to which the incoming packets are written, to the NIC. It is up to the driver

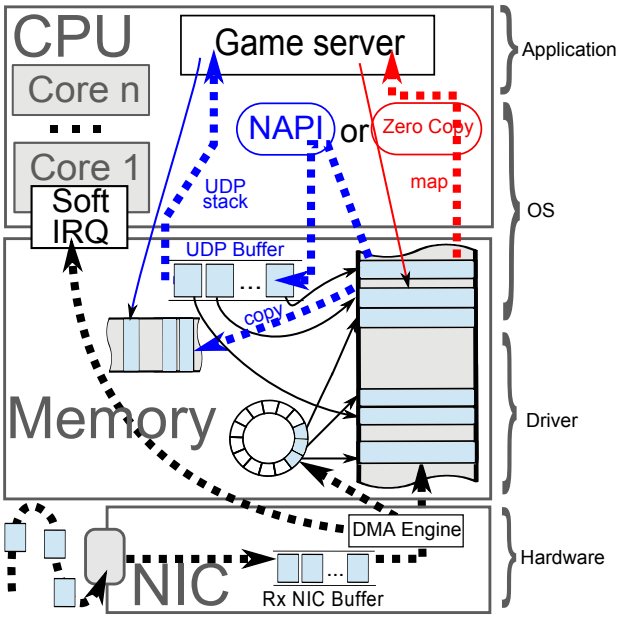

Fig. 1. Schematic view on RX packet processing and hardware resources

to ensure that the NIC always has enough descriptors available to transfer packets. The Intel X540 NIC used in our testbed drops incoming packets by default if there are no descriptors available instead of filling the Rx NIC buffer [4]. The size of the initial buffer on the NIC is therefore not relevant for the latency. Only the size of the DMA descriptor ring buffer affects the latency. This descriptor buffer has a default size of 512 entries with the `ixgbe` driver under Linux.

The NIC triggers an interrupt when the transfer is completed. The latency between the interrupt assertion and the call into the Soft IRQ interrupt handler is estimated as about 2 μs by Larsen et al. [5]. It is therefore not a significant source of latency for a game server if one interrupt is used per packet. Triggering an interrupt for each packet proved to be a bottleneck for high packet rates; modern drivers and NIC therefore moderate the maximum interrupt rate [3]. This interrupt throttle rate (ITR) increases the throughput at the cost of latency.

B. Transferring the Packet into the User Space

The packet now resides in the kernel space where it will now be processed by the OS and application. The server runs in the user space and the kernel needs to transfer it to the application.

1) Linux Network Stack: The application opens a socket on startup with an associated buffer space. It then uses the `recv` syscall on the socket, this call retrieves a packet from the buffer and copies it into a user space memory location. The Linux network stack copies the packet to the user space in order to make it available to the application. This path is highlighted with a dotted blue line in Figure 1.

The syscall involves a switch to kernel mode and back as well as a copy operation for each packet. This poses a significant overhead, limits the maximum throughput, and adds latency due to processing time. High speed applications like software routers therefore often run in the kernel context to avoid this overhead. This entails the risk of system crashes due to simple programming bugs and therefore requires careful development and extensive testing. It also complicates the development process due to additional challenges of running in the kernel. For example, regular adjustments for new kernel versions are needed because the interfaces may change between versions. This approach is therefore not suitable for complex applications like game servers.

2) Memory Mapping Frameworks: Memory mapping frameworks for packet I/O are a new approach to this problem. These frameworks provide a user space application direct access to both the NIC and the memory pages to which the packets are written. The packet's path is highlighted with a dotted red line Figure 1. This avoids the copy operation and context switch, these frameworks are therefore also referred to as zero copy frameworks [6]. The most popular frameworks that implement this are PF_RING ZC [6], netmap [7], and Intel DPDK [8]. We use DPDK for this paper because we found it to be the most pleasant to use. However, we expect similar results with the other frameworks because they all build on the same principles and make similar performance claims [6], [7], [8].

The frameworks build on modified drivers to bypass the system's network stack. They do not use interrupts and rely on polling the NIC instead, this reduces the per-packet costs. The application spends all idle time in a busy-wait loop that polls the NIC, so a packet is retrieved as soon as the application is ready to handle it. Note that this does not increase the overall latency: Even with interrupts, the packets still need to spend time in a queue until the application is ready to process them. The cost of a poll-operation is less than 80 cpu cycles for DPDK [8], i.e. 40 ns on the 2 GHz CPU used here. This is even faster than the 2 μs required to process an interrupt [5]. A major drawback is that the polling loop will always cause 100% CPU load. Sleeping between polls increases latency and is not recommended in DPDK, instead it offers APIs to control the CPUs clock frequency to alleviate the problem and conserve power [9].

These frameworks also process batches of multiple packets with a single call to further decreases the cost per packet. For a game server, a state update that needs to be sent to multiple players can be processed with a single API call while the Linux network stack requires multiple expensive system calls to achieve this.

The major drawback of these frameworks is that they offer only basic IO functionality compared to the network stack of an OS. They only handle receiving and sending of raw packets without any further processing. It is up to the application to implement protocols like IP and UDP. This simplistic approach offers an improved performance compared to a full-blown network stack as the programmer can choose which features are needed and implement them in an application-specific way.

C. Servers in the Cloud

Additional processing steps are required for the case of virtualized game servers. The virtual machine exposes a virtual NIC that needs to be connected to the physical NIC. Thus, the number of buffers that can introduce an undesired latency

increases: both the virtual NIC and its driver add additional buffers to the packet's path. The overall system load also increases: The virtual NIC that processes packets for the VM is emulated in software and the system also needs to run a virtual switch to connect the virtual NIC to the physical NIC.

D. Sending Packets

The server also needs to respond to incoming packets. This transmit path is similar to the receive path: The packet is first placed into a transmit buffer associated with the socket and copied from the user to the kernel space into a buffer associated with the socket. The driver manages a buffer of Tx DMA descriptors that are used to transfer the packet to the NIC.

III. Related Work

Rotsos et al. presented a framework for measuring the latency of OpenFlow switches [10]. They utilized an FPGA to acquire time stamps with a high precision and presented measurements of the software switch Open vSwitch running on Linux. They measured a latency of 35 μs (σ = 13 μs) for simple in-kernel forwarding under light load. This can be seen as a lower bound for the latencies of a game server that also involves user space processing.

Bolla and Bruschi applied RFC 2544 [11] to a Linux software router [12]. They used a dedicated network testing device to acquire the latency with microsecond accuracy. They measured delays from 14 μs to hundreds of μs depending on the load and configuration for packet forwarding in the kernel under normal operating conditions. Overloading the software router resulted in latencies in the millisecond range.

Whiteaker et.al. measured the impact of virtualization on the latency [13]. They observed a long tail distribution of latencies when packets are forwarded into a VM. They placed different workloads on the VM and measured the impact on latency. They also measured latencies in the millisecond range if the VM is put under a high network load. Their measurements are restricted to a 100 Mbit/s network due to hardware restrictions of their time stamping device.

Larsen et al. take a detailed look at the latency of TCP/IP traffic [5]. They provide a detailed breakdown of the latencies by the different processing steps of a server that answers a simple request. However, they do not put the system under notable load.

Our major contribution for this paper beyond the results in the related work is that we explicitly focus on game servers using UDP. The related work either measures in-kernel forwarding techniques or metrics like ICMP ping response times under load. We measure the latency of UDP packets that are processed by a user space application. We also include measurements of the memory mapping framework DPDK which circumvents the Linux network stack. Our test setup also improves beyond the related work as we use a custom load generator that utilizes hardware features of commodity NICs to obtain latency measurements. Our load generator is described in [14].

IV. Test Methodology and Setup

We ran all tests in our 10 GbE testbed with direct connections between the involved servers.

A. Hard- and Software

Our device under test (DuT) running the game server is equipped with an Intel Xeon E5-2640 v2 CPU clocked at 2 GHz. All features that scale the CPU frequency with the load, like power-saving features and Turbo Boost have been disabled as we observed measurement artifacts with these features. Hyper-threading was also disabled for the same reason.

The NIC is an Intel X540-T2 that is directly connected to the load generator which uses the same 10 GbE NIC. The DuT uses the Intel `ixgbe` driver in version 3.9.15 with both the Rx and Tx ring configured to the default of 512 DMA descriptors. NIC interrupts were restricted to a single CPU core.

The DuT and the virtual machine run a Debian-based live image with kernel 3.7. Virtual machines run on KVM 1.1.2 with VirtIO NICs and Open vSwitch 2.0 to connect them to the external network interface.

B. Load Generator and Sink

We use a custom load generator based on the packet processing framework DPDK for our measurements. This load generator is able to generate traffic at 10 GBit/s. We utilize the IEEE 1588 PTP [15] hardware time stamping features of the X540 NIC which can be used to time stamp virtually arbitrary UDP packets. The time stamped packets only require a few byte identification in the payload to be recognized as PTP packets by the NIC. The hardware is able to provide latency measurements with sub-microsecond precision [4]. We take time stamps of up to 500 packets per second. The timestamped packets are injected in the test traffic at randomized intervals. The DuT cannot distinguish them from other traffic as they only differ in payload which is ignored by the application. The packets are time stamped in hardware immediately before sending and after receiving them from the physical layer. We also use the hardware rate control feature of the X540 NIC [4] to ensure a constant inter-packet delay to generate smooth constant bitrate traffic. [14]

The load generator uses a dual-port NIC, one port is used to generate the test traffic and send it to the DuT. The second port, also attached to the DuT, measures the achieved throughput.

C. Model Game Server

Our latency measurements are based on a server application that emulates the behavior of a game server. The server opens a UDP socket and waits for packets from a client. All connected clients are kept in a list. For each received packet, the server waits 150 CPU cycles to emulate processing logic. The actual processing time causes only a very small part of the total delay; 150 clock cycles correspond to only 0.03 μs with the 2 GHz CPU in the DuT. The processing logic was only added to avoid possible artifacts caused by an unrealistic setup where a packet is sent back out immediately. Even a longer processing time of a real game-server only adds a constant latency and is dwarfed by other effects like buffering. Finally, the server sends a copy of the incoming packet to all clients in the list. The latency is defined as the time until the last client in the list receives a response.

We run tests with an incoming packet rate of 10 to 100 kpps (kilo packets per second) and 4 to 32 connected clients. All

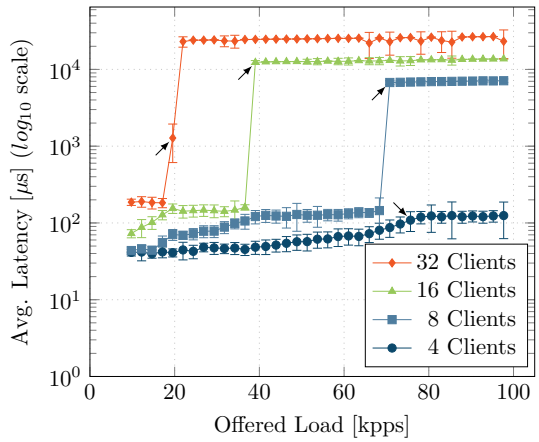

Fig. 2. Average latency under increasing load with a varying number of clients

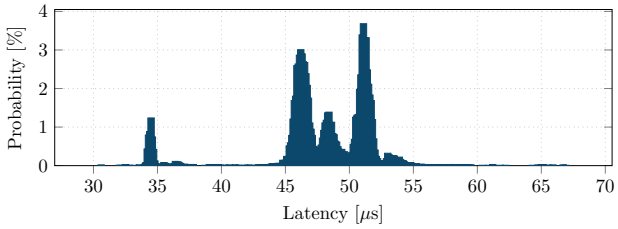

Fig. 3. Histogram of latencies for 4 clients, default buffer size, 30 kpps offered load

tests use a packet size of 128 byte to emulate a small state update packet, e.g. a player's position update. This achieves a high system load and avoids hitting the line rate limit for outgoing packets with a large number of clients.

This setup should be interpreted as a server with a large number of clients, e.g. for a massively multiplayer online game (MMO), that needs to send out state updates on average to 4 to 32 clients for each incoming packet. Another interpretation of the test scenario is a server that hosts multiple small games.

The server was restricted to a single CPU core as we have shown in previous work that packet processing can achieve linear scaling with the number of CPU cores [16].

D. Graphs

All graphed latencies are the average of at least 10 000 measurements unless mentioned otherwise. The error bars in the graphs show the standard deviation, they are omitted if they are smaller than the mark indicating the measurement point.

V. Evaluation

We evaluate the latency with our emulated game server running on Linux and in a virtualized environment. We then port the emulated game server to the packet processing framework DPDK that allows us to access the NIC directly without intervention from the OS.

A. Basic Latency Characteristics under Linux

Figure 2 shows how the average latency changes for a server running directly under Linux when the offered load

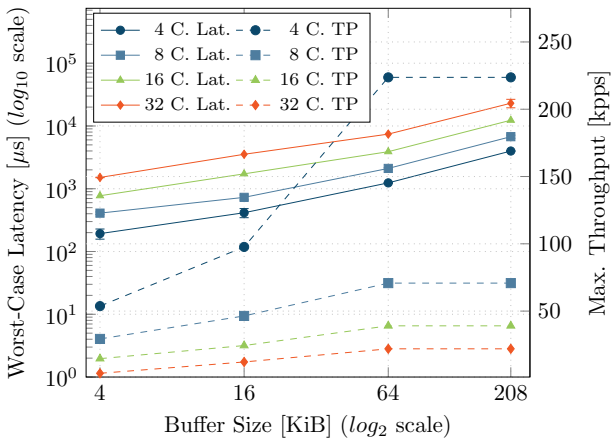

Fig. 4. Impact of UDP buffer sizes on worst-case latency and throughput

increases with a varying number of clients that need to receive a packet. We will use these results as a baseline. A higher number of clients increases the number of packets sent per incoming packet as well as the total required bandwidth. This has a significant effect on the total latency.

Figure 3 shows a histogram with a bin size of 0.5 µs for the measured latencies for the server with 4 connected clients under moderate load (30 kpps). Histograms for other measurement points have a similar shape. The different peaks in the distribution are likely caused by the large number of queues on the packet's path.

The offered loads at which packets were dropped go in hand with a sudden and high increase in latency. Figure 2 marks the first measurements where more than 5% of the packets were lost with a black arrow. The minimum observed latency was 39.6 µs (σ = 4.9 µs) at an offered load of 9.7 kpps, the highest 26.8 ms (σ = 0.2 ms) when the system was overloaded with 32 clients and 95 kpps and had to drop packets. This excessive worst-case latency is, of course, unrealistic under normal server operation. However, a malicious attacker (or a bug in the client) could exploit this behavior and flood the server with packets. The bandwidth required for such a denial of service attack is relatively low: 20 kpps to overload a server with 32 clients corresponds to only 10 MBit/s when using minimally sized UDP packets (64 byte). Examples for such attacks are the recent denial of service attacks on multiple online games including the PlayStation Network, Battle.net, and League of Legends [17], [18].

This large worst-case latency of 20 ms in the overload scenario is caused by the relatively large default buffer size of UDP buffers under Linux: 208 KiB in our Debian test system. Prevention mechanisms like rate-limiting clients at an application level are therefore also not effective if the attacker can fill this buffer which is managed by the OS and not by the application.

Figure 4 shows the effect of modifying the buffer size (via the `setsockopt` syscall). The graph plots both the maximum offered load that does not cause packet loss and the average latency under overload conditions, i.e. when the DuT drops packets. Increasing the buffer size beyond 64 KiB does not improve the throughput any further while the worst-case latency still grows linearly with the buffer size. The buffer size

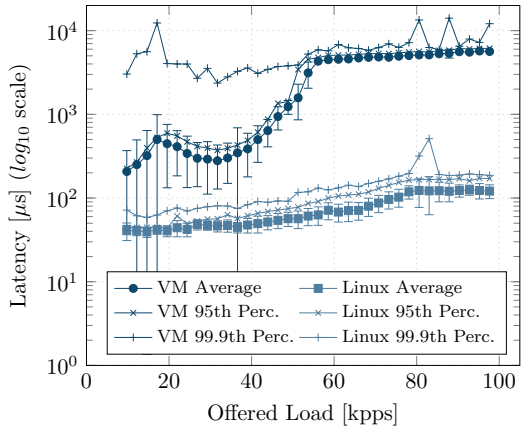

Fig. 5. Virtualization overhead, 4 clients, 64 KiB buffer size

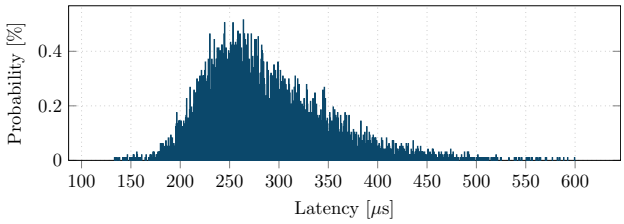

Fig. 6. Histogram of VM latencies for 4 clients, default buffer size, 30 kpps offered load

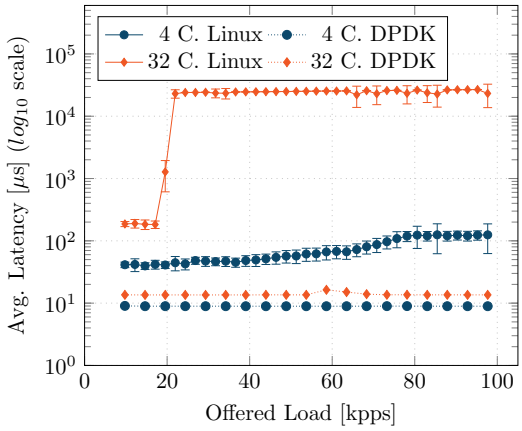

Fig. 7. Latency with DPDK compared to Linux with default buffer size

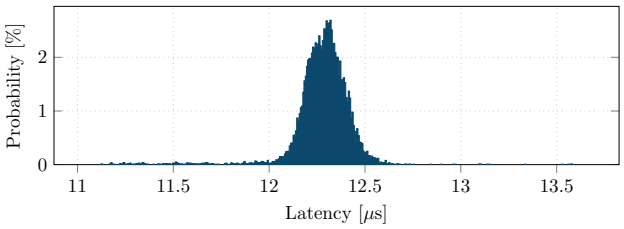

Fig. 8. Histogram of DPDK latencies for 16 clients, 500 kpps offered load

has no effect on the latency for lower packet rates. A buffer size of a maximum of 64 KiB should therefore be chosen for a game server. The buffer is also required to mitigate bursts in the traffic, so the optimum value is specific to the game's traffic characteristics and must be adjusted individually.

B. Virtual Machines

Virtualizing servers and cloud computing is nowadays a commonplace technique to simplify server management and increase availibility [19]. Virtualization comes with increased latency for networking applications. A packet now needs to be processed by two operating systems: by the hypervisor and the virtualized guest OS.

Figure 5 shows how the latency changes if the game server is moved into a VM. The scenario with 4 clients and 64 KiB buffer size is chosen as a representative example, other configurations follow a similar pattern. The peak in latency at around 20 kpps is visible in all configurations with VMs. It is caused by an increase in the system load as the dynamic adaption of the interrupt rate starts too late in this high-load scenario.

The graph also shows how the 99.9th percentile of the latency increases by a disproportionately large factor. This is also visible in the histogram of the observed latencies shown in Figure 6. The probability distribution is now a long tail distribution which negatively affects the 99.9th percentile latencies. This effect in VMs has also been observed by Xu et. al. [20].

C. High-Performance Packet I/O Frameworks

We ported our emulated game server to DPDK. DPDK bypasses the Linux network stack completely (cf. Figure 1), so a DPDK application needs to implement all required protocols separately in the user space application. This does not pose a problem for a simple protocol like UDP.

Our emulated game server accepts packets from the NIC, checks if it is a UDP packet addressed to a configured port, waits 150 CPU cycles to emulate processing, and then sends out responses by copying the packet. The packets' layer 2 and 3 addresses are adjusted accordingly. We do not use any buffers beyond the on-board memory of the NIC and the 512-entry Rx and Tx DMA ring buffer of the DPDK driver.

Figure 7 compares the latency of the DPDK-based game server with the Linux scenario from Section V-A. The game server now achieves an average latency of 9 μs for 4 clients and 11 μs for 32 clients at 100 kpps. We also tested higher incoming packet rates in order to measure the worst-case conditions in overload scenarios. But the server was only limited by the 10 GBit/s line rate of the outgoing interface.

The latencies are also normally distributed with a very low standard deviation compared to the previously observed distributions. Figure 8 shows a histogram (bin width 0.0064 μs) of the latencies under a high system load: the offered load of 500 kpps with 16 clients causes an output traffic of 9.5 GBit/s. The observed average latency of 12.32 μs (σ = 0.67 μs) under these extreme conditions is still lower than the lowest latencies with the Linux network stack.

One important technique that allows for this speed-up is that DPDK processes packets in batches. All response packets

TABLE I. COMPARISON OF LATENCY BETWEEN THE EVALUATED DEPLOYMENT OPTIONS WITH 4 CLIENTS

Deployment	Buffer [KiB]	Load* [%]	Average [μs]	Std. Dev. [μs]	50th Perc. [μs]	95th Perc. [μs]	99th Perc. [μs]	99.9th Perc. [μs]
Linux	16	20	44.7	6.6	44.6	57.0	61.8	75.2
		90	338.3	73.8	321.9	492.1	552.3	590.9
		110	414.7	68.8	408.6	540.3	598.5	623.6
	64	20	40.7	9.6	41.8	44.4	45.4	71.7
		90	142.6	23.8	143.0	164.2	170.0	527.9
		110	1240.3	9.2	1251.4	1255.9	1270.8	1242.9
Linux VM	16	20	209.3	154.0	205.7	227.1	382.1	2292.5
		90	460.3	128.4	449.0	572.2	707.3	2354.1
		110	1006.8	120.3	995.0	1108.6	1179.1	2805.3
	64	20	207.6	161.0	204.9	225.2	341.0	3030.7
		90	940.8	290.5	849.9	1351.9	1991.6	3723.2
		110	4312.6	277.9	4295.3	4764.0	5030.3	5891.0
DPDK	N/A	†20	10.6	0.4	10.4	11.0	11.1	11.3
		†90	12.3	1.1	12.2	12.4	17.8	27.2
		†99	15.2	1.9	14.8	18.1	25.8	33.3

*) Normalized to the load at which the first drops were observed, i.e. a load $\geq$ 100% indicates an overload scenario
†) Normalized to the load at which the output hits the 10 GBit/s line rate

can be sent with a single call into the DPDK library which directly passes them to the NIC – without involving the OS. DPDK also avoids expensive context switches (cf. Section II-B2) which contributes to the improvement. The per-packet processing costs are therefore significantly lower in this scenario and thus achieve a higher throughput and lower latency at the same time.

D. Comparison

Table I compares selected configurations between the three discussed deployment options. The DPDK game server is the fastest in all configurations. Virtualization has a large impact on the 99.9th percentile of the latency: Even under light load 0.1% of the packets have a latency of more than 2 ms.

VI. CONCLUSION

We measured the influence of the network stack and evaluated the effect on games servers. Virtualization increases both the average latency and its standard deviation leading to less predictable performance. Running a game server in a VM should therefore be avoided. Our measurements showed that specially the buffer sizes have significant influence on the delay which results in a optimization problem between delay and throughput.

DPDK and similar frameworks are a promising new technique to improve latency and performance for UDP-based game servers. However, they come with an additional development cost as the developers now need to implement the whole IP and UDP stack in the server application instead of relying on the OS abstraction. These frameworks should be considered to reduce latency and operating costs of large game server deployments.

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] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[2] R. Huggahalli, R. Iyer, and S. Tetrick, "Direct Cache Access for High Bandwidth Network I/O," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 50–59, May 2005.

[3] J. H. Salim, "When NAPI comes to town," in *Proceedings of Linux 2005 Conference*, 2005.

[4] "Intel Ethernet Controller X540 Datasheet Rev. 2.7." Intel Corporation, March 2014.

[5] 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, pp. 556–571, 2009.

[6] "PF_RING," http://www.ntop.org/products/pf_ring/, ntop, last visited 2014-08-20.

[7] L. Rizzo, "netmap: a novel framework for fast packet I/O," in *USENIX Annual Technical Conference*, April 2012.

[8] "Intel DPDK: Data Plane Development Kit," http://dpdk.org/, Intel Corporation, last visited 2014-08-20.

[9] Intel, "Data Plane Development Kit: Programmer's Guide, Revision 6," January 2014.

[10] 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, March 2012, pp. 85–95.

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

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

[13] J. Whiteaker, F. Schneider, and R. Teixeira, "Explaining packet delays under virtualization," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 38–44, 2011.

[14] P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," *ArXiv e-prints*, Oct. 2014.

[15] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, July 2008.

[16] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, "Performance Characteristics of Virtual Switching," in *2014 IEEE 3rd International Conference on Cloud Networking (CLOUDNET'14)*, Luxembourg, 2014.

[17] J. Haywald, "Hackers take down League of Legends, EA, and Blizzard temporarily," http://www.gamespot.com/articles/hackers-take-down-league-of-legends-ea-and-blizzard-temporarily-update/1100-6416869/, GameSpot, December 2013, last visited 2014-08-20.

[18] "Sony PlayStation Network and other game services attacked," http://www.bbc.com/news/technology-28925052, BBC News, August 2014, last visited 2014-09-02.

[19] B. Munch, "Hype Cycle for Networking and Communications," Gartner, Report, July 2013.

[20] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding Long Tails in the Cloud," in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2013, pp. 329–342.