A Wake-Up Call for Kernel-Bypass on Modern Hardware

Matthias Jasny
TU Darmstadt

Muhammad El-Hindi
TU Darmstadt

Tobias Ziegler
TU München

Carsten Binnig
TU Darmstadt & DFKI

Abstract

Kernel-bypass technologies eliminate the overhead of traditional OS stacks, offering direct access to high-speed I/O devices such as network and storage. This paper argues that kernel-bypass is no longer an optional optimization but a necessary architectural strategy for I/O-heavy applications like database systems. The motivation stems from two trends: stagnating CPU performance and rapid advances in I/O hardware, such as 800 Gbit/s NICs and SSDs exceeding 12M IOPS. In our evaluation, we show that, given these trends, it is no longer possible for DBMSs to saturate modern NICs or SSDs with traditional kernel stacks. We thus urge the research community to prioritize kernel-bypass technologies to fully harness the potential of emerging hardware in database systems.

CCS Concepts

• **Information systems** → **Parallel and distributed DBMSs**; **Flash memory**; • **Networks** → **Performance evaluation**.

ACM Reference Format:
Matthias Jasny, Muhammad El-Hindi, Tobias Ziegler, and Carsten Binnig. 2025. A Wake-Up Call for Kernel-Bypass on Modern Hardware. In *21st International Workshop on Data Management on New Hardware (DaMoN '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3736227.3736235

1 Introduction

Kernel-bypass technologies enable I/O-heavy applications like database systems to interact directly with hardware, eliminating costly kernel overhead and significantly improving performance. Despite these advantages and promising prior work [10, 13, 22, 26, 27, 31], kernel-bypass remains underutilized by both industry and academia. In this paper, we argue that kernel-bypass is no longer just an optimization; it is necessary to fully exploit modern hardware. This urgency for kernel-bypass arises from two converging trends. First, Moore's Law, which historically provided steady CPU performance gains, is reaching its limits. Second, advancements in I/O hardware have accelerated significantly, exemplified by NICs reaching up to 800 Gbit/s and SSDs exceeding 12 million IOPS.

The core problem is that using I/O devices requires the CPU to orchestrate operations. As such, traditional kernel-based I/O stacks increasingly become CPU-bound. Prior work [8, 29] already stated that the kernel stack in the case of networks consumes roughly 40% of the CPU cycles in OLTP workloads. While this finding is concerning, the situation we point to is even more severe: **the overhead of the kernel stack prevents modern hardware from being utilized to its full potential**. For example, a Mellanox ConnectX-7 network interface card (NIC) can handle up to 280 million 64-byte messages per second [12, 24]. On a typical 3 GHz CPU with 64 cores, this message rate translates to only 686 CPU cycles (3e9 *cycles* × 64 *cores*/280e6 *messages*) per message. As illustrated in Figure 1, current kernel-based interfaces – including advanced implementations like io_uring – require at least six times this cycle budget per message. Utilizing the NIC using kernel-based stacks would necessitate increasing the CPU budget sixfold(!) from 64 to 320 cores, which is **infeasible** even with modern CPUs.[1]

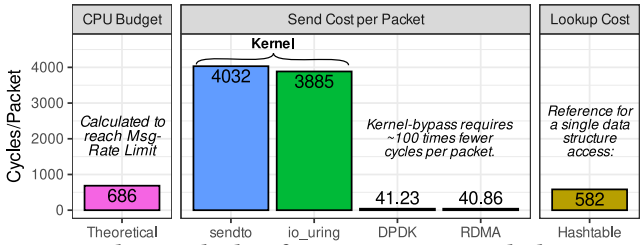

Figure 1: The CPU budget for saturating a 400G link using 64 cores is 686 cycles per 64B packet. The actual cost for kernel-based stacks is significantly higher. User-space networking like DPDK or RDMA is the only viable option, leaving room for data structure lookups.

In contrast, kernel-bypass libraries such as DPDK and RDMA handle messages using approximately 40 cycles per message (Figure 1), representing a roughly 100-fold improvement over kernel-based interfaces. With 40 cycles per message, ample CPU cycles remain available for meaningful work, such as hash table lookups in key-value stores (c.f. Figure 1 right). Given that 800 Gbit NICs are already available and 1.6 Tbit NICs are emerging [21], stagnating CPU frequencies will further shrink the CPU budget per message, reinforcing the necessity of kernel-bypass approaches. This trend similarly applies to modern NVMe SSDs [15, 16].

The central contribution of this paper is to articulate the necessity for kernel-bypass and issue a **call to the research community** to investigate how to use these techniques in modern DBMSs. To show this urgency, we systematically evaluate modern 400 Gbit NICs and PCIe Gen5 SSD arrays. While our primary focus is on networking, we also show that the same principles apply to storage: modern SSD arrays exhibit similar overheads under kernel-based stacks, further reinforcing the need for kernel-bypass technologies across the entire I/O stack.

2 The Case for Kernel-bypass

As discussed earlier, utilizing modern NICs efficiently would require the kernel stack to be 100 times more efficient. Despite significant work on kernel optimizations – including io_uring – fundamental

[1] Moreover, multiple cores would have to share the same TX and RX queues on the NIC, further restricting the feasibility of scaling to this large number of cores.

DaMoN '25, Berlin, Germany

ACM ISBN 979-8-4007-1940-0/25/06
https://doi.org/10.1145/3736227.3736235

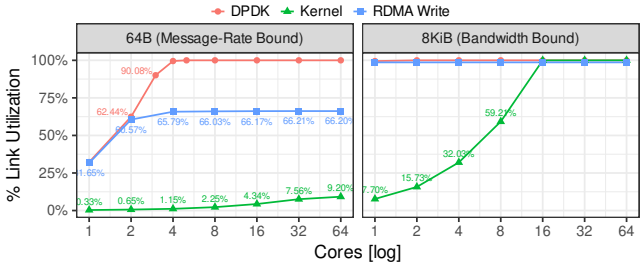

Figure 2: Required number of cores to reach full link utilization for different stacks. User-space networking is orders of magnitude more efficient than the kernel for sending 64B and 8KiB messages. The kernel-based stack cannot saturate the link for 64B and needs 16× more cores for 8KiB messages.

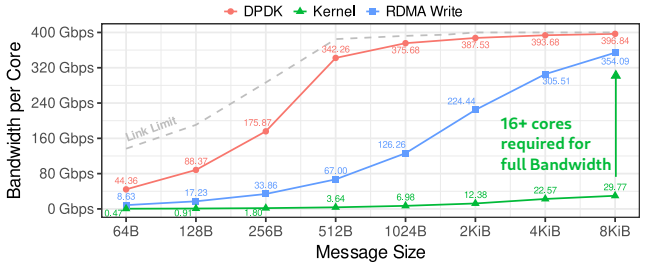

Figure 3: Bandwidth measured with a single core. DPDK saturates the network bandwidth earlier with a single core. The Kernel stack achieves the lowest per-core bandwidth. Hence, saturating the NIC is only possible with multiple cores.

inefficiencies remain [9]. To understand where these inefficiencies stem from, we analyze the cycle breakdown of a simple UDP transfer of a 64-byte message using `perf` as shown in the table below:

	Driver	IP	UDP	Sockets	App	Other	**Total**
Cycles:	549.56	703.99	1439.02	883.81	31.44	455.62	4,032
Percent:	13.63%	17.46%	35.69%	21.92%	0.78%	11.3%	100%

The data shows that UDP and Sockets consume the most cycles, but nearly every component exceeds the theoretical CPU budget (686 cycles). There is no high pole in the tent since the overhead is distributed across the entire stack. Consequently, incremental kernel improvements, including io_uring, are insufficient.

Instead of optimizing an inherently inefficient system, we should abandon the kernel stack and develop database-aware networking using kernel-bypass. Database systems, for instance, could avoid kernel memory-management overhead by using techniques incompatible with the kernel [28]. The table shows that UDP processing alone requires ≈1,500 cycles, much of which stems from memory allocations, virtual memory manipulation, and data copies.

As shown in Figure 1, kernel-bypass stacks process messages in just 40 cycles – two orders of magnitude fewer than kernel-based approaches. This is surprising, given that both stacks perform similar tasks, such as generating UDP messages and interfacing with the NIC via a driver. The key advantage of kernel-bypass lies in its ability to interface directly with the NIC from an application. Bypassing the kernel removes costly system calls, context switches, and kernel-to-user-space data copies. Direct hardware access also enables polling-based I/O rather than interrupt-driven processing. Polling can be significantly more efficient. This reduces latency and improves throughput. As a result, databases can efficiently handle high-speed workloads with CPU cycles left for DB operations.

To leverage kernel-bypass, two major stacks are commonly used for networking: (1) The first is the Data Plane Development Kit (DPDK)[3], a widely adopted user-space networking library that enables direct access to NICs without kernel involvement. DPDK achieves high-speed packet processing by polling directly on the NIC's TX and RX queues, allowing packets to be transferred via Direct Memory Access (DMA) into the application's memory. However, DPDK only provides raw access to the Ethernet layer, meaning applications must implement higher-level protocols. Hence, multiple user-space TCP stacks have been built on top of DPDK, including F-Stack[2], Seastar[5], and mTCP[18]. These implementations benefit from DPDK's low latency, direct hardware access, and reduced CPU overhead. (2) The second major stack is RDMA (Remote Direct Memory Access)[7, 19], which enables memory-to-memory communication between nodes without CPU intervention. Like DPDK, RDMA operates in user-space, bypassing the kernel networking stack, but requires specialized hardware. However, RDMA further eliminates the need for an active target CPU because the NIC can directly write data into remote user-space memory. RDMA is a reliable protocol and operates over InfiniBand or RoCE (RDMA over Converged Ethernet)[8, 14].

3 An Evaluation on Modern Hardware

We established that kernel-bypass networking is more efficient than kernel-based networking. A careful reader might argue that while CPU frequency has stagnated, modern CPUs provide more cores, which increases the amount of available CPU cycles for I/O. To evaluate this claim, we conduct a set of microbenchmarks on state-of-the-art hardware, covering both high-speed networking and fast storage devices.

Experimental Setup. Our experiments are conducted on single-socket servers equipped with AMD EPYC 9554P processors (64 cores, up to 3.75 GHz) with SMT disabled, and 768 GiB of RAM, running Ubuntu 24.04 LTS on Linux kernel 6.15.0 with default kernel flags. Each node features a PCIe5 Nvidia ConnectX-7 MT2910 RDMA NIC, configured for Ethernet and connected to a 400 Gbit Intel Tofino2 switch. We use RDMA with RoCE on the same networking hardware. For storage experiments, we use a single server with eight PCIe5 NVMe SSDs (Kioxia KCMY1RUG7T68), each capable of 2.45M random read IOPS, aggregating to a maximum throughput of 19.6M IOPS or 75GiB/s storage bandwidth. In practice, we measured slightly higher performance at 20.65M IOPS.

3.1 I/O Performance on Modern Networks

Scaling to Network Line Rate. We begin by examining the number of cores required to saturate a 400 Gbit NIC using different network stacks. In this experiment, we scale the number of cores while sending UDP messages of two sizes: small (64B) and large (8KiB). Figure 2 (left) confirms that with small messages, fully utilizing the packet rate of modern 400 Gbit NICs is infeasible with kernel-based networking. However, DPDK achieves full saturation, reaching 280 million messages per second with just four cores.

With larger messages (Figure 2 right), bandwidth becomes the limiting factor rather than packet rate. Saturating the link is easier in this case because CPU overhead, such as system calls, is amortized

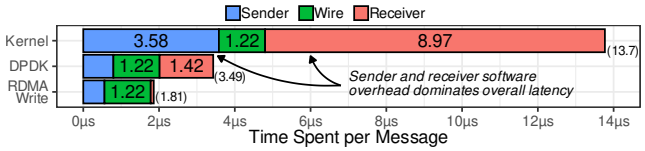

Figure 4: End-to-end latency breakdown comparison of kernel, DPDK and RDMA. The latency of the kernel-bypass stacks is significantly lower than that of the kernel stack.

over larger payloads. As a result, even kernel-based networking can eventually utilize the link, but it requires 16 times more cores than DPDK, which achieves full utilization with only a single core.

These experiments demonstrate that while high message rates remain impractical for kernel-based networking, larger messages, such as those in OLAP workloads, enable full link utilization. Nevertheless, considering ongoing hardware advancements, NICs with 800 Gbit/s require approximately 32 CPU cores, while a 1.6 Tbit/s link could potentially consume all 64 cores for kernel-based networking. This indicates that, even when larger messages partially amortize overhead, kernel-bypass remains essential for future-proofing OLAP systems as network speeds continue to increase.

Per-Core Throughput. Figure 3 shifts attention to the per-core throughput for varying message sizes. DPDK and RDMA achieve comparable peak performance for large messages. However, DPDK saturates the network bandwidth earlier with a single core (partially, since RDMA is reliable, while DPDK uses unreliable UDP), making it well-suited for maximizing throughput while minimizing CPU load. Reliable RDMA Writes consistently achieve a higher bandwidth than the kernel-based UDP stack, which does not exceed 30Gbps per core – even with large message sizes. The kernel stack scales linearly up to 4KiB, and requires 16 or more cores to reach full link utilization as shown in Figure 2.

Network Latency. Besides bandwidth utilization, network latency is another critical metric for database systems, particularly in OLTP transactions [11, 17, 30]. One might expect wire latency to dominate end-to-end message delay, but our measurements reveal that software processing, especially in the kernel stack, is the primary source of overhead. We break down latency into sender, wire, and receiver components to quantify this observation. To measure the different components, we leverage the hardware time-stamping capabilities of the ConnectX-7 NICs after synchronizing their clocks.

As shown in Figure 4, wire latency is approximately 1.2μs. A full UDP transfer in the kernel stack – from user-space on the sender to user-space on the receiver – takes 13.7μs, nearly an order of magnitude higher than the physical wire latency. This overhead is unevenly distributed between the sender and receiver. DPDK reduces total latency to 3.5μs, with wire latency accounting for 35%, while sender and receiver processing each contribute 15% and 40%, respectively. This breakdown demonstrates that kernel-bypass improves throughput and significantly reduces message latency, making it particularly attractive for latency-sensitive database workloads such as OLTP transactions.

To further examine the latency overhead of the different network stacks, we measure their end-to-end message latency and normalize it to the RDMA latency in Figure 5. RDMA is the baseline since it consistently achieves the lowest latency across all message sizes.

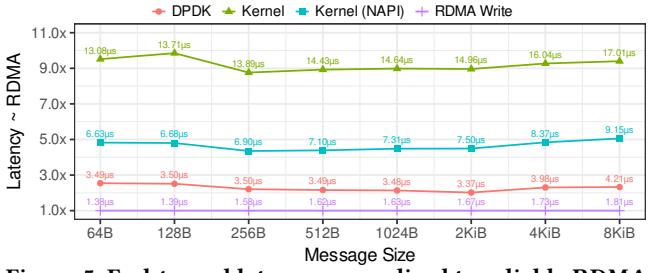

Figure 5: End-to-end latency normalized to reliable RDMA Writes. The other stacks' overhead is constant, independent of the message size. Hence, software processing overhead dominates wire latency.

The figure shows latency for a single round-trip, including sender processing, receiver processing, and wire transmission. The relative overhead of each stack remains stable across message sizes. The kernel stack exhibits the highest latency, up to 10× slower than RDMA. Kernel NAPI [4] reduces this overhead through polling within the driver, but is still 5× slower than RDMA[2].

Besides NAPI, we evaluated `AF_XDP` [1], a recent Linux kernel feature designed for high-performance packet processing. We observed benefits when processing can be offloaded to the eBPF virtual machine. However, in typical database scenarios represented by our microbenchmarks, where user-space processing is necessary, `AF_XDP` delivered latency and throughput comparable to – or in some cases worse than – standard UDP. We therefore excluded this baseline from the results.

TCP Overhead. Due to its reliability, TCP is more widely used than UDP in database systems, e.g., for transaction coordination or client-server communication [25]. We measure the TCP throughput for different networking stacks to evaluate the impact of using TPC over UDP. Figure 6 compares the previously reported raw DPDK (unreliable) performance against F-Stack [2] (TCP on top of DPDK) and kernel-based TCP. The figure shows the aggregated bandwidth across all cores for varying message sizes. Despite using kernel-bypass, F-Stack performs similarly to kernel-based TCP, highlighting that TCP is a major source of overhead.

We also applied moderate TCP tuning and assigned multiple connections to each core to ensure fair utilization. Nevertheless, saturating a 400 Gbit link requires almost all 64 cores with either TCP stack. In contrast, raw DPDK achieves peak throughput with only a few cores. These results emphasize the inefficiency of TCP for high-speed networking and the need for lighter, application-specific transport protocols.

3.2 I/O Performance on Modern SSDs

Beyond networking, storage I/O is critical in database systems and directly affects the performance of operations such as write-ahead-logging [23]. In the advent of fast NVMe PCIe5 SSDs, minimizing CPU cycles per I/O is essential to fully utilize the hardware bandwidth and leave room for query processing or other database tasks.

CPU cost per I/O. Figure 7 presents an ablation study of storage stack efficiency, reporting CPU cycles required per 4KiB read

[2]The Kernel (NAPI) baseline was excluded in the bandwidth experiments, since polling offers no significant benefit in this context.

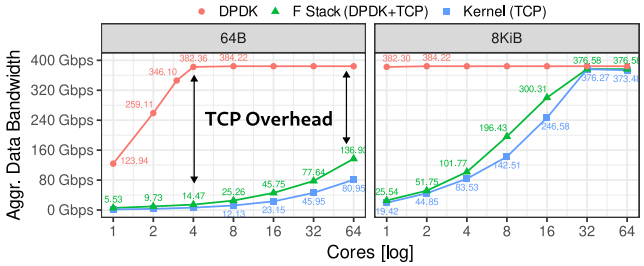

Figure 6: Comparison of a kernel-based and kernel-bypass TCP stack. Compared to UDP, the overheads are evident, even though F-Stack builds directly on top of DPDK.

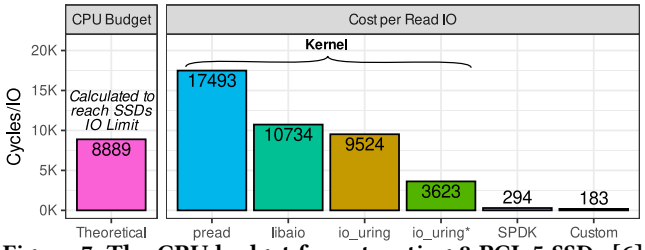

Figure 7: The CPU budget for saturating 8 PCIe5 SSDs [6] using 64 cores is 8.8K cycles per 4KiB read IO. The actual cost for kernel-based stacks is higher than the theoretical budget. User-space storage drivers like SPDK are the only viable option to saturate modern storage devices.

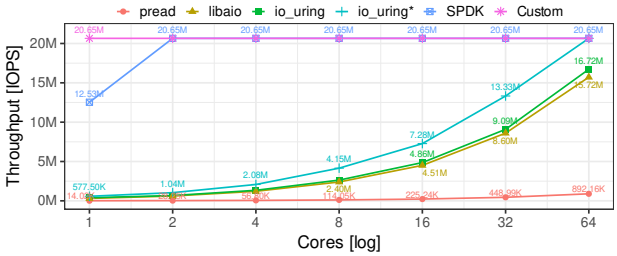

Figure 8: Random read throughput across 8 SSDs. Stock and custom SPDK reach peak IOPS with 1–2 cores. Kernel stacks require many cores or fail to saturate the SSDs.

I/O across various stacks. The theoretical CPU budget to saturate 8 PCIe Gen5 SSDs [6] using 64 cores is 8.8K cycles per I/O (3e9 *cycles* × 64 *cores*/21.6e6 *IOs*). All kernel-based stacks – pread, libaio, and io_uring – exceed this budget and cannot saturate the SSD bandwidth. With io_uring optimizations, such as buffer registration and fixed file-descriptors, enabled (io_uring*), the kernel-based stack is still an order of magnitude less performant than the user-space alternatives like stock SPDK.

The kernel-bypass stacks like stock SPDK and a minimal custom SPDK-variant, eliminating unnecessary indirections in SPDK [15], complete read I/Os in as few as 294 and 183 cycles, respectively – well below the computed theoretical threshold. These results mirror our networking findings: kernel-bypass is essential for high throughput and enabling additional computation alongside I/O operations. User-space storage drivers are thus the only viable option for interfacing with modern high-performance SSDs.

SSD Throughput. To highlight this finding further, we measure the random read throughput across storage stacks for an increasing number of cores, using the same setup with 8 PCIe5 SSDs. Figure 8 shows that kernel-bypass stacks, such as SPDK and our custom SPDK-variant, achieve the SSDs' full IOPS capacity (approximately 21M/s) with just 1–2 cores. In contrast, kernel-based stacks scale poorly, requiring many more cores to approach saturation, with some approaches (like pread) never reaching the hardware limit despite using all available cores. These results highlight the overheads of kernel-based I/O and underscore the importance of user-space drivers for saturating modern storage devices in database systems.

3.3 Implications for Database System Design

Our evaluation demonstrates that both networking and storage overheads in kernel-based approaches consume substantial CPU resources that could otherwise be dedicated to query processing. As hardware continues to advance with faster network and storage devices, these overheads become increasingly problematic. To remain computationally efficient, modern database systems must adopt kernel-bypass techniques that minimize per-I/O overhead. This is particularly crucial for analytical workloads that process large volumes of data, and latency-critical transactional workloads, where I/O latency directly impacts query performance.

4 A Call to the Database Community

Our research community must prioritize kernel-bypass technologies to fully benefit from future hardware advancements. Some systems have already demonstrated the use of RDMA-based networking as an alternative kernel-bypass approach, reporting significant performance gains [13, 22, 26, 31]. Yet, only a few database systems currently use kernel-bypass effectively, for example, ScyllaDB [27], Yellowbrick [10], and Oracle Exadata [26].

Several challenges currently limit wider adoption, creating opportunities for research to improve accessibility and usability.

First, implementing kernel-bypass is considerably simpler for storage devices like SSDs, where standardized protocols such as NVMe provide a precise specification for user-space libraries. NICs, however, lack a universal specification, making the implementation of the user-space driver complex, typically requiring custom solutions for each device. However, the increasing prevalence of virtualized NICs in the cloud may offer a promising avenue: developing lightweight user-space network libraries tailored to a limited set of standardized virtualized NICs.

Second, network protocols themselves might introduce complexities. UDP lacks essential guarantees for reliable database communication, while TCP/IP, though more robust, is difficult to implement efficiently in user-space. Even optimized TCP/IP implementations like TAS [20] require roughly 2,000 CPU cycles per packet, still prohibitive for high-performance databases. These overheads raise a fundamental question: Do databases require all TCP/IP guarantees, such as strict ordering, or can more efficient, application-specific protocols be designed? Using database semantics, custom protocols could reduce processing overhead while maintaining essential reliability properties. Exploring such tailored approaches could further enhance the viability and performance of kernel-bypass networking in database systems.

Acknowledgments

This work has been partially funded by the LOEWE Spitzenprofessur of the state of Hesse. We also thank hessian.AI and DFKI for their support.

References

[1] 2024. *AF_XDP - The Linux Kernel.* https://www.kernel.org/doc/html/next/networking/af_xdp.html

[2] 2024. *F-Stack.* https://github.com/F-Stack/f-stack

[3] 2024. *Intel DPDK (Data Plane Development Kit).* https://www.dpdk.org/

[4] 2024. *NAPI - The Linux Kernel.* https://docs.kernel.org/networking/napi.html

[5] 2024. *Seastar Networking.* https://seastar.io/networking/

[6] 2025. *KIOXIA CM7-R Series Enterprise NVMe Read Intensive SSD.* https://europe.kioxia.com/content/dam/kioxia/shared/business/ssd/enterprise-ssd/asset/productbrief/eSSD-CM7-R-product-brief.pdf

[7] Gustavo Alonso, Carsten Binnig, Ippokratis Pandis, Kenneth Salem, Jan Skrzypczak, Ryan Stutsman, Lasse Thostrup, Tianzheng Wang, Zeke Wang, and Tobias Ziegler. 2019. DPI: The Data Processing Interface for Modern Networks. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings.* www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p11-alonso-cidr19.pdf

[8] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2016. The End of Slow Networks: It's Time for a Redesign. *Proc. VLDB Endow.* 9, 7 (2016), 528–539. doi:10.14778/2904483.2904485

[9] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, Fernando A. Kuipers and Matthew C. Caesar (Eds.). ACM, 65–77. doi:10.1145/3452296.3472888

[10] Mark A Cusack, John Adamson, Mark Brinicombe, Neil A. Carson, Thomas Kejser, Jim Peterson, Arvind Vasudev, Kurt Westerfeld, and Robert Wipfel. 2024. Yellowbrick: An Elastic Data Warehouse on Kubernetes. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024.* www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p2-cusack.pdf

[11] Haowen Dong, Chao Zhang, Guoliang Li, and Huanchen Zhang. 2024. Cloud-Native Databases: A Survey. *IEEE Trans. Knowl. Data Eng.* 36, 12 (2024), 7772–7791. doi:10.1109/TKDE.2024.3397508

[12] DPDK. 2024. *NVIDIA NICs Performance Report with DPDK 24.07, Rev 1.1.* https://fast.dpdk.org/doc/perf/DPDK_24_07_NVIDIA_NIC_performance_report.pdf

[13] Philipp Fent, Alexander van Renen, Andreas Kipf, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2020. Low-Latency Communication for Fast DBMS Using RDMA and Shared Memory. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020.* IEEE, 1477–1488. doi:10.1109/ICDE48307.2020.00131

[14] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, Marinho P. Barcellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti (Eds.). ACM, 202–215. doi:10.1145/2934872.2934908

[15] Gabriel Haas, Adnan Alhomssi, and Viktor Leis. 2025. Managing Very Large Datasets on Directly Attached NVMe Arrays. In *Scalable Data Management for Future Hardware.* Springer, 223–240.

[16] Gabriel Haas and Viktor Leis. 2023. What Modern NVMe Storage Can Do, And How To Exploit It: High-Performance I/O for High-Performance Storage Engines. *Proc. VLDB Endow.* 16, 9 (2023), 2090–2102. doi:10.14778/3598581.3598584

[17] Matthias Jasny, Lasse Thostrup, Tobias Ziegler, and Carsten Binnig. 2022. P4DB - The Case for In-Network OLTP. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1375–1389. doi:10.1145/3514221.3517825

[18] Eunyoung Jeong, Shinae Woo, Muhammad Asim Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, Ratul Mahajan and Ion Stoica (Eds.). USENIX Association, 489–502. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/jeong

[19] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *Proceedings of the 2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*, Ajay Gulati and Hakim Weatherspoon (Eds.). USENIX Association, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[20] Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr. Sharma, Arvind Krishnamurthy, and Thomas E. Anderson. 2019. TAS: TCP Acceleration as an OS Service. In *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*, George Candea, Robbert van Renesse, and Christof Fetzer (Eds.). ACM, 24:1–24:16. doi:10.1145/3302424.3303985

[21] Alberto Lerner, Carsten Binnig, Philippe Cudré- Mauroux, Rana Hussein, Matthias Jasny, Theo Jepsen, Dan R. K. Ports, Lasse Thostrup, and Tobias Ziegler. 2023. Databases on Modern Networks: A Decade of Research that now comes into Practice. *Proc. VLDB Endow.* 16, 12 (2023), 3894–3897. doi:10.14778/3611540.3611579

[22] Feifei Li. 2019. Cloud native database systems at Alibaba: Opportunities and Challenges. *Proc. VLDB Endow.* 12, 12 (2019), 2263–2272. doi:10.14778/3352063.3352141

[23] Lamduy Nguyen, Adnan Alhomssi, Tobias Ziegler, and Viktor Leis. 2025. Moving on From Group Commit: Autonomous Commit Enables High Throughput and Low Latency on NVMe SSDs. In *SIGMOD '25: International Conference on Management of Data, Berlin, Germany, June 22 - 27, 2025*, Volker Markl, Joe Hellerstein, and Azza Abouzied (Eds.). ACM.

[24] NVIDIA. 2021. *NVIDIA ConnectX-7 Datasheet.* https://www.nvidia.com/content/dam/en-zz/Solutions/networking/infiniband-adapters/infiniband-connectx7-data-sheet.pdf

[25] Mark Raasveldt and Hannes Mühleisen. 2017. Don't Hold My Data Hostage - A Case For Client Protocol Redesign. *Proc. VLDB Endow.* 10, 10 (2017), 1022–1033. doi:10.14778/3115404.3115408

[26] Jia Shi. 2021. Under the Hood of an Exadata Transaction – How did we harness the power of Persistent Memory? Keynote presentation at the Advanced Data Management Symposium (ADMS). https://adms-conf.org/2021-camera-ready/jia_presentation.pdf.

[27] Nishant Suneja. 2019. Scylladb optimizes database architecture to maximize hardware performance. *IEEE Softw.* 36, 4 (2019), 96–100. doi:10.1109/MS.2019.2909854

[28] Xinjing Zhou, Viktor Leis, Jinming Hu, Xiangyao Yu, and Michael Stonebraker. 2025. Practical DB-OS Co-Design with Privileged Kernel Bypass. *Proc. ACM Manag. Data* 3, 1 (2025), 64:1–64:27. doi:10.1145/3709714

[29] Xinjing Zhou, Viktor Leis, Xiangyao Yu, and Michael Stonebraker. 2025. OLTP Through the Looking Glass 16 Years Later: Communication is the New Bottleneck. In *15th Conference on Innovative Data Systems Research, CIDR 2025, Amsterdam, NL, January 19-22, 2025.* www.cidrdb.org. https://www.cidrdb.org/cidr2025/papers/p17-zhou.pdf

[30] Tobias Ziegler, Philip A. Bernstein, Viktor Leis, and Carsten Binnig. 2023. Is Scalable OLTP in the Cloud a Solved Problem?. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023.* www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p50-ziegler.pdf

[31] Tobias Ziegler, Carsten Binnig, and Viktor Leis. 2022. ScaleStore: A Fast and Cost-Efficient Storage Engine using DRAM, NVMe, and RDMA. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 685–699. doi:10.1145/3514221.3526187