Evaluating RDMA for Distributed Cache Reads and Metadata Updates

Aengus McGuinness
CS2640 Final Project

Abstract

Distributed caches improve application latency by serving hot objects from memory, but every cache access still incurs server CPU work and often updates eviction metadata. This project asks how much of that overhead can be removed with Remote Direct Memory Access (RDMA), and how much of the benefit remains when read operations also perform lightweight metadata updates. I built RDMA Cache Prototype, a minimal key-value cache with three communication paths: a TCP/RPC baseline, a two-sided RDMA send/receive path, and a one-sided RDMA read path over a remotely readable hash table. The one-sided path can optionally issue an RDMA `FETCH_AND_ADD` after each read to approximate LRU-style recency metadata maintenance. In a final three-trial CloudLab experiment, two-sided RDMA improves throughput by 5.5–13.2× over TCP and reduces p99 latency from 132–164 μs to 6–8 μs. One-sided reads reach 160k–974k operations/s with stable 12 μs p99 latency. Metadata atomics increase one-sided mean latency by about 3.0–3.2 μs and p99 latency by 3–4 μs. These results show that one-sided RDMA can remove most server-side cache read overhead, but metadata maintenance remains a visible part of the critical path.

1 Introduction

Distributed caching systems such as Memcached, Redis, and Valkey are a common way to reduce database load and improve tail latency. They are simple in concept: clients send key-value requests to a cache server, and the server returns values from memory. At high request rates, however, even simple cache reads are not free. Each request enters the server through the operating system network stack, wakes application code, parses a command, performs a lookup, formats a response, and sends data back to the client. A read may also update eviction metadata, such as an LRU order or access counter.

RDMA changes this design space. With one-sided RDMA, a client can read or write registered memory on a remote machine without involving the remote CPU. If cache objects are laid out so clients can compute where values live, then cache reads can bypass the server's request-processing loop entirely. The main complication is metadata. Practical caches rarely perform pure reads, since they also track object recency, frequency, or admission information. These metadata updates can reintroduce remote communication and synchronization.

This project evaluates that trade-off with a small but working prototype. The original proposal defined three goals. The 75% goal was to implement a basic key-value store and compare TCP/RPC to two-sided RDMA messaging. The 100% goal was to add one-sided RDMA reads. The 125% goal was to simulate cache metadata updates and measure their impact on one-sided RDMA. RDMA Cache Prototype implements all three configurations and collects throughput, latency, server CPU, and attempted network-counter measurements on CloudLab.

This report makes three contributions:

- It implements a common key-value workload across TCP, two-sided RDMA, and one-sided RDMA, while preserving comparable request semantics where the transport allows it.
- It defines a fixed RDMA-readable hash table layout that lets clients issue one-sided reads without server CPU participation.
- It measures the latency cost of adding RDMA atomic metadata updates to the one-sided read path.

Overall, the results show that RDMA reduces cache-read overhead, while metadata updates remain a visible cost.

2 Background and Goals

2.1 Communication Modes

TCP/RPC. The baseline follows the usual client-server model. Clients send textual `GET`, `SET`, `DEL`, and `QUIT` commands over TCP. The server parses each request, calls a local key-value

store, and sends a text response. This mode represents the traditional RPC-style distributed cache structure.

Two-sided RDMA. Two-sided RDMA still involves both machines: the client posts an RDMA `SEND`, and the server must have posted a matching `RECV`. The server CPU still parses and handles each request, but the transport bypasses the normal TCP socket data path. This isolates the benefit of RDMA messaging.

One-sided RDMA. One-sided RDMA lets the client issue an RDMA `READ` directly against the server's registered memory. Once the queue pair and memory region are established, the data path does not require server CPU execution. This mode matches the central read-heavy cache optimization from the proposal.

2.2 Metadata Updates

The difficult part of one-sided caching is not fetching bytes; it is preserving cache policy. LRU-like policies update metadata on every access. A complete distributed LRU list would require ordering, synchronization, and consistency protocols. For this project, I use a narrower but measurable proxy: each cache slot contains an 8-byte `access_count`. The metadata variant of the one-sided benchmark performs one RDMA `FETCH_AND_ADD` after each RDMA read. This does not implement full LRU, but it captures the essential cost that every read must also modify remote metadata.

3 Design

3.1 System Overview

RDMA Cache Prototype is organized into storage, protocol, transport, and benchmark layers. The TCP and two-sided RDMA modes use the same text protocol and same `KeyValueStore` logic. The one-sided mode uses a different storage layout because remote clients need deterministic address arithmetic.

3.2 TCP Baseline

The TCP server uses a single-threaded coroutine event loop built on `poll(2)`. It accepts multiple client connections, sets sockets to nonblocking mode, buffers partial reads and writes, and resumes connection coroutines when descriptors become ready. The store is a `std::unordered_map` protected by a mutex. This implementation is small, but it avoids a thread-per-connection design that would make the TCP baseline less comparable to a high-performance network server.

3.3 Two-Sided RDMA

The RDMA transport is implemented using `libibverbs`. Each connection owns a Reliable Connected queue pair, a completion queue, a protection domain, and pinned send/receive buffers. A TCP side channel exchanges queue-pair metadata: QP number, LID, GID, and, in one-sided mode, the remote key and base address of the exported memory region. After the exchange, the code moves the queue pair through `INIT`, `RTR`, and `RTS`. The two-sided server then repeatedly receives text requests, runs the same protocol handler used by TCP, and sends text responses.

3.4 One-Sided Memory Layout

One-sided reads require the client to know where a key may reside without asking the server. The one-sided store is therefore a flat open-addressing hash table with 4096 fixed-size slots. Each slot is exactly 1024 bytes and is aligned to 1024 bytes. The full registered region is 4 MiB.

Table 1: RDMA-readable slot layout.

Offset	Size	Field
0	1 B	`occupied` state
1	7 B	padding
8	8 B	`access_count` metadata
16	112 B	null-terminated key
128	896 B	null-terminated value
Total	1024 B	one `RdmaSlot`

The client replicates the server's FNV-1a hash function and linear-probing logic. For a `GET`, it computes the initial slot index, issues an RDMA read for the slot, checks the occupied byte and key, and continues probing if necessary. The metadata path then issues an RDMA `FETCH_AND_ADD` to the slot's `access_count` field at offset 8. That offset satisfies the 8-byte alignment required for Mellanox atomics.

3.5 Benchmark and Automation

The project includes two benchmark executables. `kv_benchmark` targets the TCP server. `kv_client_rdma_benchmark` performs the required RDMA handshake and then runs the same style of workload against the RDMA modes. Both emit CSV files with elapsed time, throughput, mean latency, p50, p95, p99, and error counts. Runner scripts execute the TCP, two-sided RDMA, and one-sided RDMA sweeps on CloudLab and regenerate plots locally.

4 Experimental Methodology

4.1 Environment

Experiments were run on a two-node CloudLab setup with RDMA-capable Mellanox hardware. The server and client

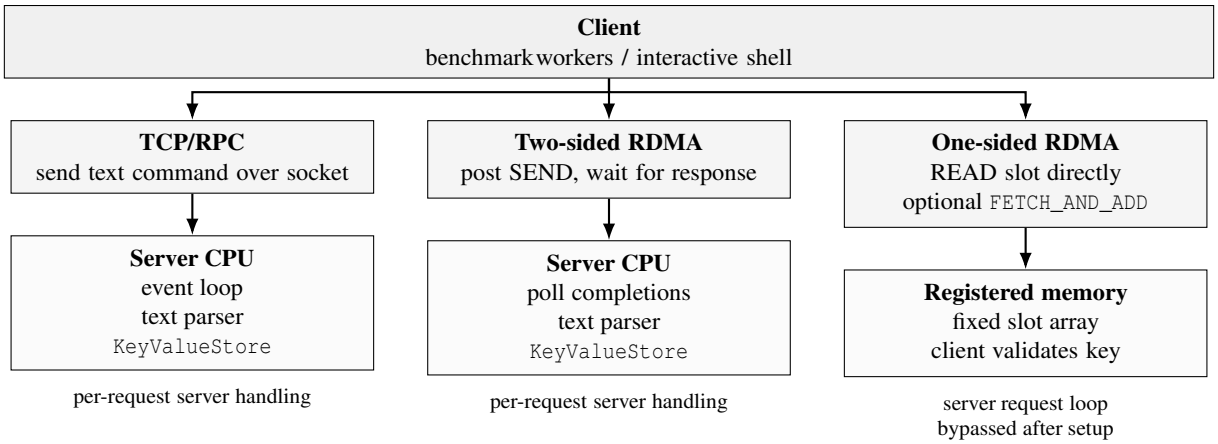

Figure 1: Data paths compared in RDMA Cache Prototype. TCP and two-sided RDMA both execute server-side request handling on every operation. One-sided RDMA reads the registered slot array directly, optionally issuing one atomic metadata update, and bypasses the server request loop after connection setup.

communicated over the private 10.10.x.x experiment network. A device-mapping check showed that the private experiment interface was `ens1f1np1`, backed by RDMA device `mlx5_3`; the public/control interface was `eno49np0`, backed by `mlx5_0`. All final RDMA measurements therefore used `-device mlx5_3` and sampled `ens1f1np1` for host network counters. The code was built with CMake and `libibverbs`; RDMA targets are compiled only when the verbs library is available.

Table 2: Final CloudLab experiment configuration.

Item	Value
Server data IP	`10.10.1.1`
Client data IP	`10.10.1.2`
Private netdev	`ens1f1np1`
RDMA device	`mlx5_3`
Control netdev	`eno49np0`
Control RDMA device	`mlx5_0`
Compiler	GNU C++ 11.4.0
Build system	CMake + `libibverbs`
Trials per point	3

4.2 Workloads

The final client-count sweep used 1, 2, 4, and 8 clients with 100,000 measured operations per point, 5,000 warmup operations, and 1024 prefilled keys. Each point was repeated three times. TCP and two-sided RDMA used 64-byte values, a 95% GET ratio, and Zipf skew $s = 0.8$.

These parameters were chosen to make the experiment large enough to produce stable measurements while keeping the workload focused on transport overhead. The client counts form a power-of-two scaling sweep from a single-client baseline to modest concurrency. Each point uses 100,000 measured operations, which reduced variance relative to earlier 10,000-operation runs, and 5,000 warmup operations to exclude connection setup, queue initialization, and cold cache effects from the measured interval. The 1024-key working set is intentionally small and preloaded which keeps the data resident and avoids conflating transport costs with capacity misses, paging, or large hash-table behavior. The 64-byte value size represents small cache objects, where fixed software and transport overheads are especially important.

Because the one-sided prototype supports direct reads over a preloaded RDMA slot array, its measurements are GET-only. The three-way comparison should therefore be read as a cache-read comparison rather than a complete replacement for mixed GET/SET service.

The metadata experiment compared one-sided reads with and without one `FETCH_AND_ADD` per operation. All final runs completed with zero warmup errors and zero measured errors.

4.3 Metrics

Throughput is measured as completed operations divided by timed elapsed time. Latency is measured on the client side and reported as mean, median, p95, and p99. Plots show the mean across three trials with error bars representing sample standard deviation. Server process CPU was sampled externally so the benchmark hot paths did not include instrumentation.

I also attempted server NIC byte-counter measurements after correcting the RDMA device mapping to `mlx5_3`/`ens1f1np1`. TCP counter values were plausible, but RDMA still reported near-zero bytes per operation.

5 Results

5.1 Transport Comparison

Figure 2 shows the main comparison across TCP, two-sided RDMA, and one-sided RDMA. TCP scales from 8.9k operations/s at one client to 72.9k operations/s at eight clients. Two-sided RDMA reaches 117k operations/s at one client and 404k operations/s at eight clients, a 5.5–13.2× improvement over TCP depending on concurrency. One-sided RDMA reads reach 161k operations/s at one client and 974k operations/s at eight clients.

The p99 latency results are more dramatic. TCP p99 latency is 132–164 μs across the sweep. Two-sided RDMA reduces p99 latency to 6–8 μs. One-sided RDMA reports a stable 12 μs p99 latency at every client count. These results support the first two proposal goals: RDMA messaging improves significantly on TCP, and one-sided reads remove most of the remaining server-side request overhead.

5.2 Metadata Overhead

Figure 3 isolates the cost of cache metadata updates. Adding one RDMA atomic operation after each one-sided read increases mean latency from roughly 4.8 μs to 7.8–8.0 μs. p99 latency rises from 12 μs to 15–16 μs.

The throughput results also show a metadata cost in the final trials. The metadata variant is 27–36% lower throughput than pure one-sided reads across the client-count sweep.

5.3 Server CPU Utilization

Figure 4 shows sampled server-process CPU utilization. TCP server CPU rises with client count, reaching 60.7% average process CPU at eight clients. Two-sided RDMA consumes more sampled server CPU because the server busy-polls completions and handles every request, reaching 479% average process CPU at eight clients. The one-sided server stays near idle for most samples because the server is not on the read data path after setup. However, it is important to note that many RDMA benchmark runs are short enough to produce only one or a few CPU samples. The qualitative direction is meaningful, but the exact percentages should not be overfit.

5.4 Network Counter Attempt

The final automation also recorded server network byte counters on `ens1f1np1`. These measurements are useful mostly as a validation lesson. TCP reported a plausible 233 total bytes per operation across all client counts. The RDMA modes, however, reported only 0.02–0.08 total bytes per operation, which is impossible for operations that exchange RDMA messages or read 1024-byte slots. I therefore exclude the generated network plot from the main evidence and treat Linux netdev counters as the wrong instrument for this RDMA/RoCE traffic on this setup.

6 Discussion

6.1 Progress Relative to the Proposal

The implementation satisfies the proposal's three project tiers. The 75% goal is met by the TCP key-value store, benchmark harness, and two-sided RDMA comparison. The 100% goal is met by the one-sided RDMA read path over a registered hash-table memory region. The 125% goal is substantially met by the RDMA atomic metadata experiment. The metadata implementation is not a full LRU list, but it captures the core issue from the proposal in that every read that updates cache policy must perform additional remote synchronization.

6.2 Interpretation

The results suggest a layered view of RDMA's benefit. Moving from TCP to two-sided RDMA removes a large amount of transport overhead but keeps the server CPU in the critical path. Moving from two-sided RDMA to one-sided RDMA removes server request processing from reads, producing very stable read latency. Adding metadata atomics then puts a small but consistent remote synchronization cost back into the path.

6.3 Limitations

The largest experimental limitation is that the one-sided benchmark is GET-only. This is appropriate for measuring one-sided reads, but it is not a complete mixed-operation cache. The interactive one-sided client supports RDMA writes, but the benchmark intentionally avoids concurrent one-sided SETs because a production-quality write protocol would need versioning, compare-and-swap, or a server-mediated slow path to prevent readers from observing partially updated slots.

The second limitation is measurement duration. The final runs use 100,000 operations per point and three trials, which is enough to stabilize the main latency and throughput trends. However, the fastest one-sided points still complete in a fraction of a second, which reduces the reliability of sampled CPU utilization. A stronger follow-up experiment would use longer 1M+ operation points for CPU and counter measurements.

Finally, network byte-counter measurements were attempted but are not used as evidence. Even after mapping the private experiment network to `mlx5_3`/`ens1f1np1`, the server's Linux netdev counters produced plausible TCP bytes per operation but near-zero RDMA bytes per operation. This likely means the counter path was not observing the RoCE/RDMA data path. Accurate network overhead

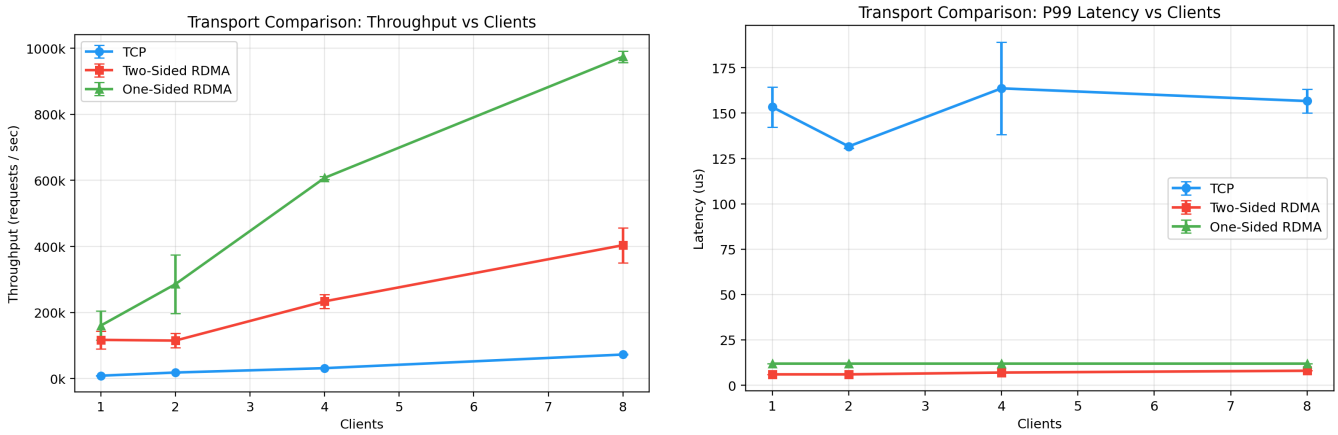

Figure 2: Transport comparison on CloudLab. Points are means across three trials and error bars show sample standard deviation. Two-sided RDMA substantially improves throughput over TCP while keeping server-side request processing. One-sided RDMA directly reads registered memory and provides the lowest read latency, but this sweep is GET-only and should be interpreted as the direct cache-read path.

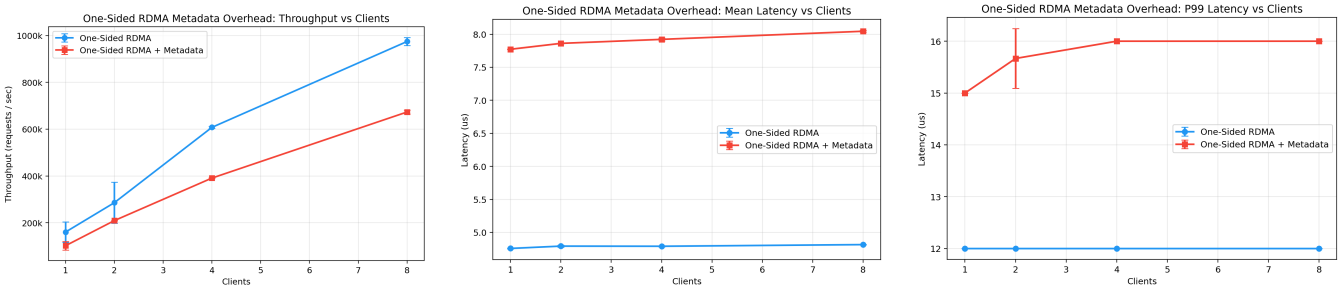

Figure 3: One-sided RDMA metadata overhead. The metadata variant performs an RDMA `FETCH_AND_ADD` after every RDMA read. Points are means across three trials and error bars show sample standard deviation. The atomic operation reduces throughput and adds a consistent latency penalty, which is the central cost predicted by the proposal.

analysis would need NIC hardware counters, `ethtool -S`, `perfquery`, or Mellanox-specific RDMA port counters.

7 Related Work

CliqueMap demonstrated that one-sided RDMA can be used to build production distributed caching systems, but also showed that the engineering details of metadata, consistency, and deployment matter greatly [5]. Recent work on disaggregated data structures studies the pitfalls of one-sided RDMA synchronization and emphasizes that avoiding remote CPU work does not eliminate the need for careful concurrency control [2]. Dinomo explores high-performance key-value storage over disaggregated persistent memory [3]. Cache policy work such as lazy promotion and hit-ratio/throughput studies motivates this project's focus on metadata: improving hit ratio or recency tracking can hurt throughput when the metadata path becomes too expensive [1, 4].

This project is much smaller than those systems, but it targets the same fundamental trade-off. By stripping the system down to TCP, two-sided RDMA, one-sided RDMA, and one metadata atomic, it isolates the cost of each communication mechanism in a way that is useful for a coarse-grained evaluation.

8 Conclusion

RDMA Cache Prototype demonstrates that the original proposal was feasible within the project scope. The prototype implements a TCP baseline, a two-sided RDMA server, and a one-sided RDMA read path over a registered hash table. On CloudLab, RDMA substantially improves throughput and tail latency relative to TCP. One-sided reads achieve low and stable latency without server CPU involvement, while an RDMA atomic metadata update adds a measurable 3–4 μs p99 latency cost.

The final answer to the research question is therefore nuanced. One-sided RDMA can significantly reduce distributed

Table 3: Primary client-count results, averaged across three trials. Throughput is in thousands of operations per second. All runs reported zero measured errors.

Mode	Workload	Clients	Throughput (kops/s)	Mean latency (μs)	p99 latency (μs)
TCP	95% GET, Zipf 0.8	1	8.9	106.4	153
		2	18.5	100.3	132
		4	31.7	110.9	164
		8	72.9	95.3	157
Two-sided RDMA	95% GET, Zipf 0.8	1	117.3	6.2	6
		2	115.1	11.2	6
		4	233.9	13.7	7
		8	403.8	15.2	8
One-sided RDMA	GET-only reads	1	160.5	4.8	12
		2	285.9	4.8	12
		4	607.6	4.8	12
		8	974.3	4.8	12

Table 4: Added latency from one RDMA metadata atomic per read.

Clients	Mean increase (μs)	p99 increase (μs)	Metadata p99 (μs)
1	3.0	3.0	15.0
2	3.1	3.7	15.7
4	3.1	4.0	16.0
8	3.2	4.0	16.0

cache read overhead, but cache metadata is not free. Even a simple atomic recency counter consumes part of the latency budget. A full production cache would need to decide which metadata updates truly belong on the read critical path and which can be batched, approximated, or moved off path.

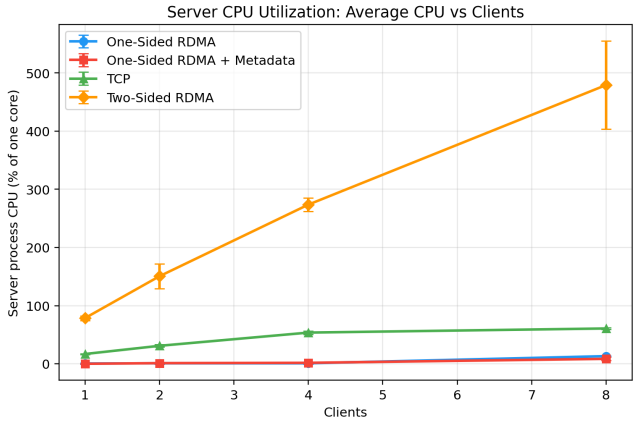

Figure 4: Sampled average server-process CPU. The one-sided server is mostly idle during the read data path, while TCP and two-sided RDMA require server CPU work for each request. Short RDMA runs make this metric noisy.

Availability

The submitted artifact contains the C++ implementation, CloudLab runner scripts, raw CSV files, aggregated CSV files, and generated plots. The primary final inputs are the raw trials under `experiments/final_trials_mlx5_3_100k/`, the aggregated CSVs under `experiments/final_summary_mlx5_3_100k/`, and the report figures under `report/plots/final_summary_mlx5_3_100k/`.

References

[1] Qinghan Chen, Muhammad Haekal Muhyidin, Ziyue Qiu, Zhuofan Chen, Rashmi Vinayak, and Juncheng Yang. Demystifying and Improving Lazy Promotion in Cache Eviction.

[2] Matthias Jasny, Tobias Ziegler, Jacob Nelson-Slivon, Viktor Leis, and Carsten Binnig. Synchronizing disaggregated data structures with one-sided RDMA: pitfalls, experiments and design guidelines. *ACM Transactions on Database Systems*, 50(1), 2025.

[3] Sekwon Lee, Soujanya Ponnapalli, Sharad Singhal, Marcos K. Aguilera, Kimberly Keeton, and Vijay Chidambaram. Dinomo: an elastic, scalable, high-performance key-value store for disaggregated persistent memory. arXiv:2209.08743, 2022.

[4] Ziyue Qiu, Juncheng Yang, and Mor Harchol-Balter. Can Increasing the Hit Ratio Hurt Cache Throughput? In *EAI International Conference on Performance Evaluation Methodologies and Tools*, 2024.

[5] Arjun Singhvi, Aditya Akella, Maggie Anderson, Rob Cauble, Harshad Deshmukh, Dan Gibson, Milo M. K. Martin, Amanda Strominger, Thomas F. Wenisch, and Amin Vahdat. CliqueMap: productionizing an RMA-based distributed caching system. In *Proceedings of ACM SIGCOMM*, 2021.