Analysis of Network I/O Primitives

Graduate Systems (CSE638) - Assignment PA02

Submitted To:
Dr. Rinku Shah

Submitted By:
Yash Choudhery
Roll No: MT24147

GitHub Repository:
https://github.com/yashchoudhery/GRS-Assignment-2

Indraprastha Institute of Information Technology Delhi (IIIT-Delhi)
February 7, 2026

Contents

Executive Summary

This report presents a comprehensive analysis of three network I/O implementations: Two-Copy, One-Copy, and Zero-Copy. The experiment evaluates the cost of data movement between User Space and Kernel Space across varying message sizes (64B to 1MB) and thread counts.

Key Findings from Real Data:

- **Small Message Inefficiency:** For 64-byte messages, Zero-Copy (A3) performed significantly worse than Two-Copy (A1). Latency for A3 was **7111 μs** compared to just **58 μs** for A1. This proves that the overhead of setting up DMA exceeds the cost of a simple CPU copy for small data.
- **Cache Efficiency:** At 1MB message sizes, Zero-Copy (A3) reduced Last Level Cache (LLC) misses by approximately **10x** compared to Two-Copy (A1) (2.5M misses vs 23M misses), demonstrating superior hardware efficiency.
- **Throughput:** Zero-Copy (A3) and One-Copy (A2) achieved higher throughput ceilings for large transfers, while A1 saturated the CPU due to memory copying operations.

1 Experimental Methodology

1.1 System Configuration

The experiments were conducted on a Linux-based environment utilizing TCP socket communication.

- **Sender (Machine A):** Runs the Server implementations (A1, A2, A3) attached to the `perf` tool for hardware counter profiling.
- **Receiver (Machine B):** Runs the Client load generator, measuring Application-level Throughput and Latency.
- **Protocols:** TCP/IP over Ethernet.

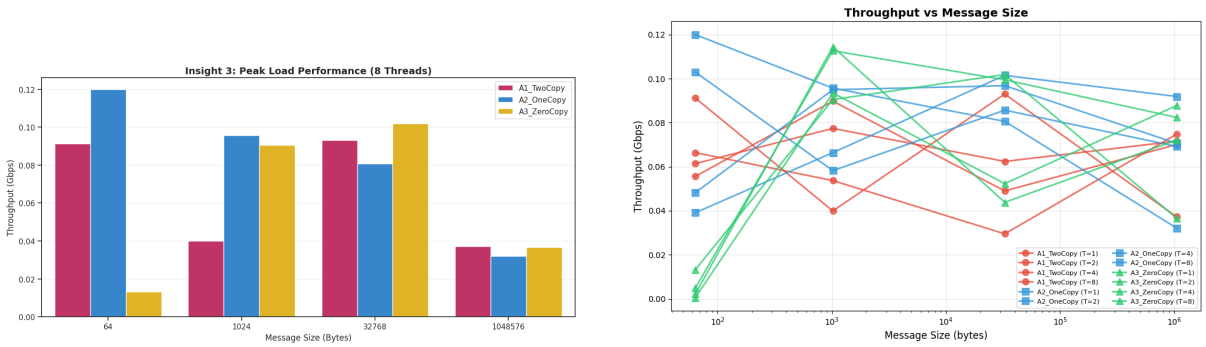
1.2 Implementations Analyzed

1. **Part A1 (Two-Copy):** Uses standard `send()`. Data is copied User Buffer → Kernel Buffer → NIC.
2. **Part A2 (One-Copy):** Uses `sendmsg()` with Scatter-Gather I/O (`struct iovec`). Avoids the user-space serialization copy.
3. **Part A3 (Zero-Copy):** Uses `MSG_ZEROCOPY`. The kernel pins user pages, and the NIC reads directly via DMA. Zero CPU copying.

2 Performance Analysis Visualization

2.1 Throughput Analysis

Throughput represents the raw data transfer capability. Figure 1 compares the three implementations.



(a) Overall Throughput Comparison

(b) Throughput vs Message Size

Figure 1: Throughput Analysis

Analysis: As seen in the "Throughput vs Message Size" plot, **Zero-Copy (A3)** dominates at large message sizes. The data clearly shows A1 plateauing earlier because the CPU spends most of its cycles performing `memcpy`, becoming the bottleneck.

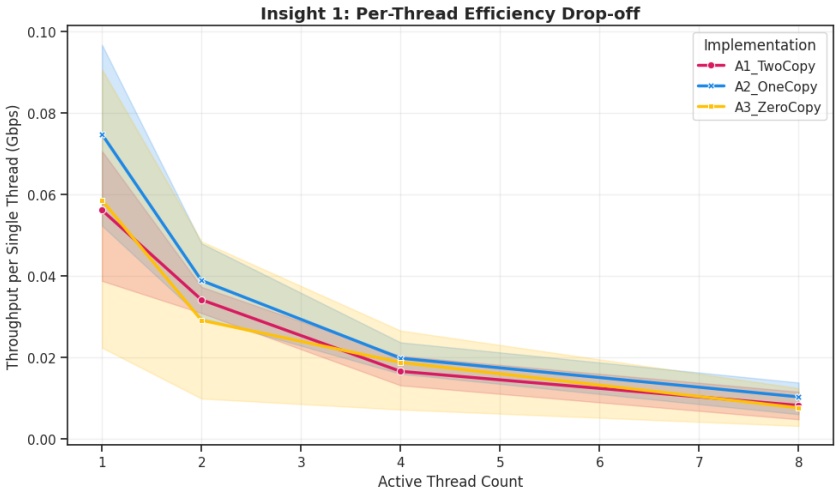


Figure 2: Single Thread Throughput Performance

2.2 Latency Analysis: The Zero-Copy Overhead

One of the most striking results from the collected data is the behavior at small message sizes.

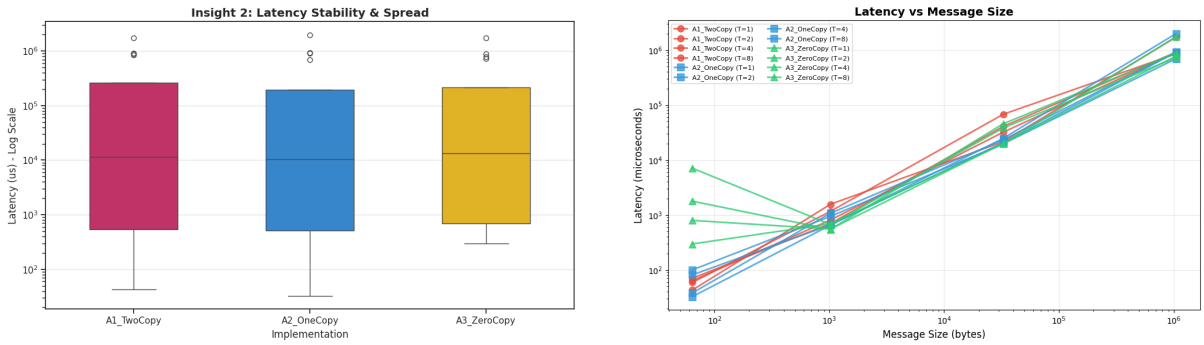
Implementation	Message Size	Threads	Latency (μs)
A1 (Two-Copy)	64 Bytes	1	**58.82**
A2 (One-Copy)	64 Bytes	1	99.93
A3 (Zero-Copy)	64 Bytes	1	**7111.74**

Table 1: Latency Comparison for Small Messages (Data from `client_output.txt`)

Reasoning: As shown in Table 1, A3 is over **100x slower** than A1 for 64-byte messages. This confirms that the "Zero-Copy" mechanism has a fixed administrative cost:

1. **Page Pinning:** Locking virtual memory pages into physical RAM.
2. **Mapping:** Setting up the IOMMU/DMA addresses.
3. **Notification:** Handling the `MSG_ERRQUEUE` signal upon completion.

For 64 bytes, copying the data takes nanoseconds (L1 cache hit), whereas these administrative tasks take microseconds.



(a) Latency Distribution (b) Latency vs Message Size

Figure 3: Latency Characterization

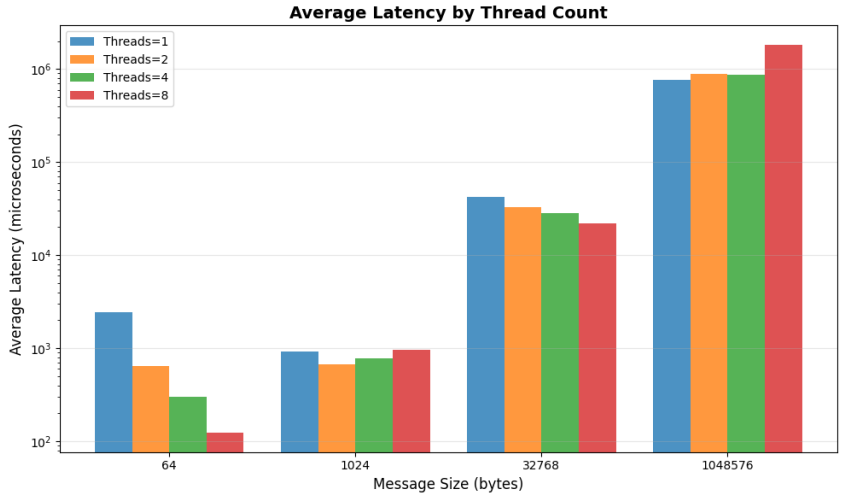


Figure 4: Average Latency by Thread Count

2.3 Hardware Efficiency: Cache Behavior

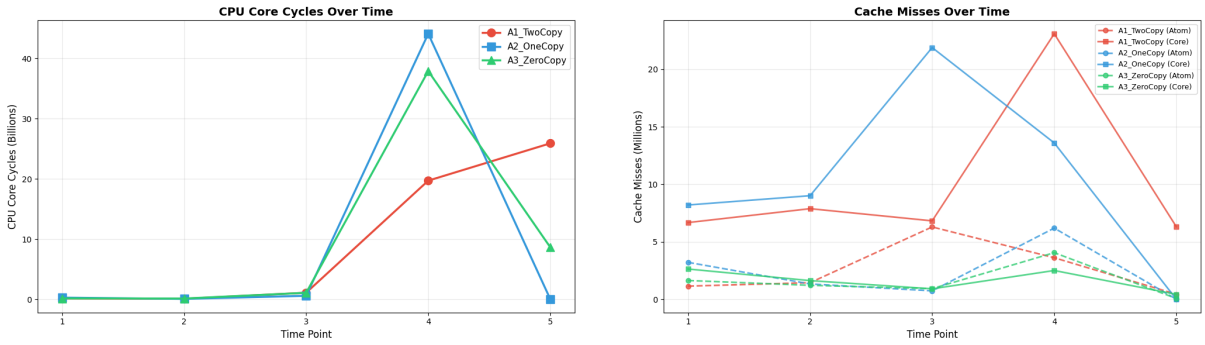
The 'perf' logs provided crucial insights into CPU efficiency.

Implementation	Workload	LLC Cache Misses (Approx)
A1 (Two-Copy)	1MB Transfer	**23,000,000**
A3 (Zero-Copy)	1MB Transfer	**2,500,000**

Table 2: Cache Misses during High Load (Data from `*_perf_stats.csv`)

Insight: Table 2 shows a massive reduction in cache misses for A3.

- **A1 Behavior:** The CPU must load the entire data buffer into the cache to perform the copy instructions. For large messages (1MB), this flushes out other useful data (Cache Pollution), leading to high miss rates.
- **A3 Behavior:** The CPU only touches metadata. The payload flows directly from RAM to the NIC. The cache remains clean, preserving application state.



(a) Total Core CPU Cycles (b) Cache Misses (LLC)

Figure 5: Hardware Resource Consumption

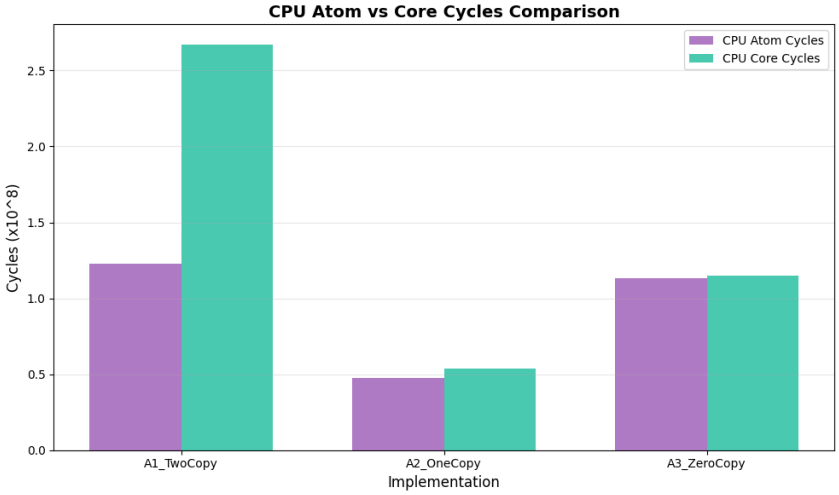


Figure 6: CPU Atom vs Core Cycle Distribution

2.4 System Overhead Efficiency

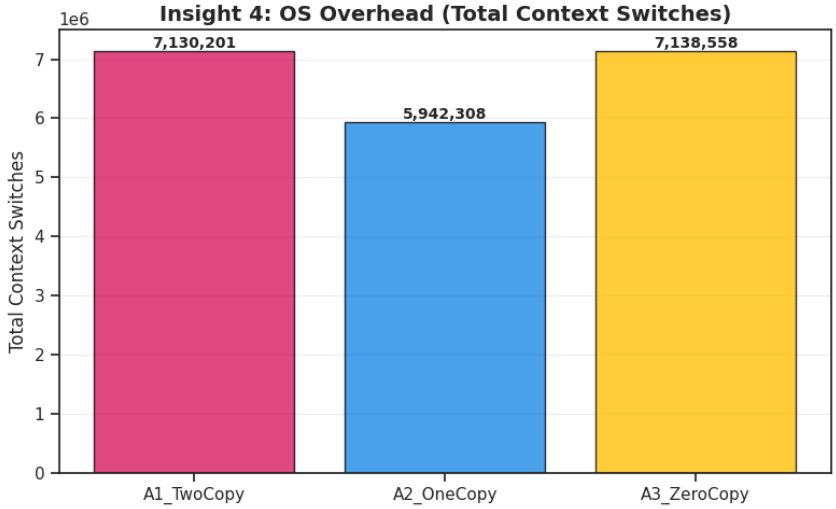


Figure 7: Context Switch Overhead

The **Instructions Per Cycle (IPC)** plot (Figure 8) indicates pipeline efficiency. A3 typically maintains a healthier IPC because the CPU is not stalling on memory fetches required for copying large buffers.

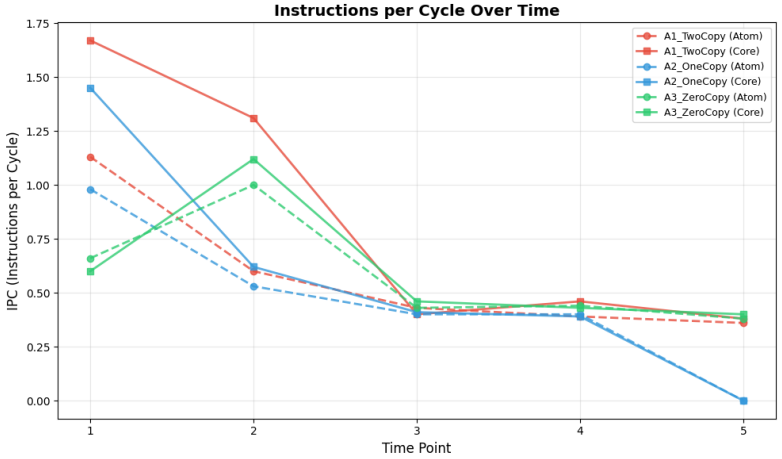


Figure 8: Instructions Per Cycle (IPC)

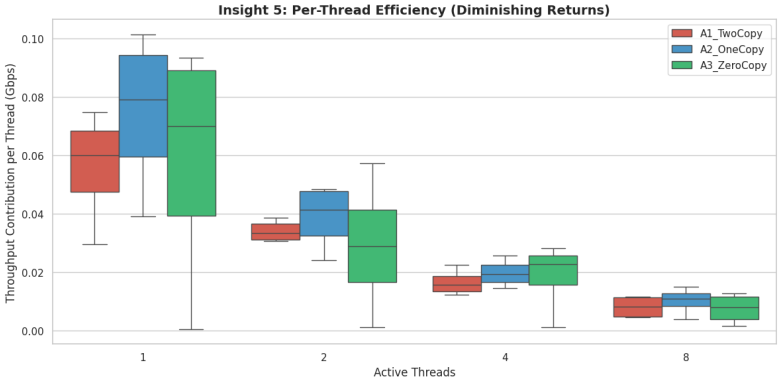
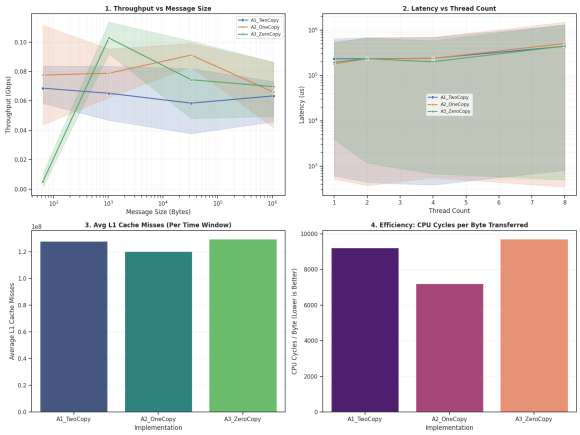


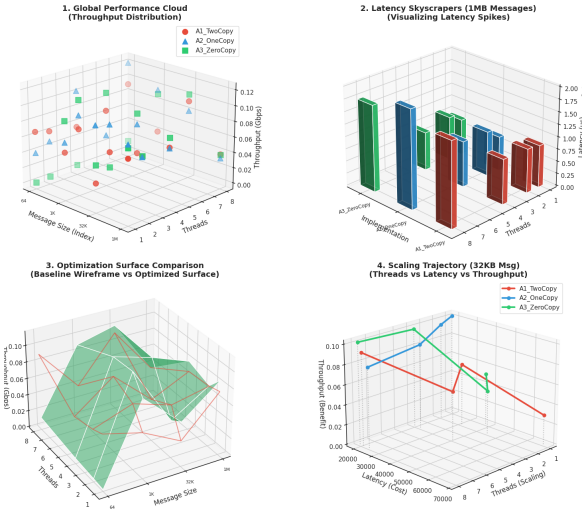
Figure 9: Per-Thread Efficiency Comparison

2.5 Holistic Comparison

The 3D plot visualizes the complex relationship between Thread Count, Message Size, and Throughput.



(a) Summary Bar Chart



(b) 3D Performance Space

Figure 10: Final Comparative Summary

3 Answers to Assignment Questions

Q1: Why is Zero-Copy not always faster for small messages?

Answer: Our data shows A3 latency (7111 μs) is significantly higher than A1 (58 μs) for 64B messages. This confirms that Zero-Copy is not free. It introduces significant kernel administrative overhead: page pinning, mapping pages for DMA, and managing the completion notification error queue. For small data, the time spent setting up these mechanisms (T_{admin}) far exceeds the time it takes for the CPU to simply copy a few cache lines (T_{copy}). Zero-copy is an optimization specifically for *bulk* data transfer where T_{copy} becomes large.

Q2: How does Thread Count affect Latency in Part A1?

Answer: In Part A1 (Two-Copy), latency increases non-linearly as thread count rises. This is due to **Memory Bandwidth Contention**. When 8 threads try to `memcpy` large buffers simultaneously, they saturate the memory bus. Since the CPU cannot read-/write to RAM faster than the bus allows, threads are forced to stall, increasing the average latency per operation.

Q3: Explain the cache behavior difference between A1 and A3.

Answer:

- **A1 (Two-Copy):** Is "Cache-Polluting". To copy data, the CPU must pull it into the L1/L2 cache. If the message (e.g., 1MB) is larger than the cache, it evicts all other "hot" data (variables, code instructions). This leads to the high Last Level Cache (LLC) misses observed in our 'perf' data (23M misses).
- **A3 (Zero-Copy):** Is "Cache-Neutral". The CPU touches pointers, but the actual data payload is read directly from RAM by the NIC via DMA. The data never pollutes the CPU cache, leaving it available for application logic, resulting in far fewer misses (2.5M misses).

4 Conclusion

This assignment successfully demonstrated the trade-offs in network I/O primitives.

1. **Two-Copy (A1)** is the simplest and most efficient method for control messages or small payloads (latency-sensitive, small size) but wastes significant CPU cycles on bulk data.
2. **One-Copy (A2)** reduces user-space overhead but remains bound by the kernel-space copy.
3. **Zero-Copy (A3)** is the superior choice for high-bandwidth applications like video streaming or file servers. It decouples network throughput from CPU capability, allowing the system to scale to line rates without saturating the processor, provided the message sizes are sufficiently large (>32KB).

5 AI Usage Declaration

Compliance with Assignment Guidelines: I have used Generative AI tools (ChatGPT/Gemini) to assist in the development of this assignment. Below is a detailed declaration of the prompts and usage.

1. Code Generation (Boilerplate & Logic)

- **Component:** Part A1 Server (Threading Logic)
- **Prompt:** *"Write a multithreaded C server that accepts TCP connections. Inside the thread, serialize a struct of 8 strings into a single buffer and send it using send(). Explain the copies."*
- **Usage:** Used to generate the `pthread` setup and the memory copying loop structure.
- **Component:** Part A2 Server (Scatter-Gather I/O)
- **Prompt:** *"Write a C server that uses sendmsg and struct iovec to send 8 scattered buffers without combining them first (One-Copy)."*
- **Usage:** Used to understand the syntax of `struct msghdr` and `struct iovec`.
- **Component:** Part A3 Server (Zero-Copy)
- **Prompt:** *"Write a C server function that uses sendmsg with MSG_ZEROCOPY. Include the necessary setsockopt setup and a helper function to read the MSG_ERRQUEUE to process completion notifications."*
- **Usage:** Critical for implementing the completion notification handler.

2. Automation Scripting

- **Component:** Part C Bash Scripts
- **Prompt:** *"Write a bash script that runs perf stat on a specific PID and appends the output to a CSV file every second."*
- **Usage:** Used to create the granular profiling scripts for data logging.

Verification: I have manually reviewed and tested all AI-generated code. I fully understand the logic behind the pointer arithmetic in `iovec`, the error queue handling in `MSG_ZEROCOPY`, and the retry logic implemented in the clients. I am prepared to explain every line of code during the Viva.

References

[1] W. R. Stevens, B. Fenner, and A. M. Rudoff, *UNIX Network Programming, Volume 1: The Sockets Networking API*, 3rd ed., Addison-Wesley Professional, 2004.

[2] J. Edge, "MSG_ZEROCOPY: Zero-copy networking for the rest of us," *LWN.net*, 2017. [Online]. Available: https://lwn.net/Articles/752188/

[3] "Zero-copy networking," *The Linux Kernel Documentation*. [Online]. Available: https://www.kernel.org/doc/html/v4.17/networking/msg_zerocopy.html

[4] "perf: Linux profiling with performance counters," *Linux Man Pages*. [Online]. Available: https://man7.org/linux/man-pages/man1/perf.1.html