An Ultra-Low Latency and Compatible PCIe Interconnect for Rack-scale Communication

Yibo Huang⋆, Yukai Huang⋆, Ming Yan⋆, Jiayu Hu†, Cunming Liang†, Yang Xu⋆, Wenxiong Zou⋆, Yiming Zhang⋆, Rui Zhang⋆, Chunpu Huang⋆, Jie Wu⋆
⋆Fudan University, China †Intel, China

1 MOTIVATION

The emerging resource disaggregation architecture emphasizes the need for ultra-low latency and high throughput rack-scale communication within data centers, enabling high-quality online services and real-time analysis [21, 7, 5, 10, 22, 8]. The lower the latency, the better [15, 6]. There is a growing trend that, high-density modern computing (e.g., TPU [1]) and storage (e.g., Non-Volatile Memory [23]) hardware are deployed and disaggregated in a rack [5], termed as rack-scale computers [25], which shift the potential bottleneck from computation to network [24, 16]. The latency requirements of rack-scale networks need to be maintained in the range of 3-5us [5]. However, many Ethernet-based studies on rack-scale networks [4, 13] are dependent on either workload which limits generalization, or a centralized controller which compromises performance. Therefore, recent microsecond-latency kernel-bypassing networking technologies, including hardware offloading like Remote Direct Memory Access (RDMA) [9] or user-space IO stacks like DPDK [11], are used as key enablers for high-speed rack-scale communication.

Unfortunately, these state-of-the-art works fundamentally rely on heavily layered protocol stacks, and the translation overheads between protocol layers are inevitable. For example, while RDMA over Converged Ethernet (RoCE) [9] allows the network card to directly access the memory of remote machines, one-way data movement in Figure 1(a) requires at least four translations among PCIe bus, RDMA protocol (i.e., IBTA protocol) and UDP. Thus, it is hard to avoid extra latency overhead due to this protocol translation. Meanwhile, the Ethernet-based access to PCIe-attached peripherals across machines in a rack further increases the frequency of data-path translations. Additionally, protocol stack hardware offloading schemes also require complex connection/memory resource management within RNICs, which further adds communication overhead. For example, RDMA exploits limited RNIC memory to cache physical-virtual memory mapping tables and connection contexts, resulting in higher tail latency under cache miss [12].

Opportunities with high-speed PCIe interconnect. Can we get rid of protocol translation overhead and complex in-NIC resource management for the rack-scale networks to further reduce the communication latency? We argue that using advanced PCIe interconnect, such as compute express link (CXL), Gen-Z and PCIe non-transparent bridge (NTB), presents an optimal choice for high-speed networking, due to no translation between PCIe and network protocol and bypassing complex in-NIC resource management (as shown in Figure 1(b)). Particularly, PCIe NTB provides as a special device, allows to interconnect independent machines and peripherals to the same PCIe fabric like a bridge [17, 20]. By mapping the shared memory space between independent systems via reliable transaction layer protocol (TLP), it enables nanosecond-level ultra-low latency and efficient remote memory access. For example, PCIe NTB achieves up to 5.6 × speedup compared to RDMA in Figure 3). It can also achieve high throughput by matching evolving PCIe bandwidth (e.g., PCIe 6.0×16). Due to the well-known inefficiency (e.g., high system call overhead) [11, 14] of the kernel-space implementation (e.g., *ntb_hw_intel*) [2], this paper focuses only on user-space NTB (e.g., *DPDK polling mode driver*) [19] to bypass the kernel's complexity.

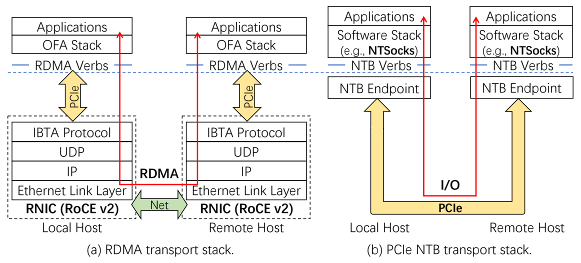

Figure 1: PCIe fabric avoids the translation between PCIe and network protocol, compared to RDMA.

2 KEY INSIGHTS AND CONTRIBUTIONS

Therefore, our vision is using PCIe interconnect for high-speed rack-scale network, as shown in Figure 2. We propose an in-rack network architecture with PCIe NTB fabric, where one PCIe cluster switch is used to do memory address routing among different PCIe domains and a PCIe NTB endpoint at end hosts is leveraged to process memory-mapped IO request. By this, we can enable ultra-low latency and lightweight PCIe interconnect for rack-scale communication. However, we find that native PCIe NTB lacks transparency support because it is originally designed for device sharing

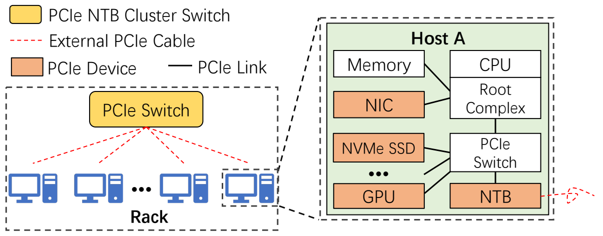

Figure 2: The in-rack network with PCIe NTB fabric.

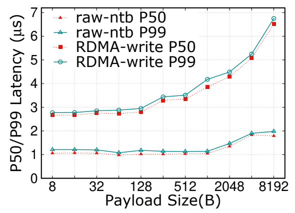

Figure 3: User-space PCIe NTB can achieve better latency than RDMA (Mellanox ConnectX-5 RNIC).

across independent PCIe domains. So, we designed and implemented NTSocks, a lightweight end-host network stack over PCIe interconnect for rack-scale applications to preserve high performance in a transparent manner. With this indirection layer, any existing rack-scale applications over socket communication can run with minimal or no code modification.

Based on this architecture, we make the following contributions:

- We propose a *generic* and *high-performance* socket-like abstraction with common network functions by using: i) lock-free ringbuffers over NTB memory via *Remote Write* primitive, ii) transparent zero-copy support, iii) and adaptive receiver-driven flow control for preventing message overflow.
- We design a dataplane *core-partition model*, which allocates cores for each partition on demand (e.g., one core for multiple partitions), to trade-off between *multi-core scalability* and *CPU efficiency*. *Partition* is a new abstraction for scalability on multi-core machines that divides the limited NTB-enabled shared memory into multiple parallel units, and each unit (i.e., *a partition*) is *core-driven* and multiplexed by a set of connections.
- We propose a *hierarchical performance isolation mechanism* on top of *partition* with: i) a per-connection message slicing for eliminating head-of-line (HOL) blocking among *intra-partition* connections, and ii) *inter-partition* load balancing at connection granularity which uses a round-robin connection distribution based on workloads (e.g., number of per-*partition* active connections).
- We evaluate the benefits of NTSocks with micro-benchmarks and multiple real-world applications. We open source NTSocks at `https://github.com/NTSocks/`.

3 MAIN ARTIFACTS

By building NTSocks on DPDK NTB Polling Mode Driver (PMD), we implement NTSocks in C language, including three components: *NTP*, *libnts* and *NTM*. Based on the DPDK NTB polling mode driver, we implement data plane component *NTP* with about 2500 lines of C code (LoCs). The runtime library *libnts* can be directly linked to the application program to provide all POSIX socket-related function calls, and its logic implementation uses about 3700 LoCs. The control plane *NTM* runs on each machine as a daemon process, stores user-defined control plane information, and provides IP port allocation, access control, fault migration, and TCP fallback capabilities. It takes about 4100 LoCs to implement *NTM*. The above three components all rely on a customized general function library *libnts-utils* implemented by about 4000 LoCs, which provides functions such as inter-process SHM communication protocol, SHM pool, hash function, and so on. NTSocks currently supports 64-bit X86 architecture, adapting to other platforms which needn't change lots of code. The entire NTSocks system runs as a user-mode process in the Linux system environment without any changes.

4 KEY RESULTS

We find that NTSocks outperforms the prior state-of-the-art network stacks [12, 18] with acceptable overhead while realizing high scalability and isolation. For example, NTSocks achieves dramatically better latency by up to 20.4× and 2.3×, and lower tail latency by up to 22.7× and 2.6× than Linux TCP and *libVMA* [18], respectively. We further port typical *Key-Value Store (KVS)*, *Nginx* and *Apache benchmarking tool (ab)* to NTSocks with little or even no modification on them. By benchmarking *KVS* with various YCSB workloads [3], NTSocks achieves better latency by up to 24.5× and 1.58×, compared to TCP Redis and RDMA respectively. For Nginx, NTSocks outperforms Linux TCP by up to 6.7×.

REFERENCES

[1] Google Cloud. 2018. Tpu pods. https://cloud.google.com/tpu/. (2018).
[2] Linux Kernel Community. 2020. Ntb drivers in linux kernel. https://www.kernel.org/doc/Documentation/ntb.txt. (2020).
[3] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving sys- tems with ycsb. In *Proceedings of the First ACM Symposium on Cloud Computing*, 143–145.
[4] Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. 2015. R2c2: a network stack for rack-scale computers. *ACM SIGCOMM Computer Communication Review*, 45, 4, 551–564.
[5] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network requirements for resource disaggregation. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 249–264.

[6] Yixiao Gao et al. 2021. When cloud storage meets {rdma}. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*, 519–533.

[7] Dan Gibson et al. 2022. Aquila: a unified, low-latency fabric for datacenter networks. In *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*, 1249–1266.

[8] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. 2017. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 649–667.

[9] 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 2016 ACM SIGCOMM Conference*, 202–215.

[10] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. 2022. Clio: a hardware-software co-designed disaggregated memory system. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 417–433.

[11] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. Mtcp: a highly scalable user-level {tcp} stack for multicore systems. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 489–502.

[12] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter rpcs can be general and fast. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 1–16.

[13] Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, Antony Rowstron, Hugh Williams, and Xiaohan Zhao. 2016. Xfabric: a reconfigurable in-rack network for rack-scale computers. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 15–29.

[14] Bojie Li, Tianyi Cui, Zibo Wang, Wei Bai, and Lintao Zhang. 2019. Socksdirect: datacenter sockets can be fast and compatible. In *Proceedings of the ACM Special Interest Group on Data Communication*, 90–103.

[15] Yuliang Li et al. 2019. Hpcc: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, 44–58.

[16] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. 2018. Parameter hub: a rack-scale parameter server for distributed deep neural network training. In *Proceedings of the ACM Symposium on Cloud Computing*, 41–54.

[17] Jonas Markussen, Lars Bjørlykke Kristiansen, Håkon Kvale Stensland, Friedrich Seifert, Carsten Griwodz, and Pål Halvorsen. 2018. Flexible device sharing in pcie clusters using device lending. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, 1–10.

[18] Mellanox. 2019. Messaging accelerator (vma). Available at https://github.com/mellanox/libvma. (2019).

[19] DPDK Project. 2020. Ntb rawdev driver. https://doc.dpdk.org/guides/rawdevs/ntb.html. (2020).

[20] Jack Regula. 2004. Using non-transparent bridging in pci express systems. *PLX Technology, Inc*, 31.

[21] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. Legoos: a disseminated, distributed {os} for hardware resource disaggregation. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 69–87.

[22] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. 2020. Disaggregating persistent memory and controlling them remotely: an exploration of passive disaggregated key-value stores. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 33–48.

[23] Xingda Wei, Xiating Xie, Rong Chen, Haibo Chen, and Binyu Zang. 2021. Characterizing and optimizing remote persistent memory with rdma and nvm. In *2021 {USENIX} Annual Technical Conference ({USENIX}{ATC} 21)*, 523–536.

[24] Xiantao Zhang, Xiao Zheng, Zhi Wang, Hang Yang, Yibin Shen, and Xin Long. 2020. High-density multi-tenant bare-metal cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 483–495.

[25] Hang Zhu, Kostis Kaffes, Zixu Chen, Zhenming Liu, Christos Kozyrakis, Ion Stoica, and Xin Jin. 2020. Racksched: a microsecond-scale scheduler for rack-scale computers. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 1225–1240.