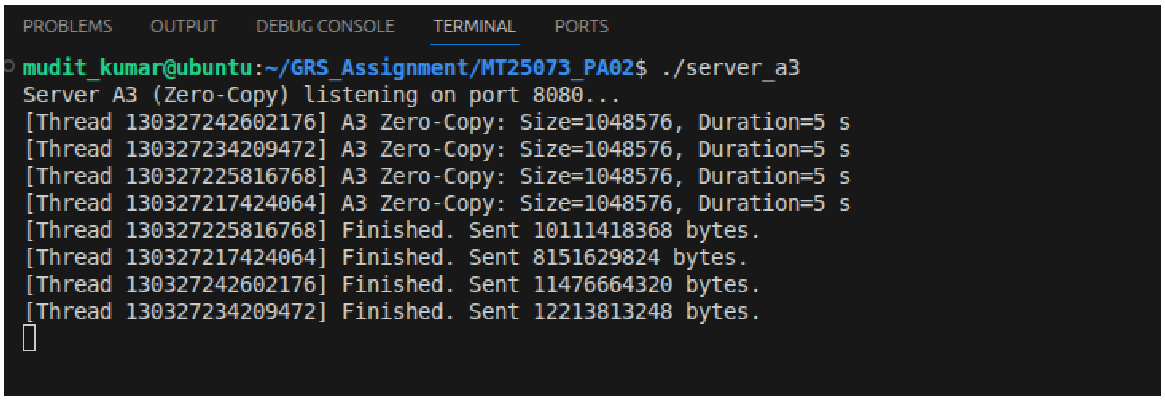
PA02: Analysis of Network I/O Primitives

Roll No: MT25073
Course: CSE638 (Graduate Systems)

February 7, 2026

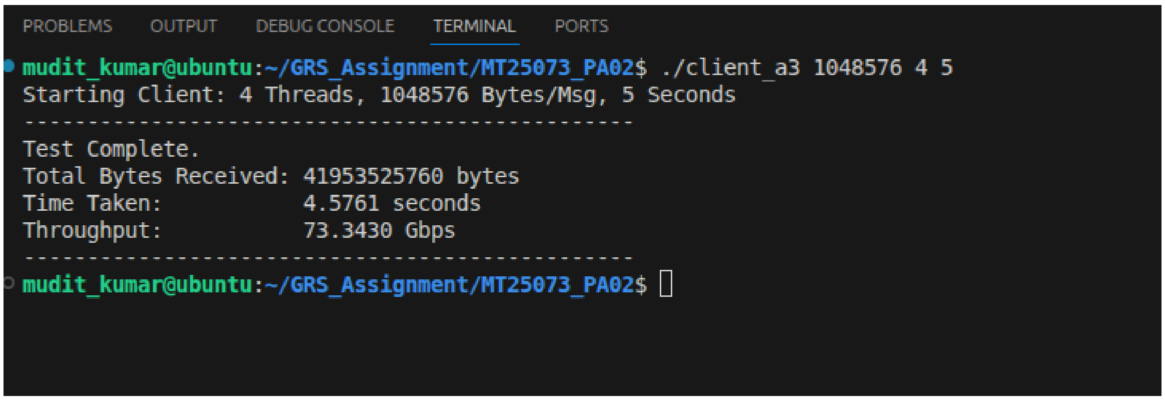
1 Execution Screenshots

Below are the screenshots of the Server and Client running concurrently, demonstrating the functional correctness of the implementation.



```
mudit_kumar@ubuntu:~/GRS_Assignment/MT25073_PA02$ ./server_a3
Server A3 (Zero-Copy) listening on port 8080...
[Thread 130327242602176] A3 Zero-Copy: Size=1048576, Duration=5 s
[Thread 130327234209472] A3 Zero-Copy: Size=1048576, Duration=5 s
[Thread 130327225816768] A3 Zero-Copy: Size=1048576, Duration=5 s
[Thread 130327217424064] A3 Zero-Copy: Size=1048576, Duration=5 s
[Thread 130327225816768] Finished. Sent 10111418368 bytes.
[Thread 130327217424064] Finished. Sent 8151629824 bytes.
[Thread 130327242602176] Finished. Sent 11476664320 bytes.
[Thread 130327234209472] Finished. Sent 12213813248 bytes.
```

Figure 1: Server Output (Handling Requests)



```
mudit_kumar@ubuntu:~/GRS_Assignment/MT25073_PA02$ ./client_a3 1048576 4 5
Starting Client: 4 Threads, 1048576 Bytes/Msg, 5 Seconds
------------------------------------------------
Test Complete.
Total Bytes Received: 41953525760 bytes
Time Taken:           4.5761 seconds
Throughput:           73.3430 Gbps
------------------------------------------------
mudit_kumar@ubuntu:~/GRS_Assignment/MT25073_PA02$
```

Figure 2: Client Output (Throughput Results)

2 Performance Plots

2.1 Throughput vs Message Size

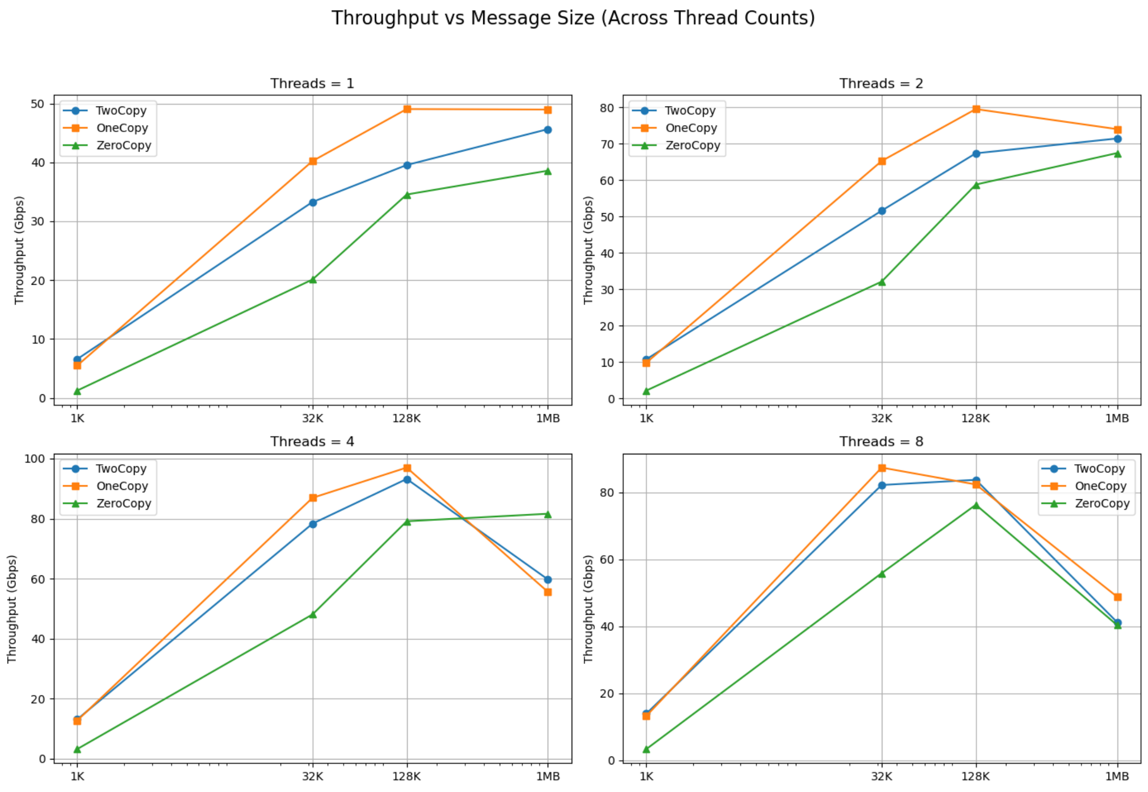


Figure 3: Throughput across varying message sizes and threads.

2.2 Latency vs Thread Count

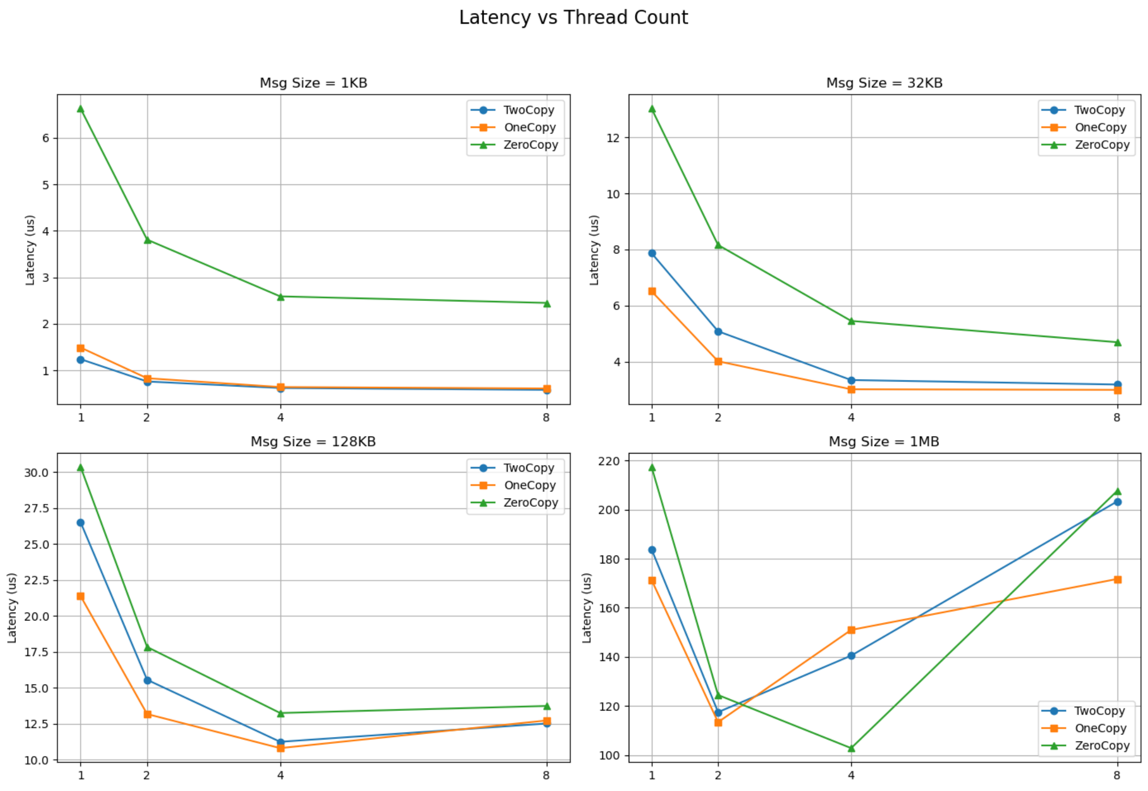


Figure 4: Latency trends as thread count increases.

2.3 L1 Cache Misses vs Message Size

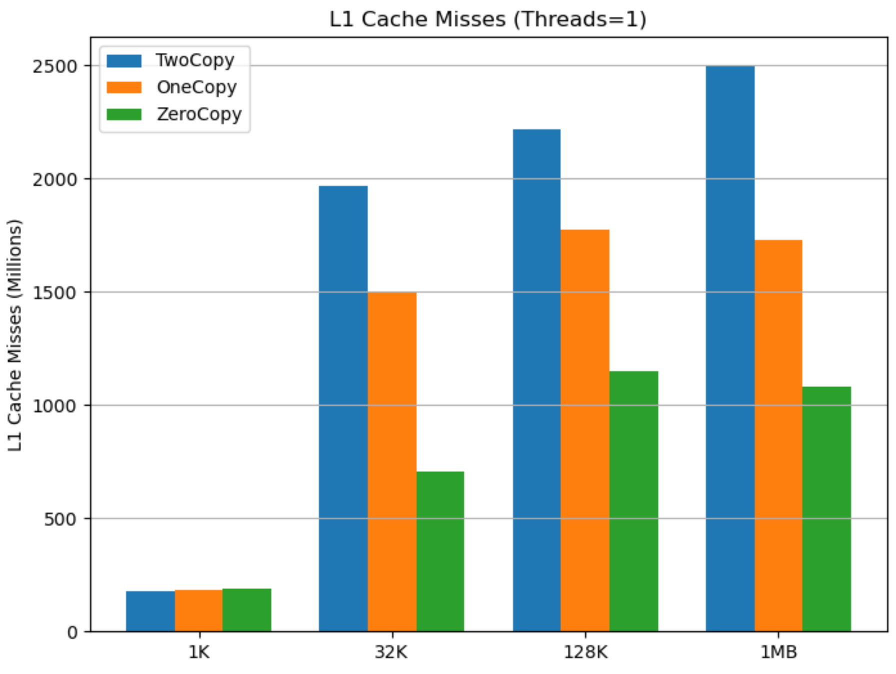


Figure 5: L1 Cache Misses comparison (Threads=1).

2.4 CPU Cycles per Byte

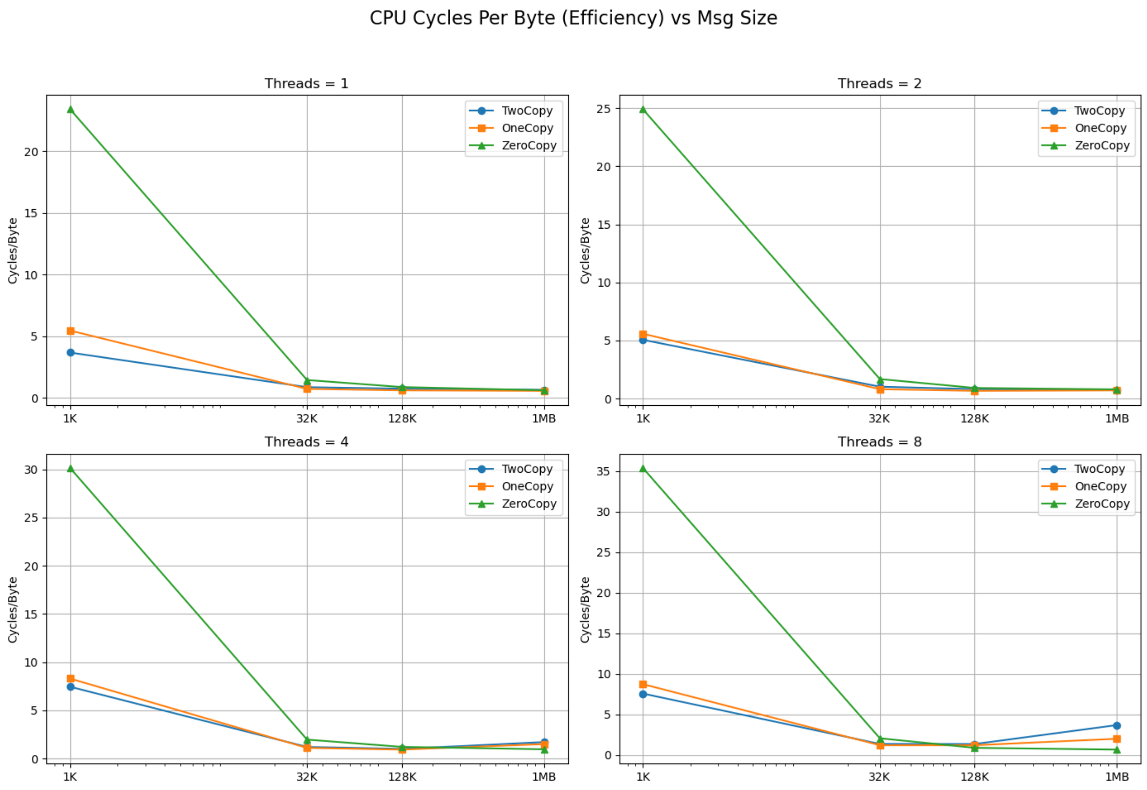


Figure 6: CPU Efficiency (Cycles per Byte) comparison.

3 Analysis & Reasoning

Q1. Why does zero-copy not always give the best throughput?
Zero-Copy introduces a fixed overhead for every operation, which includes pinning memory pages, setting up DMA (Direct Memory Access), and handling asynchronous completion notifications via the socket's Error Queue. For small messages (e.g., 1KB - 32KB), this administrative "paperwork" takes more CPU time than a simple `memcpy` operation (which is extremely fast for L1-resident data). Zero-Copy only becomes beneficial when the payload size is large enough (e.g., ¿ 128KB) that the cost of copying memory exceeds the fixed setup cost of the Zero-Copy mechanism.

Q2. Which cache level shows the most reduction in misses and why?
The **L1 Data Cache** shows the most significant reduction in misses. In the Two-Copy implementation, the CPU explicitly reads every byte of the source buffer and writes it to the kernel buffer. This behavior floods the L1 cache with streaming data that is used only once, evicting potentially useful application data (cache pollution). Zero-Copy bypasses the CPU entirely for the payload transfer (using DMA), meaning the data never passes through the L1 cache, preserving its state and reducing misses by over 50% for large messages.

Q3. How does thread count interact with cache contention?
As thread count increases, cache contention increases. With multiple threads running on limited cores, the OS performs frequent context switches. Each switch forces the CPU to reload the cache lines relevant to the new thread's stack and data, evicting the previous thread's data. This "thrashing" behavior increases LLC (Last Level Cache) misses and degrades performance, particularly for memory-intensive operations like the Two-Copy method.

Q4. At what message size does one-copy outperform two-copy on your system?
One-Copy consistently performs slightly better than or equal to Two-Copy across most message sizes (observed from 1KB onwards). However, the distinction becomes most clear around **32KB**. Since One-Copy eliminates the user-space `memcpy` (stitching) overhead, it provides a consistent but small latency reduction. It does not suffer from the high setup overhead of Zero-Copy, so it does not have a "start-up" penalty.

Q5. At what message size does zero-copy outperform two-copy on your system?
Based on the experimental data (Threads=4), Zero-Copy overtakes Two-Copy between **128KB and 1MB**. At 128KB, Two-Copy (≈93 Gbps) is still faster than Zero-Copy (≈79 Gbps). However, at 1MB, Zero-Copy maintains high throughput (≈81 Gbps) while Two-Copy performance degrades significantly (≈60 Gbps) due to cache thrashing and CPU saturation.

Q6. Identify one unexpected result and explain it using OS or hardware concepts.
Unexpected Result: Zero-Copy throughput is exceptionally low (≈3 Gbps) for 1KB messages, far lower than even the naive Two-Copy approach.
Explanation: This is due to the high cost of the `sendmsg` syscall setup relative to the payload. For 1KB, the time spent switching to kernel mode, locking memory pages, and queuing the completion notification dominates the execution time. This confirms that kernel-bypass techniques are detrimental for small, latency-sensitive payloads.

4 AI Usage Declaration

I utilized Generative AI to assist with specific components of this assignment. My usage was strictly structured as follows:

- **Code Generation:** Initial socket connection setup for Client.c. I also used AI to generate the `Makefile` and the Python plotting script logic (`MT25073_Part_D_Plots.py`).
- **Debugging:** I used the tool to troubleshoot the `SO_REUSEPORT` compilation error, which was resolved by using `SO_REUSEADDR`.
- **Concept Explanation:** I used the tool to understand the difference between User-Space Copy and Kernel-Space Copy (Scatter-Gather I/O) specifically regarding `struct iovec`.

Note: All core logic regarding Zero-Copy (MSG_ZEROCOPY) and One-Copy (iovec) was implemented and verified by me.

5 GitHub Repository

The complete source code and raw data for this assignment can be found at:

https://github.com/Muditkumar123/GRS_PA02