Xenic: SmartNIC-Accelerated Distributed Transactions

Henry N. Schuh
Univ. of Washington

Weihao Liang
Univ. of Washington

Ming Liu
Univ. of Wisconsin, Madison

Jacob Nelson
Microsoft Research

Arvind Krishnamurthy
Univ. of Washington

Abstract

High-performance distributed transactions require efficient remote operations on database memory and protocol metadata. The high communication cost of this workload calls for hardware acceleration. Recent research has applied RDMA to this end, leveraging the network controller to manipulate host memory without consuming CPU cycles on the target server. However, the basic read/write RDMA primitives demand trade-offs in data structure and protocol design, limiting their benefits. SmartNICs are a flexible alternative for fast distributed transactions, adding programmable compute cores and on-board memory to the network interface. Applying measured performance characteristics, we design Xenic, a SmartNIC-optimized transaction processing system. Xenic applies an asynchronous, aggregated execution model to maximize network and core efficiency. Xenic's co-designed data store achieves low-overhead remote object accesses. Additionally, Xenic uses flexible, point-to-point communication patterns between SmartNICs to minimize transaction commit latency. We compare Xenic against prior RDMA- and RPC-based transaction systems with the TPC-C, Retwis, and Smallbank benchmarks. Our results for the three benchmarks show 2.42×, 2.07×, and 2.21× throughput improvement, 59%, 42%, and 22% latency reduction, while saving 2.3, 8.1, and 10.1 threads per server.

CCS Concepts

• **Information systems → Distributed database transactions**; **Parallel and distributed DBMSs**.

Keywords

Distributed Transactions, SmartNICs, RDMA.

SOSP '21, October 26–29, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8709-5/21/10…$15.00
https://doi.org/10.1145/3477132.3483555

ACM Reference Format:
Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. 2021. Xenic: SmartNIC-Accelerated Distributed Transactions. In *ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21), October 26–29, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3477132.3483555

1 Introduction

Distributed transactions, though a valuable programming model, are a challenging workload in the datacenter environment. Providing replication and serializability requires coordination between multiple shards of data with multiple replicas of each shard. Together, these guarantees incur a high communication cost, making the practicality of the distributed transaction model contingent on the performance of datacenter networks [9, 15, 45].

Recent developments in hardware and software acceleration have increased the performance of distributed transaction systems. Kernel-bypass networking reduces both the latency of network transfers and the end-host processing overhead. RDMA further cuts server processing costs by offloading simple memory operations from the server CPU to the NIC itself. One-sided RDMA enables reads, writes, and atomic operations on a remote server's memory without involving the remote server's CPU and with lower latency than operations that traverse the host networking stack. By expressing the transaction commit and replication protocol in terms of one-sided RDMA operations, the server-side computation involved in performing transactions can be eliminated, avoiding software network stack overheads [8, 9, 45].

A critical limitation of current RDMA NICs, however, is their small set of memory access primitives: read, write, fetch-and-add, and compare-and-swap. Applying these RDMA primitives to a distributed system typically requires significant design trade-offs in terms of data structures and protocol logic. For instance, a remote hash lookup using one-sided RDMA reads necessitates multiple network roundtrips for a hash miss; this can be mitigated by reading multiple buckets at once, but we then waste bandwidth to improve latency [8]. Likewise, one-sided RDMA supports only a request/response message pattern, limiting options for protocol communication. Often, these compromises negate the benefits of hardware offloading. In the context of distributed transactions,

using RPCs for some [44] or all [15] remote operations can lead to higher performance than fully applying one-sided RDMA. Ultimately, the limited applicability of one-sided RDMA limits the potential for performance benefit.

SmartNICs provide a path forward. These devices integrate compute cores and memory into the network interface, plus accelerators for common packet-processing functions. In particular, on-path ("bump in the wire") SmartNICs offer flexible compute cores on the packet data path, suggesting a new approach to hardware-accelerated distributed transactions. The SmartNIC's cores enable remote data structure accesses without network or RPC overhead. The on-board DRAM allows for maintenance of metadata state on the NIC itself, eliminating unnecessary PCIe memory accesses. When PCIe DMAs are required, the NIC can batch these operations to reduce overhead. Finally, the NIC can handle arbitrary protocol logic at both the source and remote target of a request, implementing flexible, multi-hop, network communication.

Given these potential benefits, we conduct a performance characterization of SmartNIC packet processing to identify the challenges and opportunities of using SmartNICs to accelerate distributed systems protocols. We find that the SmartNIC's software-based packet processing comes at a performance cost relative to hardware-supported RDMA. Therefore, a SmartNIC solution would only be effective if the NIC programmability can be used to significantly optimize communications over the wire and PCIe. Further, the NIC's limited resources pose a challenge. The NIC cores have low computational power relative to host cores, and the on-board memory is small. Careful placement of state and logic is critical to benefit from the NIC's limited resources. Operations must be interleaved and aggregated to effectively utilize NIC compute, PCIe, and network bandwidth.

Armed with these insights, we design Xenic, a SmartNIC-accelerated transaction processing system. Xenic adapts the protocol of prior designs to benefit from a stateful, asynchronous SmartNIC execution model. First, Xenic employs a co-designed data store that resides in host and SmartNIC DRAM, conforms to the SmartNIC's restrictions, and provides fast access to host-based data via indexing hints on the SmartNIC. Second, Xenic maintains temporary synchronization state on the SmartNIC to optimize concurrency control mechanisms. Third, Xenic takes advantage of the SmartNIC's flexible communication primitives and a function shipping interface [7, 9] to implement multi-hop (i.e., non-request-response) distributed concurrency control optimizations that lead to lower latencies and higher throughputs. Finally, Xenic achieves communication efficiency by asynchronously aggregating work at all inputs and outputs of the SmartNIC. The batched, asynchronous execution model enables high utilization of network bandwidth and the PCIe DMA engine.

We implement Xenic using Marvell LiquidIO SmartNICs [25] and compare it to well-optimized RDMA- and RPC-based designs using Mellanox CX5 RDMA NICs [29]. Our evaluation focuses on the TPC-C, Retwis, and Smallbank transaction benchmarks. On a 100Gbps network, Xenic demonstrates a 2.42×, 2.07×, and 2.21× peak throughput increase relative to the best-performing RDMA and RPC alternatives, for the three respective benchmarks, with 59%, 42%, and 22% improvements in median latency, while saving 2.3, 8.1, and 10.1 threads per server.

2 Background & Related Work

2.1 RDMA NICs

Modern datacenter NICs commonly implement a hardware-accelerated remote memory interface known as RDMA. An application uses RDMA by registering regions of host DRAM with the local NIC to enable remote access. There are two categories of RDMA operations:

One-sided RDMA operations are simple memory manipulations that are handled fully by the RDMA NIC. The target server's NIC parses the request, issues a PCIe DMA to read or write the host memory region, and sends a response over the network. One-sided verbs utilize connection-based transport. Three one-sided RDMA verbs are supported by mainstream RDMA NICs: (a) READ a remote memory address, copying the requested data over the network, (b) WRITE data from a local buffer to a remote address, returning a completion ack, and (c) ATOMIC compare-and-swap or fetch-and-add a remote buffer, returning the result.

Two-sided RDMA provides an efficient send/receive interface for message passing. Two-sided operations involve the host CPU on both sending and receiving ends. On the receiving end, the host CPU must poll for received messages, handle the buffer contents, and release the buffers to receive later messages. Two-sided RDMA offers a lightweight message-passing abstraction to support an RPC-based system but does not provide the CPU-bypass properties of one-sided RDMA.

2.2 Distributed Transactions

We target serializable distributed transactions over a replicated key-value store. The keyspace is partitioned, with designated primary and backup replicas for each partition. Recent work in this space assume persistent memory or battery-backed DRAM for fault tolerance and a separate service off the critical path to handle reconfiguration [9, 15, 44].

2.2.1 Commit Protocol Recent research systems share a similar commit protocol design, extending optimistic concurrency control (OCC) [43] with primary-backup replication for availability [6, 9, 15, 44]. Each transaction consists of a set of keys to read and a set of key-value pairs to write. The coordinator issues a series of operations for each transaction:

(1) In the EXECUTE phase, the coordinator reads all read-set objects from the objects' primary replicas. Writes are buffered locally at the coordinator, and the coordinator contacts the primary for each write-set key to lock the object. If a lock is already held, the transaction aborts.

(2) In the VALIDATE phase, the coordinator again reads each read-set value from its primary. If any value has changed after being read in the execution phase (determined using version counters), or its lock is held, the transaction aborts. Otherwise, the transaction will commit.

(3) In the LOG phase, the transaction record is written to a log on each write-set backup replica. The write-set updates are applied to the backup shards in the background.

(4) In the COMMIT phase, the coordinator applies the new write-set values to the primary replicas, increments the objects' version counters, and unlocks the objects.

2.2.2 RDMA-assisted Distributed Transactions

Recent systems use RDMA to accelerate the commit protocol, both by using one-sided verbs to reduce host involvement and two-sided verbs to implement low-latency RPCs.

FaRM [9]: FaRM's design prioritizes the use of one-sided RDMA. In particular, FaRM applies a Hopscotch hash data structure, enabling remote key lookups with a single one-sided READ. The Hopscotch structure incurs a high bandwidth overhead, as a neighborhood of multiple objects must be read for a single lookup. Further, key insertion also requires displacement of existing objects, which cannot be done efficiently using one-sided RDMA primitives. Thus, FaRM can fully offload execution and validation phase reads using one-sided RDMA, but it consumes remote CPU cycles for all other operations. To acquire write locks, to log transaction records, and to commit write objects, FaRM applies an RPC protocol based on one-sided WRITEs to pairwise message logs on each server. The servers poll logs and handle requests, sending back responses using the same mechanism.

FaSST [15]: Instead of pursuing the CPU and latency savings of one-sided RDMA, FaSST implements a lightweight two-sided RPC protocol for all remote operations. With an RPC model, no specialized data structure is required since lookups and insertions occur locally at the RPC handler. This avoids the read amplification and insertion complexity of FaRM's hash structure. FaSST also consolidates multiple operations into a single RPC: one RPC can lock a write-set object and retrieve a read-set value, providing performance benefits at the cost of host core usage.

DrTM+R [6]: DrTM+R aims to handle all remote operations with one-sided RDMA. This is accomplished with separate locking schemes for local and distributed transactions: remote locking uses one-sided ATOMIC operations, and local locking uses hardware transactional memory (HTM). DrTM+R addresses the incompatibility of RDMA ATOMICs with host CPU atomic instructions by applying an HTM procedure for each local key operation, and instead of optimistically reading and performing a validation check, the coordinator locks all keys in a transaction.

DrTM+H [44]: DrTM+H uses both one-sided RDMA and two-sided RPCs, performing a phase-by-phase selection of one-sided versus two-sided options to maximize performance. Like FaRM, DrTM+H uses one-sided RDMA to read remote records during the execution/validation phases and to write backup log entries. Writes during the execution and commit phases are done via RPCs. The RPC protocol makes use of two-sided RDMA, like FaSST, instead of FaRM's one-sided RDMA message logs. DrTM+H stores objects in a standard open hash table and achieves one-sided lookups in a single roundtrip by storing at each coordinator the remote memory address of each key, incurring a memory cost. Ultimately, DrTM+H exploits the CPU savings of one-sided RDMA to a limited extent while using two-sided RPCs for all other work. This selective use of one-sided RDMA shows a performance benefit relative to the purely two-sided alternative.

2.3 SmartNIC-based Systems

Emerging programmable NICs, or SmartNICs, represent another promising approach to reducing host processing overheads. By offloading computations onto a NIC-side multicore processor [4, 25, 28, 33, 34] or an FPGA [1, 10, 11, 30, 46], we can not only save server CPU cores but also achieve lower request latency and higher overall energy efficiency.

A SmartNIC has become a cost-effective computing unit for stateful packet processing, as the programmable components require only a modest amount of chip logic. For example, the Pensando Elba chip devotes less than 30% of its die for sixteen 2.8GHz ARM cores and the accompanying memory controller [26, 35], with the majority of the chip logic consumed by flow processing engines (e.g., Broadcom's TruFlow [3], Mellanox's ASAP2 [27], and Pensando's P4 [35]). Moreover, the enclosed processors and ASIC-based accelerators consume much less power than a Xeon-based solution when performing line-rate traffic processing (e.g., the Pensando NIC consumes less than 25W [26]).

Thus, there is a growing body of research on SmartNICs, with a significant focus on the offloading of network functions [2, 11, 12, 14, 17, 19, 20, 32, 37, 39, 40]. There is also work on generic frameworks for offloading [16, 21, 23, 36] and individual case studies focused on accelerating specific applications (e.g., key-value storage offloads [18, 22]). Our work is along these lines and examines the utility of SmartNICs in accelerating distributed transactions, an application that has traditionally been optimized using RDMA. Crucially, unlike prior efforts that focus on network functions or complete offloads of applications, we pursue a design where the

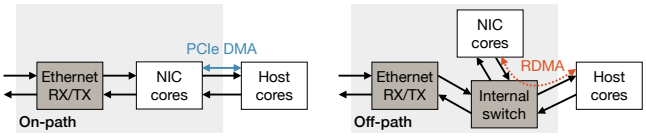

Figure 1: On-path (a) and off-path (b) SmartNIC architectures, showing packet data paths and the SmartNIC's host memory interface.

NIC and the host are closely coupled, with shared data structures and a fine-grained division of application logic.

3 SmartNIC Performance Analysis

We perform an experimental characterization to identify the opportunities and challenges of using SmartNICs. We focus on SmartNICs that enclose a system-on-chip (SoC) multicore processor. These SmartNICs offer the potential for hardware acceleration while, unlike RDMA, delivering a flexible, programmable interface. We first compare two SmartNIC architectures: *on-path SmartNICs* and *off-path SmartNICs* [21]. Second, we provide a performance evaluation of the 2x50GbE Marvell LiquidIO 3 CN3380 SmartNIC. The LiquidIO has 24 ARMv8 cores running at 2.2GHz, 16GB of on-board DDR4 DRAM, and an 8-lane PCI 3.0 interface. We compare the LiquidIO to the 100GbE Mellanox CX5 (MCX516A-CCAT) RDMA NIC. We detail our server specifications in §5.

3.1 On-Path and Off-Path SmartNICs

On-path SmartNICs (Figure 1a) implement a software packet pipeline, using the SoC to handle all inbound and outbound traffic between the Ethernet port and PCIe host (e.g., Marvell LiquidIO [24, 25] and Netronome Agilio [33]. The SoC exposes low-level hardware interfaces for packet manipulation. These include Ethernet RX/TX queues, packet buffer management, packet scheduling and ordering modules, and PCIe DMA engines. Functionality is offloaded to the SmartNIC by adding logic to the packet processing pipeline; the SoC can modify traffic, or generate traffic, to the host and the wire. Additionally, the SoC can manipulate host memory by issuing PCIe DMA requests. This architecture requires all traffic to be handled by SoC cores. As a result, offloaded application work and basic packet-processing tasks both contend for the limited SoC resources.

Off-path SmartNICs (Figure 1b) expose the SoC as a second network endpoint, with an internal packet switch between the host interface, SoC interface, and the wire. The SoC typically runs Linux with a full networking stack such as DPDK. Functionality is offloaded to the SoC by directing traffic to the SoC instead of the host, typically via a secondary Ethernet address. Current off-path devices lack a low-level interface for host memory manipulation. Instead, the SoC issues network requests to the host (e.g., RDMA operations or RPCs); the SmartNIC internally forwards traffic between the two endpoints. The off-path design allows traffic to be

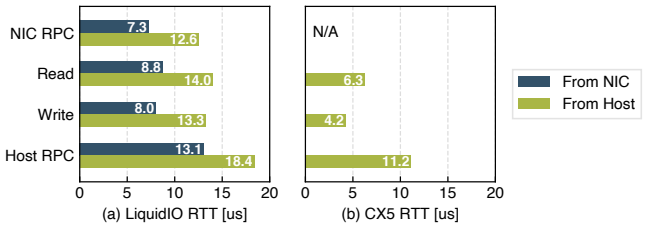

Figure 2: Roundtrip latency for LiquidIO SmartNIC remote operations originating from the host and from the NIC (a), and for CX5 RDMA (b). 256B data buffer used for all measurements.

selectively directed to the SoC; however, the full network stack on the SoC adds overhead to offloaded packet processing. Communication between the SoC and host is expensive due to the networking layers involved. To demonstrate this, we measure roundtrip latency for operations between a remote endpoint, the host, and the NIC SoC using the Mellanox Bluefield 1M322A [28]. While RDMA writes to host memory from a remote server have a median latency of 3.5us, we measure 4.5us to SoC memory from a remote server and 5.1us to host memory from the local SoC. The Broadcom Stingray PS225 [4] showed similar overheads: 7.6us to write host memory from a remote server, but 8.5us from the local SoC. This inflated SoC-to-host latency suggests that operations accessing host DRAM cannot be offloaded to the SoC without a prohibitive latency cost. We, therefore, target the on-path SmartNIC architecture for our offloading work.

3.2 NIC and Host Access Latency

In Figure 2, we present a roundtrip latency comparison of remote operations for the LiquidIO and CX5 NICs. For the LiquidIO, we measure operations initiated on the source host server via DPDK and operations initiated on the source NIC cores. For both sources, we measure the end-to-end latency of operations executed on the remote NIC, such as a NOP (NIC RPC case), DMA reads and writes to host memory (Read/Write cases), as well as two-sided operations handled by DPDK on the target-side server (Host RPC). For RDMA, we perform comparable experiments with the CX5 NIC, demonstrating READ and WRITE verbs, as well as two-sided RPCs using SEND/RECV verbs with the RPC framework from DrTM+H [44]. All cases use a 256B data payload; latency is similar for smaller sizes.

Our measurements show a latency penalty for the SmartNIC's software packet pipeline. RDMA operations, which leverage specialized hardware, demonstrate lower latency than the equivalent operations implemented in target-side LiquidIO and initiated from the host CPU. While the SmartNIC's software overhead poses a challenge, the ability to reduce costly PCIe operations presents an opportunity for performance improvement. Both devices show a significant cost for PCIe operations, with two-sided host RPCs incurring the highest latency. For the LiquidIO, operations local

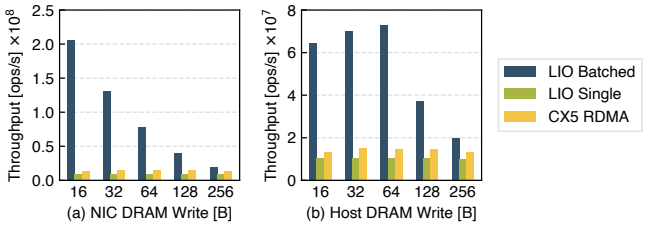

Figure 3: Remote memory write throughput, targeting SmartNIC DRAM (a) and host DRAM (b), with and without batching enabled. CX5 RDMA WRITE throughput is also shown for comparison.

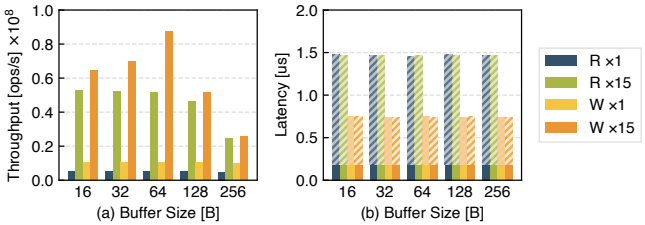

Figure 4: DMA engine throughput (a), and latency (b), with individual requests and with full 15-element vectors. For latency, solid bars denote submission time and hatched bars denote completion time.

to the NIC cores (e.g., NIC RPC with NIC source) outperform all operations involving PCIe accesses. Operations initiated from the NIC, avoiding the latency cost of a host-side DPDK roundtrip, outperform two-sided RDMA RPCs. These observations suggest the LiquidIO has the potential for latency improvement over two-sided RDMA without sacrificing the software flexibility of RPCs.

3.3 NIC and Host RPC Throughput

To compare packet-handling throughput between NIC and host cores, we implement a minimal echo RPC handler in a host DPDK application and in the LiquidIO firmware. For this experiment, RPC requests and responses consist of 80B UDP packets. We send requests from 5 remote servers to the target server and measure total response throughput. First, we deploy the host RPC handler on the target server, using 16 threads with dedicated RX/TX queues (enough to reach maximum throughput) and a packet burst size of 64. Second, we repeat the experiment with the same configuration but instead deploying the RPC handler on 16 NIC threads. In both cases, further increasing the thread count did not increase throughput. We measure an average host RPC throughput of 23.0Mops/s and an average NIC RPC throughput of 71.8Mops/s. This suggests that the NIC cores, though wimpier in computational performance (see §3.6), demonstrate higher packet-handling efficiency than the host-side alternative. Handling RPCs on the NIC cores, therefore, creates the potential for throughput improvement, in addition to latency reduction, relative to host RPCs.

3.4 Batching Optimizations

Using read and write microbenchmarks, we consider the potential of aggregation and batching at all stages of the packet pipeline. We apply a software batching layer at the NIC's PCIe TX/RX queues, Ethernet packet output, and DMA engine. We measure throughput for remote DMA writes to host memory and remote writes to the LiquidIO's on-board memory, with and without batching. To compare against the CX5, we measure RDMA WRITE throughput, applying doorbell batching [31] of up to 64 requests (more batching did not increase throughput). Figure 3 shows throughput for remote memory writes at a range of 16-256B buffer sizes, with and without batching. Reads demonstrate similar performance. We find that batching enables efficient bandwidth utilization for small remote memory operations. With batching disabled, throughput is consistent across the range of buffer sizes, 9.0-10.4Mops/s for both NIC and host memory targets. Batching network and PCIe transfers results in a throughput increase of up to 22.2× for NIC memory writes and 7.0× for host memory writes. For operations on remote NIC memory, throughput scales to the usable network bandwidth for all write sizes. For operations on host memory, throughput is limited by the DMA engine for requests smaller than 64B; larger requests saturate the usable network bandwidth. For CX5 RDMA, we observe 13.5-15.0Mops/s across the range of buffer sizes, lower than the respective batched LiquidIO operations. 16-256B RDMA writes do not saturate network bandwidth, even with extensive doorbell batching. This indicates that application-level doorbell batching is insufficient to achieve high throughput with small RDMA operations.

3.5 DMA Performance

To understand the performance characteristics of the LiquidIO's DMA hardware, we measure DMA throughput (Figure 4a) and latency (Figure 4b) for singular and vectored host memory accesses at a range of sizes. The DMA engine provides 8 hardware request queues; we initiate DMAs on 8 NIC cores, with each core assigned a dedicated queue. The DMA engine supports vectors of up to 15 reads or writes; we measure with individual requests and full vectors.

Our throughput measurements indicate that using vectored submission to batch DMAs improves throughput for the range of request sizes, up to the hardware maximum of 8.7Mops/s. Full vectors do not increase submission or completion latency relative to single-buffer requests. Instead, vectored operations may amortize the request submission time, up to 190ns, across up to 15 memory operations. Finally, we observe that the significant DMA completion latency, typically up to 1295ns for reads and 570ns for writes, must be hidden to efficiently utilize the NIC cores.

3.6 SmartNIC Core Performance

We compare the performance of the ARM and Intel Xeon Gold 5218 CPU cores using the Coremark benchmark and

Benchmark	Cores	ARM	Xeon	×
Coremark	multi	4530	14771	3.3
DPDK `hash_perf`	multi	349.8s	108.1s	3.2
DPDK `readwrite_lf_perf`	multi	179.6s	52.5s	3.4
Coremark	single	14294	29193	2.0
DPDK `memcpy_perf`	single	325.8s	174.4s	2.0
DPDK `rand_perf`	single	7.5s	2.9s	2.6
DPDK `hash_perf`	single	186.5s	84.0s	2.2

Table 1: Benchmark results for the NIC ARM and host Xeon cores, with relative the per-thread performance for the Xeon versus ARM.

relevant performance tests in DPDK's test suite. We measure single-threaded performance and per-thread performance for workloads utilizing all cores. Table 1 shows the results. For the LiquidIO's 2.2GHz 24-thread ARM CPU, we measure a Coremark throughput score of 108724 or 4530 per thread. The host-side 2.3GHz, 32-thread Xeon's score is 472691, with per-thread throughput 3.26× higher than that of the LiquidIO. While the Xeon's throughput scales with its 32 threads, the LiquidIO's per-thread Coremark throughput is substantially lower with all cores active. Running Coremark on one thread of each CPU, we observe higher relative throughput on the ARM, with a smaller 2.04× difference. The DPDK tests, demonstrating hash table, random number, and memcpy workloads, show a similar single-threaded (1.99× to 2.60×) and multi-threaded (3.24× to 3.42×) performance difference.

3.7 Opportunities

Our measurements suggest that the SmartNIC's software-based packet processing comes at a latency cost relative to RDMA. Despite this, we identify three optimization opportunities. The first is using the SmartNIC cores for stateful remote operations without host RPC overhead or one-sided RDMA limitations. NIC cores can handle protocol logic with the flexibility of an RPC design. NIC cores are also a valuable target for function shipping; logic can be pushed to NIC cores to eliminate PCIe roundtrips, exploit low-latency NIC-to-NIC communication (§3.2), and efficient packet-handling (§3.3). The second is using the SmartNIC memory to serve remote operations without PCIe overhead. With co-designed data structures spread across the host and NIC, we can use NIC memory to avoid PCIe latency. We can use PCIe DMAs to access host memory with lower latency than RPCs (§3.2), and high throughput potential relative to one-sided RDMA (§3.4). The third is leveraging the SmartNIC's efficient hardware interfaces, which show high throughput with software-defined asynchronous (§3.5), batched (§3.4) operations.

4 Design

Xenic provides a distributed, replicated database in server DRAM with a transactional interface. Each node acts as a transaction coordinator, a primary replica of one database shard, and a backup replica for f other shards, if we use a

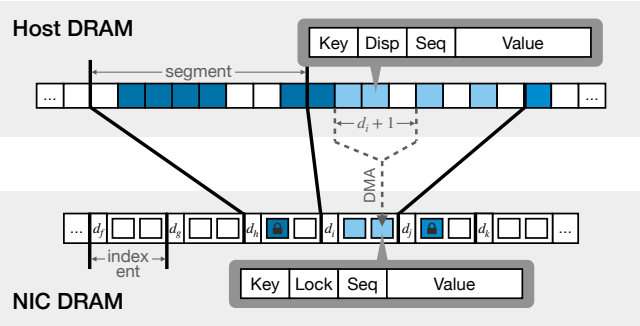

Figure 5: Overview of the Xenic data store, showing data and metadata placement. Overflow is omitted for simplicity.

replication factor of $f+1$. A coordinator application, running on each node, initiates transactions. The commit protocol utilizes the coordinator's local ("coordinator-side") SmartNIC, and the ("server-side") SmartNICs at remote primary and backup nodes. Xenic is designed to benefit from SmartNICs in the following ways:

- *Stateful offloads:* Xenic implements its transaction commit protocol as a set of stateful operations on the coordinator- and server-side SmartNICs. By storing temporary transaction state, e.g., locks, in SmartNIC memory, Xenic avoids PCIe roundtrips and host RPCs on the critical path.
- *Co-designed data store:* Xenic's data structures are spread across the host and SmartNIC memory. All key-value objects are stored in server DRAM, supporting local memory access at the server. For remote access, Xenic utilizes server-side SmartNIC memory to avoid PCIe reads for hot objects. By storing lightweight location metadata on the distribution of objects in host memory, Xenic can minimize the latency and size of DMAs for cold objects.
- *Distributed multi-hop OCC protocol:* Like prior systems [8, 9], Xenic uses function shipping [7]. But, Xenic can target SmartNICs and employ non-request-response protocols, unlike RDMA-based systems. This not only reduces PCIe operations but also enables flexible OCC communication protocols that reduce network communication.
- *Runtime support for asynchronous and batched communication:* Xenic performs all work asynchronously and aggregates operations at all inputs and outputs. By implementing an asynchronous, batched operation model, Xenic efficiently utilizes the limited SmartNIC cores. By batching work across PCIe DMAs, packet IO, and Ethernet transmissions, Xenic achieves high bandwidth utilization.

4.1 Co-designed Data Store

Xenic's data store is a co-designed hash structure residing in host and SmartNIC DRAM. All key-value objects are stored in host memory. The following factors drive this design choice. First, the host application may retrieve objects

via local memory access, not requiring communication with the SmartNIC. Second, the host memory size is typically much larger than that of the SmartNIC memory. Finally, the host's memory can be battery-backed to provide durability (as is the case in FaRM [9]). We optimize the host-side structure for efficient lookups and reads from the SmartNIC via PCIe DMA. Local transactions' lookups, insertions, and writes are performed on the host via local memory access.

The SmartNIC hash structure serves as a caching index of the host data store. It maintains fine-grained distribution metadata for regions of the host hash structure, enabling low-cost lookups via PCIe DMA. Given the latency of PCIe, we target lookups with a common-case single DMA read and low bandwidth overhead. In addition to storing lookup metadata, the SmartNIC structure maintains transaction metadata for key-value objects accessed by ongoing transactions.

This design leads to three possible cases for lookups: first, the host can perform lookups in its local memory; second, the SmartNIC can serve remote lookups of hot objects via its cache; third, upon a cache miss, the SmartNIC can retrieve objects in host memory via a low-overhead DMA read.

4.1.1 Data Structures The Xenic data store applies three structures. The host-side hash table (see §4.1.2) contains all key-value objects. The NIC hash table (see §4.1.3) caches hot objects and stores metadata. It is not a complete index of the host hash structure but instead a cache with location hints to facilitate efficient DMA lookups on the host-side hash table. The NIC table also stores transaction commit metadata, e.g., lock state, for ongoing transactions. Placing this metadata in NIC memory brings it closer, in terms of latency, to inbound remote accesses, while the on-path NIC architecture keeps it on the data path of outbound local requests. Figure 5 shows the layout of the host and SmartNIC hash structures. Finally, a host memory log (see §4.2) stores recently committed transactions. The NIC efficiently appends transactions' write sets to the log, and the host applies the updates to the host-side structure off the critical path.

4.1.2 Robinhood Hash Table Xenic's host-side hash structure is a closed hash table adopting the Robinhood hash table design [5], with several modifications to achieve efficient operations in the SmartNIC context. The Robinhood design is a form of closed hash table applying linear probing, which aims to reduce the cost of lookup probing by displacing existing objects as new ones are inserted. The insertion procedure attempts to even out the *displacement* of objects in the table: the distance of each object from its initial hash position. Objects with a comparatively low displacement are moved further as later insertions take place. The insertion probing function accomplishes this by checking the displacement of each element it reaches in the table. If the existing element's displacement is less than the current displacement of the element to be inserted, it swaps the existing element with the one to be inserted; thus, it steals the "displacement wealth" from well-placed elements and hands it out to other elements. Probing continues until reaching an empty slot, where the element to be inserted is placed.

This swapping procedure results in low probing distance variance throughout the hash table, even at high occupancy. While the insertion cost is higher than simply probing for an empty slot, the uniformity of probing distance improves lookup efficiency. This is important for lookups in the context of high-latency, throughput-limited memory access, i.e., PCIe DMA. Unlike other swapping designs, such as Cuckoo hashing, the Robinhood design prioritizes the locality of objects mapping to the same hash position. As a result, typical lookups read a single, contiguous region of memory instead of multiple disjoint buckets. This is crucial in the context of remote memory access, where the initiation cost of reading disjoint addresses is higher than that of a single buffer.

Xenic imposes a global limit on *maximum displacement*, D_m. Xenic divides the table memory into fixed-size *segments*, and for each segment, a linked overflow bucket may be allocated if necessary. If displacement reaches D_m during insertion, the object to be inserted is instead appended to the overflow bucket corresponding to its initial hash position. This allows Xenic to limit the cost of insertion and deletion. While prior Robinhood implementations typically apply tombstones to ensure an erased entry does not prematurely end probing, Xenic uses a simpler approach. Xenic simply swaps an overflow element over the deleted element, if one exists. If no overflow element exists, Xenic performs a backward shift; size is limited by D_m.

DMA-Consistent Swapping When Robinhood insertion swaps a table element, the existing element is replaced with the object to be inserted and buffered until the next swap occurs. A concurrent DMA read could therefore miss the existing element. Xenic addresses this by building a copy list and performing swaps starting from the last (free) element. This ensures an existing object is never removed from the table. Xenic must also guarantee consistent DMA reads for objects spanning multiple host cache lines. In this case, Xenic surrounds swaps with transactional memory instructions (`XBEGIN`, `XEND`), causing the swap to abort and retry if there is a concurrent DMA. Because the NIC caches objects returned in a DMA read, the host retries an aborted swap without continued contention. Xenic stores large objects above 256B outside the host hash table to avoid swapping large object payloads and reduce DMA lookup cost. Instead, the hash table contains pointers, which the NIC can use to retrieve the value via a single-object DMA read.

4.1.3 SmartNIC Caching Index Xenic uses NIC memory to maintain lookup metadata for the host-side hash table, as

Data Structure	Objects Read	Roundtrips
Xenic Robinhood, $D_m = 8$	3.43	1.07
Xenic Robinhood, $D_m = 16$	4.13	1.04
Xenic Robinhood, $D_m = 32$	4.84	1.02
Xenic Robinhood, no limit	6.39	1
FaRM Hopscotch, $H = 8$ [8]	> 8	1.04
DrTM+H Chained, $B = 4$	4.65	1.16
DrTM+H Chained, $B = 8$	8.81	1.10
DrTM+H Chained, $B = 16$	16.96	1.06

Table 2: Average number of objects read and number of roundtrips per lookup, at 90% occupancy.

well as transaction metadata for objects actively involved in transactions. For each segment of the host-side table, Xenic allocates a NIC *index entry*. An index entry contains a cache of objects that map to the corresponding host-side segment, transaction metadata (lock, version number) for those objects, and a *known displacement* value, d_i, for objects mapping to the corresponding host-side segment. Xenic implements a fixed-size set of cache positions for each index entry, with chained overflow pages allocated as necessary.

The NIC index also enables efficient host memory lookups when a cache miss occurs. Each NIC index entry maintains the highest known displacement d_i of objects mapping to the entry's host-side segment as well as an overflow address if objects in the segment have reached the displacement limit. Lookups require reading a region of the table in host memory, from the key's initial hash position to its actual displacement. While this actual displacement value is unknown until reaching the key, d_i serves as an effective location hint for locating a key with a single DMA read.

The NIC's d_i values may be invalidated by concurrent insertions performed in host memory: inserting one object may move another object beyond the corresponding d_i maintained at the NIC. To address this, the NIC reads $d_i + k$ additional elements beyond its known displacement, up to the limit D_m. While insertions will invalidate d_i values somewhat regularly (e.g., 6% of insertions at 90% occupancy), d_i is rarely increased by more than one (e.g., only 0.2% of insertions at 90% occupancy); therefore, we set $k = 1$ based on experimentation. If the NIC does not read the item within $d_i + k$ entries, the NIC performs a second, adjacent DMA read up to the limit, D_m. If d_i is already equal to D_m, the NIC instead reads the segment's overflow page.

Insertions, deletions, and cache eviction make use of the transaction protocol and its metadata to ensure consistency between the SmartNIC index and the host structure (§4.2).

4.1.4 Lookup Efficiency

We compare the efficiency of lookup operations to the hash designs of FaRM [8] and DrTM+H [44]. The three designs share similar priorities; each is optimized for remote hash lookups via remote memory access. All three designs perform updates using local memory operations at the target, and their insertion procedures prioritize placing objects mapping to the same hash value within a small, contiguous area of memory. This enables common-case remote lookups with one remote memory read at the cost of reading multiple objects per lookup. FaRM applies a Hopscotch hash table; like Robinhood, the Hopscotch table is a variant of linear probing. This design ensures that any element must be located within a fixed neighborhood size H, with $H = 8$ in FaRM's published results. A remote lookup first reads the neighborhood of H elements, and if the object is not found, issues a second read of the corresponding overflow bucket, resulting in an additional roundtrip. DrTM+H applies a simpler hash design, with a closed array of B-element fixed-size buckets and additional linked buckets allocated as necessary. A remote lookup traverses bucket links until finding the object.

We measure remote lookup performance at 90% table occupancy, comparing to FaRM's published results at the same occupancy. Table 2 shows the mean number of objects read and mean roundtrips per lookup for 8 million uniform-random keys. FaRM's design reads $H = 8$ objects per lookup in the common case, with a second roundtrip necessary for 4% of keys; average overflow read size is not published. DrTM+H reads at least B keys for each lookup, and due to its chained placement policy, often incurs multiple roundtrips traverse the chain. Xenic dynamically bounds the size of lookup reads based on hints stored in the NIC index. At high occupancy, and with a similar 4% overflow utilization to FaRM, Xenic's average read size is 48% lower than that of FaRM. If we disable overflow buckets and do not limit displacement, Xenic still achieves 20% fewer object reads per lookup than FaRM while also eliminating the overflow roundtrip.

FaRM and DrTM+H both perform remote hash lookups across the network via RDMA. Xenic instead performs remote lookups at the target-side NIC. RDMA lookups impose an end-to-end bandwidth cost and a full network roundtrip penalty if multiple reads are needed. Xenic's lookups, in contrast, consume only PCIe bandwidth and incur only PCIe access latency. Likewise, Xenic's remote lookups are always performed at the target NIC rather than at any remote client. This creates the opportunity for the NIC to cache objects and metadata for the host-side structure. While DrTM+H also applies index caching, it must store remote object addresses for each remote primary at each coordinator. DrTM+H's approach is limited in scalability, given its memory overhead, and lacks an efficient mechanism for cache invalidation.

4.2 Transaction Protocol

We first describe Xenic's distributed transaction commit procedure, then we detail special transaction cases and Xenic's respective optimizations. Xenic applies function shipping to

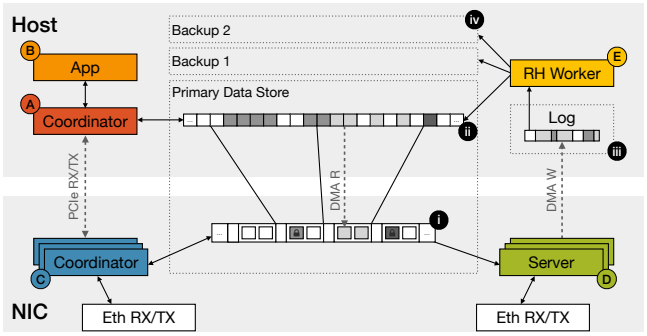

Figure 6: Xenic design overview, showing one server. Solid lines indicate local memory access; dashed lines indicate PCIe transfers.

offload execution logic from the host server to the coordinator-side SmartNIC (§4.2.2). Further, Xenic extends OCC with distributed, multi-hop protocol variants to increase communication efficiency based on a transaction's access pattern (§4.2.3). While Xenic targets distributed transaction workloads, where common-case transactions involve remote data shards, we ensure the SmartNIC optimizations do not add communication complexity to purely local transactions. §4.2.4 outlines Xenic's fast path for local transactions.

Figure 6 shows the components of each Xenic node. The coordinator application, running on each host, initiates all transactions and handles commits and aborts. We summarize the execution of a distributed, read-write transaction:

(1) A host coordinator thread A initiates the transaction, determining the initial read-set and write-set objects. The coordinator may either generate the transaction or poll a request queue from an external application thread B. It assigns a transaction ID (node index and sequence number), then sends the transaction state, including its read-set and write-set objects, to its local SmartNIC.

(2) A coordinator-side SmartNIC thread C then issues remote Execute requests to each primary involved, specifying the shard's read-set and write-set keys. The request is received by a server-side SmartNIC thread D, which performs a lookup for each read- and write-set key in its local-memory index i. If any key exists and is locked, the NIC returns an *abort* response. Otherwise, the NIC allocates an index entry, if necessary, and acquires a lock for each write-set key. The NIC then retrieves the values and version numbers of the read-set keys. As described in §4.1, cached values are retrieved from SmartNIC memory i, and cache-miss reads retrieve the value from host memory ii via PCIe DMA. Finally, the coordinator sends a response containing the read-set values and versions.

(3) After receiving successful Execute responses from all primaries, the coordinator SmartNIC updates its transaction state with the returned read-set values and sends the transaction state via PCIe to the host. The host coordinator performs an application-level function to generate write-set values given the read-set values and sends these writes to its NIC. For a multi-shot transaction, the coordinator may issue subsequent execute requests to read and/or lock additional keys until execution is finished.

(4) The coordinator-side SmartNIC issues a Validate request to the primary of each read-set key, except for those locked for writing. The request includes the version number for each key obtained by Execute. The primary NIC retrieves the current version for each key in its index i, and returns *commit* if the version numbers match and no keys are locked. Otherwise, the NIC returns *abort*, which is propagated to the application and other primaries.

(5) After receiving successful Validate responses, the transaction completes if it is read-only. For read-write transactions, the coordinator replicates the write set to each shard's backup replicas using a Log request with the shard's key-values and version numbers. The NIC for each backup handles the Log request by appending it to a hugepage of host memory reserved for logging iii via DMA write. The NIC responds after the DMA completes.

(6) After receiving all Log responses, the coordinator-side NIC reports a *Committed* outcome to the host, and issues a Commit request to each primary, with write-set key-values and version numbers. The primary NIC appends the Commit request to the host-memory log iii. Then, it applies the new values to cached entries in the index and updates the write version numbers. Once the DMA completes, the NIC releases the write-set locks and sends an ack response. The write-set objects are pinned in the NIC's index cache and cannot yet be evicted. This ensures NIC lookups will not read a stale object before the host applies the Commit writes to its hash structure.

(7) The host-side Robinhood worker threads E poll the log for entries written by the NIC. The host threads asynchronously handle requests by applying Log write sets to the backup shards iv in host memory and Commit write sets to the primary shard ii. The host application appends a log ack to traffic between the host and the NIC, allowing the NIC to reclaim log space and unpin cache entries for committed writes.

4.2.1 Fault Tolerance Xenic applies the reconfiguration and recovery design of FaRM without additional requirements. To do so, we ensure that (a) lock state is maintained in only one location (SmartNIC memory) and rebuilt upon recovery, (b) Xenic's host-side hash table maintains the same set of objects as that of a static hash table, and (c) operations are executed in the same sequence across the coordinator, primaries, and backups as in FaRM's protocol, with log records

written to host memory before a Log operation or Commit operation returns an acknowledgement.

Given these similarities, FaRM's recovery protocol applies to Xenic as follows. Xenic uses a typical Zookeeper-based cluster manager to determine membership. Each node holds a lease with the cluster manager, and lease expiration triggers reconfiguration. Only primaries maintain lock state, so when a primary fails, a backup is promoted to become the new primary, and the lock state is reconstructed. While other shards may proceed, each node of the recovering shard scans its log for transactions that have not yet been acknowledged as committed to the primary. These recovering transactions' write-set keys are communicated to the new primary, which acquires locks on each object. Once all locks are set, the shard can serve new transactions. Meanwhile, the replicas communicate to ensure each recovering transaction is either aborted or fully applied to all replicas before its associated locks are finally released.

4.2.2 SmartNIC Function Shipping Offloading execution logic from the host to the coordinator-side NIC provides an opportunity for latency reduction, eliminating all but one coordinator PCIe roundtrip. We apply function shipping [7–9]; while FaRM used it between hosts, we use it to move execution from the host to the NIC. Xenic implements function shipping by adding an optional, application-defined data field to each transaction state entry maintained on the NIC. This data consists of the application's *external state*, if any, required for a transaction's execution. Second, Xenic provides an abstract interface for execution logic. This interface exposes the transaction's read and write sets and the external state associated with the transaction. When a transaction request is initially sent from the host to its coordinator-side NIC, any external state data is attached to the request and buffered at the NIC. When the transaction's Execute responses are received by the coordinator-side NIC, the execution logic is invoked, transforming the transaction's read and write sets based on the current objects and external state. If execution adds keys to the transaction, coordinator issues Execute requests for the new keys, collects responses, and repeats the execution function. Otherwise, the coordinator proceeds to commit the transaction. Offloading execution requires performing execution logic on the NIC and sending associated application state to the NIC, potentially incurring additional NIC CPU load and PCIe bandwidth utilization. Offloading execution is feasible only when the object manipulation is not computationally intensive and the application state is small, i.e., when it does not introduce the NIC cores or PCIe bandwidth as a performance bottleneck.

4.2.3 Multi-Hop OCC Communication Xenic additionally applies function shipping to reduce commit protocol

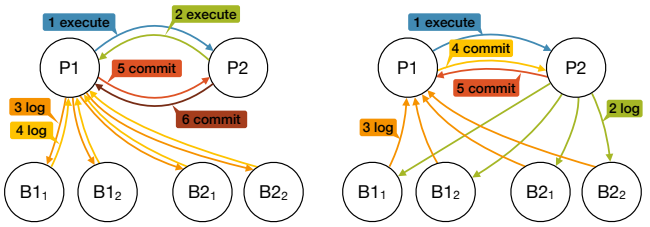

Figure 7: Commit messages for a transaction writing to local and remote shards, with execution (a) performed at the coordinator *P1*, and (b) shipped to remote *P2* to minimize communication.

network communication. By leveraging point-to-point operations between NICs, in addition to the standard coordinator-server pattern, Xenic can reduce commit messages and message delays based on a transaction's access pattern. When the coordinator-side NIC receives a transaction request, it determines the optimal execution node based on any remote accesses in the read and write sets. If commit communication can be simplified by performing execution at a remote primary NIC, Xenic applies function shipping to invoke execution remotely and uses multi-hop requests to reduce communication. For instance, transactions writing to the local shard and one remote shard are executed at the remote primary NIC. Figure 7 shows communication with coordinator execution and only request-response operations (a) and the optimized communication pattern enabled by shipping execution to the remote primary NIC (b). In the optimized case, the *P2* NIC performs execution, then issues Log requests to the backup NICs, and the backups send Log responses to the coordinator-side *P1* NIC. By shipping execution to the remote primary NIC, Xenic eliminates a network message delay from the commit protocol. Xenic handles remote execution with the same function shipping mechanism as in the coordinator offload optimization. We limit remote execution to transactions involving a single execution round, where all keys are specified in the initial request. We implement multi-hop commit operations for all single-shard transactions and transactions involving the local shard and one remote shard.

4.2.4 Local Transactions Local write transactions execute optimistically on the host, accessing objects in the host-side hash structure. After execution, the host sends the transaction state to its coordinator-side SmartNIC for replication. Before issuing Log requests, the coordinator-side NIC acquires write-set locks in its index and aborts if any lock is already held. Otherwise, the NIC proceeds with the commit protocol. This adds no network or PCIe overhead for committed transactions. Local read transactions require no PCIe communication, performing reads and Validate logic locally at the host-side hash table.

4.3 SmartNIC Operations Framework

We apply our performance analysis to design an efficient framework for Xenic's SmartNIC commit operations. First,

our results show that PCIe DMAs have significant submission and completion latency (§3.5). To achieve high core utilization, NIC cores must perform work while awaiting DMA completion. Second, we find substantial throughput opportunities in batching DMA submissions (§3.5). Submitting full vectors to the DMA engine amortizes submission cost, without adding completion latency, and increases maximum throughput. Combining batched DMA submission with batched Ethernet transmission (§3.4) results in high network utilization, with potential for improved throughput over one-sided RDMA. Individual commit operations, however, typically do not fill a 15-buffer DMA vector or an Ethernet MTU and do not have work to perform while awaiting DMA completion. For this reason, we must interleave commit operations and aggregate work at the point of DMA submissions, NIC-to-NIC, and NIC-to-host communications.

4.3.1 Asynchronous Operations Xenic implements continuation-passing, asynchronous operations to interleave work, and to minimize blocking for DMA completions. Each NIC core maintains two vectors for pending read and write DMAs, respectively. Transaction operations insert entries (NIC/host addresses, size) into the read and write vectors, along with a callback function to be executed upon DMA completion. This callback may produce a network output, e.g., a Log acknowledgement, or further manipulate NIC state, e.g., unlocking objects after Commit writes are transferred to host memory. When a NIC core is idle, or when the DMA vector fills, it is submitted to the core's assigned DMA engine. The DMA engine writes a completion status byte once it has performed the DMA. Each core tracks in-flight DMAs using a core-local ring buffer, mapping completion byte addresses to the associated batch of callback work.

4.3.2 Opportunistic Batching Xenic runs a burst-oriented polling loop on each NIC core, applying the NIC's hardware flow engine to route flows to cores. Each loop iteration handles a burst of Ethernet traffic and a burst of DMA completions, accumulating DMA requests and their callbacks in the pending read/write vectors. After handling the burst, the NIC submits any DMA requests and collects all outbound NIC-to-host and NIC-to-NIC packet transmissions. The NIC core uses a gather-list for each destination and performs an aggregated Ethernet or PCIe packet transmission. This allows Xenic to combine as many outputs as possible into each packet.

Xenic's batching approach allows the SmartNIC to aggregate communication whenever there is sufficient traffic between two nodes. PCIe communications are batched separately, and do not always achieve full batches; for instance, a read-heavy workload largely served by the SmartNIC cache results in few PCIe accesses. However, this scenario does not result in lower performance because cache hits to SmartNIC memory are lower-cost than DMA lookups.

4.3.3 Limited SmartNIC Resources The SmartNIC's compute and memory capacities are small relative to the host server. Xenic is designed with this in mind, allowing workloads to appropriately utilize the SmartNIC. First, Xenic selectively applies function shipping to execute transactions on NIC cores. This is applied on a per-transaction basis via a user annotation. Doing so allows the NIC to execute latency-critical transactions, reducing PCIe crossings, while the host executes compute-heavy or predominantly local transactions. Second, Xenic uses SmartNIC memory to cache objects, adapting to available capacity. When caching is ineffective, due to the access pattern or cache eviction policy, the need for DMA lookups increases. These misses incur PCIe bandwidth overhead (§4.1.2), potentially becoming a bottleneck. Decreasing probing distance sufficiently, say by expanding the host-side hash memory, reduces PCIe overhead and allows lookups to reach network throughput.

4.3.4 Other SmartNIC Platforms Xenic relies on SmartNIC hardware characteristics to reduce latency. First, handling a remote request on the SmartNIC must be lower latency than doing so with a host RPC. Some off-path SmartNICs demonstrated higher latency when directing traffic to the SmartNIC cores versus sending requests directly to the host with an RDMA NIC (§3.1). Second, the SmartNIC must have an efficient mechanism for host memory access. SmartNICs that rely on an RDMA interface between the NIC cores and host memory showed prohibitively high latency, precluding Xenic's latency reduction goal. If the SmartNIC hardware does not show latency reduction potential, using SmartNICs may not be justifiable over a host-only design. In contrast, with a platform meeting these requirements, Xenic can improve both latency and throughput.

5 Evaluation

We implement Xenic using LiquidIO 3 SmartNICs. We extend the generic NIC firmware to add transaction-processing logic, written in C using DPDK and the LiquidIO hardware interfaces. The host-side coordinator also uses DPDK. Our testbed consists of 6 servers, each with Intel Xeon Gold 5218 CPUs (16 cores, 32 hyperthreads, 2.3GHz) and 96GB DDR4 DRAM. Each server contains a 2x50GbE Marvell LiquidIO 3 (CN3380), with 24 2.2GHz ARM cores, 16GB DDR4 DRAM, and a PCIe 3.0 x8 interface. We utilize both links of the NIC for a per-server total network bandwidth of 100Gbps. Each server also contains a 100GbE Mellanox CX5 (MCX516A-CCAT) NIC with a PCIe 3.0 x16 interface for comparison.

5.1 Comparisons

We evaluate Xenic with case studies of the TPC-C [42], Retwis [38, 41, 47], and Smallbank [13] benchmarks. We

use each benchmark to compare performance against recent work in hardware-accelerated distributed transactions, measuring per-server average throughput and median latency. We focus on versions of DrTM+H [44] for this comparison, a well-optimized research system applying a hybrid of one-sided RDMA and two-sided RPCs to maximize performance. In addition to its hybrid design, DrTM+H provides additional versions representing alternate decisions in the RDMA design space. We compare the following configurations:

- *DrTM+H* is the best-case combination of one-sided and two-sided operations for each protocol phase. One-sided operations are typically used for execution reads, validation, and logging. DrTM+H avoids remote data structure traversals by caching addresses for remote objects.
- *DrTM+H with no remote caching (NC)* matches DrTM+H but disables the coordinator's remote address cache. This configuration demonstrates the impact of RDMA hash traversal for Execute reads.
- *FaSST* involves two-sided RPC operations exclusively, emulating the design by Kalia *et al.* [15]. This version performs remote data structure lookups via host RPC, and where possible, consolidates multiple operations (e.g., reading and locking) into individual RPCs.
- *DrTM+R.* This configuration emulates DrTM+R's use of one-sided RDMA, retaining DrTM+H's OCC protocol [44].

Of the open-source related work, only DrTM+H implements the TPC-C benchmark. However, DrTM+H's support is limited to a simplified version of the TPC-C workload, consisting of *new order* transactions, instead of the typical mix of five types, and using a customized access pattern. We evaluate Xenic using this workload for comparison with the DrTM+H configurations (§5.2). Xenic supports the full TPC-C workload, which we evaluate separately (§5.3). DrTM+H provides a Smallbank implementation; we migrate their code to Xenic and implement Retwis on both systems. For the three benchmarks, we discuss host and NIC resource utilization (§5.6). Finally, we evaluate key aspects of Xenic's design and their contributions to throughput and latency (§5.7).

5.2 Case Study: TPC-C New Order

The TPC-C benchmark simulates a warehouse order processing system with nine tables and a range of object sizes up to 660B. We first evaluate the performance of TPC-C's *new order* transaction, the predominant transaction of the five types in the TPC-C specification. Because DrTM+H only supports the *new order* transaction, not the full workload, we use this benchmark to compare performance with DrTM+H and evaluate the full workload mix in §5.3. Each *new order* selects 5-15 items, updates stock counts, and writes order line-item records. The coordinator picks items from partitions chosen uniformly at random; this matches the DrTM+H authors' evaluation, creating a strenuous remote access pattern. Three of the tables are accessed by transactions across the cluster, while the others are B+ trees local to their respective coordinators; all tables are replicated. We deploy TPC-C on the 6-server testbed with a replication factor of 3 (2 backups for each primary) at the scale of 72 warehouses per server. Figure 8a shows the results.

Xenic achieves an average peak throughput of 1.19M txn/s per server, a 2.42× improvement over DrTM+H, the best alternative. While both systems saturate network bandwidth, DrTM+H requires multiple network operations for each TPC-C stock object to retrieve the value, then lock and validate. Xenic can lock and read a remote object in one remote operation, reducing bandwidth consumption and latency. Xenic effectively aggregates work at the SmartNIC, further allowing throughput to scale. Xenic's throughput is 3.81× greater than DrTM+H with coordinator-side caching disabled, showing the overhead of DrTM+H's remote lookups. Although RPCs avoid these one-sided RDMA inefficiencies, handling all operations with host RPCs limits FaSST's throughput to 232k txn/s, even when utilizing all host threads.

At low load, Xenic's median latency is 59% below that of DrTM+H, the lowest-latency alternative. While DrTM+H applies one-sided RDMA for reads, this requires separate remote operations to read, lock, and validate a remote object, limiting latency savings. Xenic can perform these functions with a single remote request while reducing latency relative to a host RPC. The latency penalty of FaSST's RPC approach is high for this benchmark since FaSST handles RPCs on the same threads performing compute-intensive B+ tree operations. At 95% of peak throughput, FaSST shows high latency: 2.2× that of DrTM+H and 4.0× that of Xenic.

5.3 Case Study: TPC-C

The full TPC-C workload consists of five transaction types, including *new order*. We deploy the full workload mix at the same scale as §5.2, configured to match the standard benchmark specification. Like prior implementations [45], we chop the long-running local transaction logic into multiple database transactions. Xenic ships execution of the *new order* and *payment* transactions to the NIC; other transactions execute on the host. Per the specification, we measure throughput as the rate of *new order* transactions per second within the full workload mix; this is approximately 45% of the overall transaction throughput. Figure 8b shows the result.

Xenic achieves peak throughput of 541k new orders per second per server, saturating the network. With *new order* transactions comprising 45% of the workload, the other transactions consume bandwidth and limit throughput to approximately half that of the *new order* workload in §5.2. In the standard TPC-C configuration, only ~10% of *new order* and 15% of *payment* transactions access a remote warehouse's

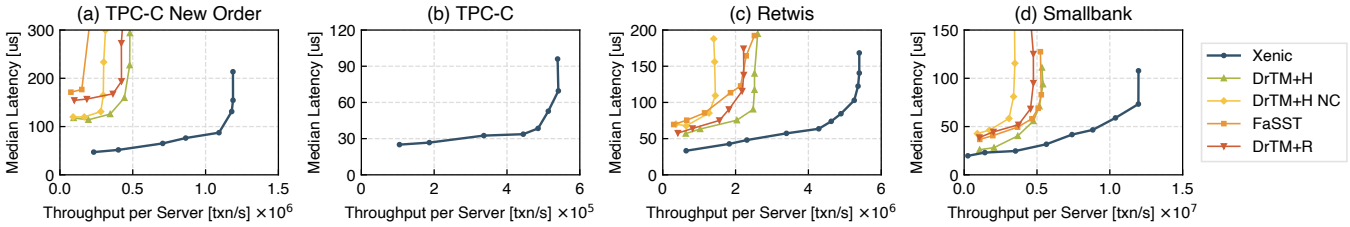

Figure 8: Throughput per server and median latency for (a) TPC-C New Order, (b) TPC-C, (c) Retwis, and (d) Smallbank benchmarks.

objects; this results in a median latency of 25μs at low load, below that of the modified *new order* workload.

Of the related work, DrTM+R and FaRM implement the full TPC-C workload. While neither system is open-source, DrTM+R's authors provide a throughput evaluation at the same scale as our testbed: 6 servers with 3-way replication [6]. With a 56Gbps network, DrTM+R's evaluation reports 150k new orders per second per server, fully utilizing the network bandwidth (a higher per-server throughput than FaRM). Because DrTM+R's throughput is limited by network bandwidth, we deploy Xenic with a similar network configuration to compare throughput with this published result. For Xenic, we use one 50Gbps link per server, instead of two, and run TPC-C at a scale of 384 warehouses to match DrTM+R. In this experiment, Xenic achieves a peak throughput of 322k new orders per second per server, 2.1× higher than DrTM+R. This is a smaller increase than Xenic's 2.7× improvement for the modified *new order* workload. The full TPC-C workload involves a higher frequency of local transactions, which only utilize the network for replication (Log operations). Xenic does not improve efficiency for these transactions relative to DrTM+R.

5.4 Case Study: Retwis

We evaluate the Retwis benchmark [38, 47], representing a Twitter-like application. The benchmark includes a mix of transaction types, with 50% read-only transactions and 1-10 keys per transaction. Unlike TPC-C, minimal coordinator-side computation is involved in performing transactions. Relative to Smallbank, objects are moderately larger (64B versus 4B values), accessed with a Zipf distribution, α = 0.5, with a higher proportion of read-only transactions. We deploy Retwis with a replication factor of 3 and 1 million keys per server. Figure 8c shows the results.

Xenic shows a 2.07× peak throughput increase relative to DrTM+H and 42% lower median latency at low load. As with TPC-C, both systems fully utilize network bandwidth, while Xenic achieves higher efficiency. DrTM+H's hybrid design improves the performance of Retwis' read-only transactions, but its use of one-sided RDMA multiplies the number of requests for read-write transactions. This imposes a throughput and latency cost; we evaluate this impact on Retwis throughput in §5.7. Given the minimal computation involved in the benchmark, FaSST nears the peak throughput of DrTM+H without fully utilizing the host CPU. However, its RPC design results in consistently higher latency, with a minimum median latency 2.12× higher than that of Xenic.

5.5 Case Study: Smallbank

The Smallbank benchmark represents simple transactions on a database of account balances, with small 12B objects. 15% of transactions are read-only, and the remainder involves additions and subtractions of balances, with up to 3 keys per transaction. 90% of transactions access 4% of keys, resulting in relatively low contention. We deploy Smallbank at a comparable scale to our related work: 2.4M accounts per server, with a replication factor of 3. Figure 8d shows the results.

We observe a peak throughput of 12.0M txn/s per server with Xenic, 2.21× the maximum throughput of DrTM+H. Both systems saturate network bandwidth at peak throughput. Xenic delivers throughput improvement through protocol and communication efficiency. Smallbank's workload of 12B key-value objects presents a significant opportunity for batching. Given the small object sizes, minimizing the metadata overhead of each remote request is especially critical for bandwidth efficiency. The software flexibility of Xenic's commit operations enables higher bandwidth utilization, and its aggregation of remote requests enables aggressive batching.

However, Smallbank's small remote operations also demonstrate the best-case latency potential of one-sided RDMA, and DrTM+H performs optimal one-sided READs due to its pointer cache. Xenic shows 21.5% lower minimum median latency than DrTM+H, achieving competitive performance by eliminating PCIe accesses, utilizing NIC memory for transaction metadata, and caching hot objects. As in the other benchmarks, Xenic's commit protocol requires fewer remote operations per key than that of DrTM+H. For most Smallbank transactions, Xenic reduces communication via function shipping; we evaluate this optimization in §5.7.

5.6 SmartNIC Resource Utilization

To study utilization, we measure the minimum number of cores to run each benchmark at peak throughput, with Xenic, DrTM+H, and FaSST. We run each benchmark and decrease thread count until throughput drops below 95% of its maximum. For Xenic, we repeat this analysis with NIC cores.

Benchmark	Xenic Norm. (Host, NIC)	DrTM+H	FaSST
TPC-C NO	21.7 (18, 12)	24	32
Retwis	9.9 (5, 16)	18	24
Smallbank	9.9 (5, 16)	20	28

Table 3: Normalized thread count, for Xenic, DrTM+H, and FaSST. NIC thread count is scaled by NIC/host Coremark score ratio.

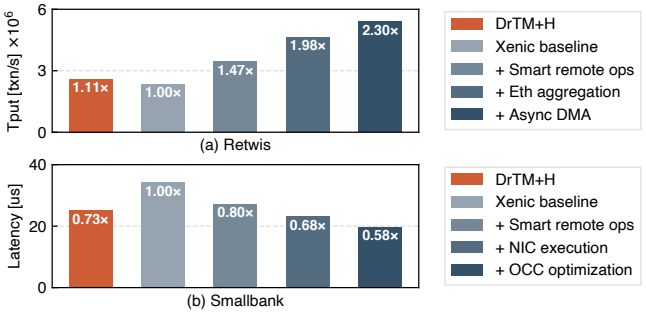

Figure 9: Retwis per-server throughput (a) and Smallbank median latency (b), sequentially enabling key aspects of Xenic's design.

Table 3 shows the result. We find that Xenic requires few host threads for Retwis and Smallbank: 2 application threads to initiate transactions and handle completions, and 3 worker threads to apply writes to the primary and backup tables. TPC-C, however, requires 18 host threads due to its compute-intensive local B+ tree manipulations, which are performed on both host application threads and worker threads (to apply updates to each backup). Smallbank and Retwis offload all execution to the NIC, resulting in higher NIC utilization; TPC-C instead shows higher host utilization.

To compare cumulative utilization across host and NIC processors, we use the Coremark benchmark to normalize computation power. We use the ratio of the NIC's per-thread Coremark score to that of the host: 0.31×. This clearly is an approximation as the relative power is workload-dependent (§3.6). With this approximation, we report that relative to DrTM+H, Xenic saves 2.3 threads for TPC-C, 8.1 threads for Retwis, and 10.1 threads for Smallbank. In all cases, Xenic achieves higher throughput and core savings relative to FaSST and DrTM+H. Xenic's lower utilization suggests that exploiting wimpy NIC cores close to the NIC's hardware interfaces enables higher overall computation efficiency.

5.7 Impact of Optimizations

To evaluate how Xenic's design features contribute to improvements in throughput and latency, we begin with a baseline design and sequentially enable features. The Xenic baseline resembles DrTM+H, implementing the same set of remote operations. We impose the same restrictions that arise from DrTM+H's use of one-sided RDMA; in particular, we use separate requests to read, lock, and validate objects.

In Figure 9a, we enable a series of throughput-oriented optimizations and measure their impact on Retwis' throughput relative to the baseline and to DrTM+H. Despite their similar protocol, the Xenic baseline shows 10% lower throughput than DrTM+H. The NIC cores are saturated, with the NIC's software packet processing limiting throughput. Adding Xenic's optimized remote commit operations reduces the number of remote requests; this increases throughput by 1.47×. Adding aggregated Ethernet transmissions facilitates higher bandwidth utilization, for an overall 1.98× increase. Finally, we enable asynchronous NIC execution, batching DMAs across multiple operations, to amortize overhead and minimize blocking time. This results in a cumulative 2.30× peak throughput increase, 2.07× relative to DrTM+H.

Next, we evaluate latency-oriented optimizations and their impact on Smallbank's median latency. Figure 9b shows these measurements. Relative to DrTM+H, the Xenic baseline latency is 1.37× higher. As in §3.2, the LiquidIO demonstrates consistently higher latency than the CX5 for comparable remote memory accesses, explaining this latency difference. Enabling Xenic's optimized commit operations, reducing the number of requests involved per transaction, improves latency by 20%. By shipping execution to the coordinator SmartNIC, Xenic eliminates intermediate coordinator-side PCIe traversals during each transaction, further reducing latency, 32% below the baseline. Smallbank's workload of 1-2 shard transactions presents the opportunity to further reduce latency by shipping execution to remote SmartNICs and applying optimized communication patterns. This achieves a 42% latency reduction over the baseline, 22% below DrTM+H.

6 Conclusion

We argue that SmartNICs offer an opportunity for high-performance, hardware-accelerated distributed transactions, without the trade-offs that define RDMA systems. Using measured performance characteristics to inform our design, we build Xenic, a transaction processing system leveraging on-path SmartNICs. Xenic employs a co-designed data store spread across the NIC and the host, an asynchronous and batched execution model, and flexible communications to improve efficiency. With three benchmarks comprising a range of workloads, we compare Xenic against RDMA-based systems. Our results show that despite software overheads relative to RDMA, Xenic effectively applies the SmartNIC to increase throughput and reduce latency.

Acknowledgements. We would like to thank the anonymous reviewers and our shepherd, Natacha Crooks, for their comments and feedback. This work is supported by NSF grants (CNS-2028771, CNS-2006349, and CNS-2106199) and Futurewei.

References

[1] Alpha Data. ADM-PCIE-9V3 - High-Performance Network Accelerator, Sept. 2021. https://www.alpha-data.com/dcp/products.php?product=adm-pcie-9v3.

[2] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff. Enabling programmable transport protocols in high-speed nics. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 93–109, Santa Clara, CA, Feb. 2020. USENIX Association.

[3] Broadcom. The TruFlow Flow processing engine. https://www.broadcom.com/applications/data-center/cloud-scale-networking, 2021.

[4] Broadcom Inc. Stingray SmartNIC Adapters and IC, Sept. 2021. https://www.broadcom.com/products/ethernet-connectivity/network-adapters/smartnic.

[5] P. Celis, P. Larson, and J. I. Munro. Robin hood hashing (preliminary report). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 281–288. IEEE Computer Society, 1985.

[6] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen. Fast and general distributed transactions using rdma and htm. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.

[7] D. W. Cornell, D. M. Dias, and P. S. Yu. On multisystem coupling through function request shipping. *IEEE Transactions on Software Engineering*, SE-12(10):1006–1017, 1986.

[8] A. Dragojevic, D. Narayanan, M. Castro, and O. Hodson. Farm: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014)*. USENIX – Advanced Computing Systems Association, April 2014.

[9] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 54–70, New York, NY, USA, 2015. Association for Computing Machinery.

[10] Exablaze. ExaNIC V5P High Density Network Application Card, Sept. 2021. https://exablaze.com/exanic-v5p.

[11] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018.

[12] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren. Smartnic performance isolation with fairnic: Programmable networking for the cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2020.

[13] H-Store Project. SmallBank Benchmark - H-Store, Sept. 2021. https://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/.

[14] S. Ibanez, M. Shahbaz, and N. McKeown. The case for a network fast path to the cpu. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 52–59, New York, NY, USA, 2019. Association for Computing Machinery.

[15] A. Kalia, M. Kaminsky, and D. G. Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram rpcs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 185–201, Savannah, GA, Nov. 2016. USENIX Association.

[16] A. Kaufmann, S. Peter, N. K. Sharma, T. Anderson, and A. Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[17] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. Swift, and T. Lakshman. Uno: uniflying host and smart nic offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 506–519, 09 2017.

[18] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[19] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[20] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella. PANIC: A high-performance programmable NIC for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020.

[21] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, 2019.

[22] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya. Incbricks: Toward in-network computation with an in-network cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[23] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana. E3: Energy-efficient microservices on smartnic-accelerated servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 363–378, Renton, WA, July 2019. USENIX Association.

[24] Marvell Technology Group Ltd. LiquidIO III Solutions Brief, Sept. 2021. https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf.

[25] Marvell Technology Group Ltd. Multi-Core Processors - LiquidIO Smart NICs | Network adapter, Sept. 2021. https://www.marvell.com/products/infrastructure-processors/multi-core-processors/liquidio-smart-nics.html.

[26] F. Matus. Pensando: Distributed services architecture. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–17. IEEE Computer Society, 2020.

[27] Mellanox. Accelerated Switch and Packet Processing. http://www.mellanox.com/page/asap2?mtag=asap2, 2021.

[28] Mellanox. BlueField SmartNIC Ethernet, Sept. 2021. https://www.mellanox.com/products/BlueField-SmartNIC-Ethernet.

[29] Mellanox. ConnectX-5 EN Single/Dual-Port Adapter, Sept. 2021. https://www.mellanox.com/products/ethernet-adapters/connectx-5-en.

[30] Mellanox. Mellanox Innova SmartNIC. http://www.mellanox.com/page/products_dyn?product_family=275&mtag=bluefield_smart_nic, 2021.

[31] Mellanox. OFED Documentation Rev 7.4.1.0.0.1, Sept. 2021. https://docs.mellanox.com/display/MLNXOFEDv471001.

[32] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1–18, Santa Clara, CA, Feb. 2020. USENIX Association.

[33] Netronome. Agilio LX SmartNICs, Sept. 2021. https://www.netronome.com/products/agilio-cx/.

[34] Pensando. Pensando DSC-100 Distributed Services Card, Sept. 2021. https://pensando.io/documents/pensando-dsc-100-distributed-services-card/.

[35] Pensando floor plan. https://www.servethehome.com/pensando-distributed-services-architecture-smartnic/, 2021.

[36] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson. Floem: A programming system for nic-accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018.

[37] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat. SENIC: Scalable NIC for end-host rate limiting. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 475–488, Seattle, WA, Apr. 2014. USENIX Association.

[38] Redis. Retwis - Example Twitter clone based on the Redis Key-Value DB, Sept. 2021. http://retwis.redis.io.

[39] B. Stephens, A. Akella, and M. Swift. Loom: Flexible and efficient NIC packet scheduling. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 33–46, Boston, MA, Feb. 2019. USENIX Association.

[40] B. Stephens, A. Akella, and M. M. Swift. Your programmable nic should be a programmable switch. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, page 36–42, New York, NY, USA, 2018. Association for Computing Machinery.

[41] A. Szekeres, M. Whittaker, J. Li, N. K. Sharma, A. Krishnamurthy, D. R. K. Ports, and I. Zhang. Meerkat: Multicore-scalable replicated transactions following the zero-coordination principle. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Transaction Processing Performance Council. TPC Benchmark C Standard Specification, Revision 5.11, Sept. 2021. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf.

[43] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 18–32, New York, NY, USA, 2013. Association for Computing Machinery.

[44] X. Wei, Z. Dong, R. Chen, and H. Chen. Deconstructing rdma-enabled distributed transactions: Hybrid is better! In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 233–251, Carlsbad, CA, Oct. 2018. USENIX Association.

[45] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 87–104, New York, NY, USA, 2015. Association for Computing Machinery.

[46] Xilinx. Alveo Adaptable Accelerator Cards for Data Center Workloads, Sept. 2021. https://www.xilinx.com/products/boards-and-kits/alveo.html.

[47] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building consistent transactions with inconsistent replication. *ACM Trans. Comput. Syst.*, 35(4), Dec. 2018.