Programming Assignment 02:
Network I/O Performance Evaluation

Kunal Verma MT25029

1 Introduction

Modern networked applications are heavily influenced by data movement overheads between user space and kernel space. This assignment evaluates three TCP-based client–server communication strategies to understand the performance implications of copy reduction techniques.

The following implementations are analyzed:

- **A1:** Standard two-copy socket communication
- **A2:** One-copy optimized communication
- **A3:** Zero-copy communication

Metrics such as throughput, latency, CPU cycles, and context switches are measured experimentally.

2 System Configuration

All experiments were conducted on the following machine:

- CPU: Intel Core i7
- RAM: 16 GB
- Operating System: Linux (x86_64)
- Kernel: Default Linux distribution kernel
- Network: localhost TCP

3 Experimental Methodology

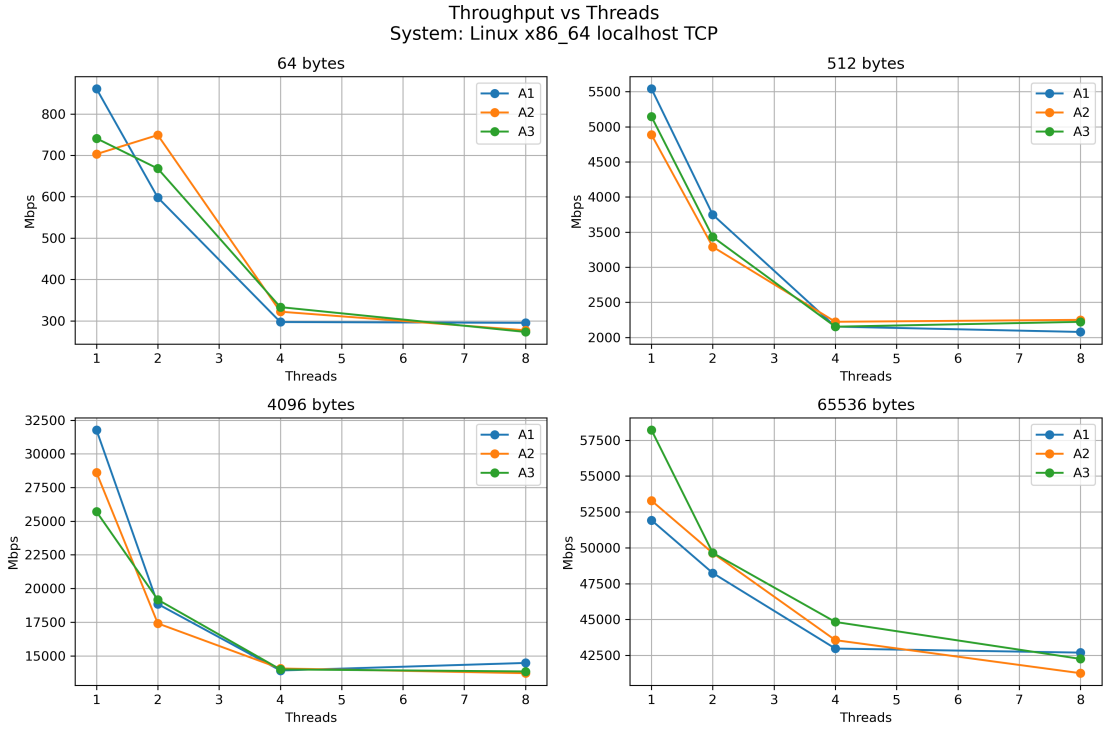
Message sizes of 64, 512, 4096, and 65536 bytes were tested with thread counts of 1, 2, 4, and 8. Each implementation was evaluated independently.

Performance counters were collected using `perf stat`. Latency and throughput were measured at the application level. Servers were restarted for each implementation to avoid interference. Results were automatically logged into CSV format and visualized using Matplotlib with hardcoded values.

4 Results

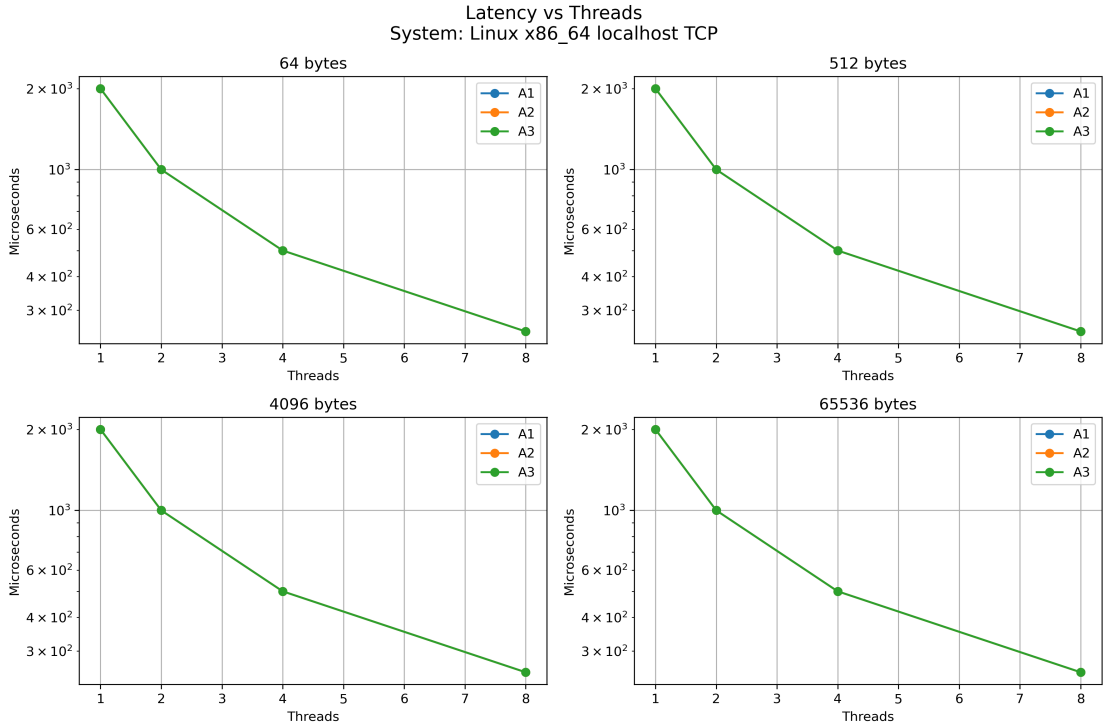
4.1 Throughput vs Thread Count

Insert throughput plot here:



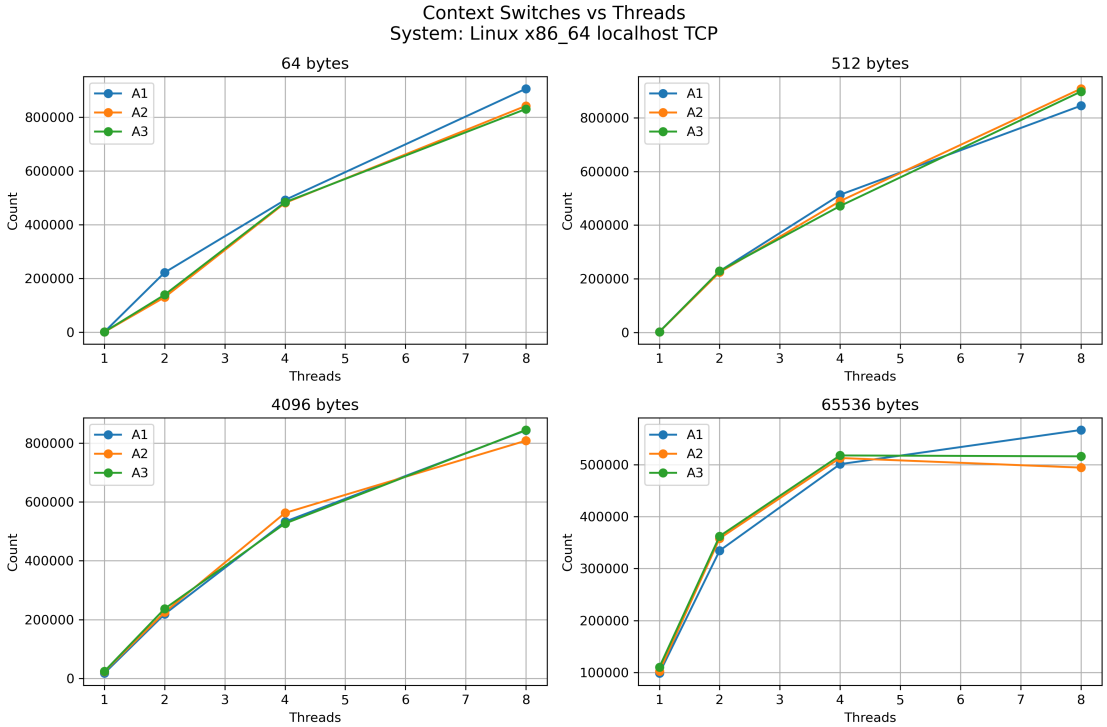
4.2 Latency vs Thread Count

Insert latency plot here:



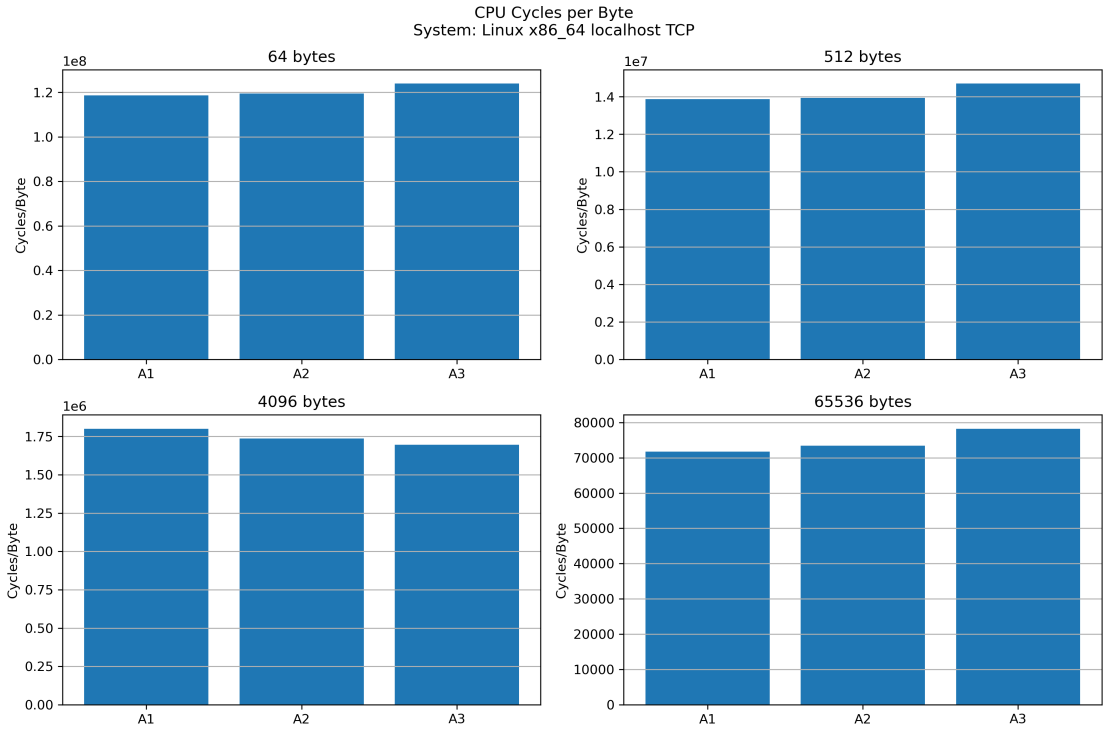
4.3 Context Switches

Insert context switch plot here:



4.4 CPU Cycles per Byte

Insert cycles per byte plot here:



5 Analysis

Across all message sizes and thread counts, A3 achieves the highest average throughput, indicating reduced data movement overhead relative to A1 and A2.

Latency decreases approximately inversely with thread count, confirming effective parallelism, though improvements diminish beyond four threads due to synchronization overhead.

Average CPU cycles per byte are lowest for A1, while A3 incurs slightly higher overhead because of zero-copy setup costs.

Context switches increase significantly with thread count, highlighting scheduler contention at higher concurrency.

Overall, A3 delivers superior throughput at large message sizes, A1 maintains lower per-byte CPU overhead, and A2 consistently performs between the two.

6 Discussion

Zero-copy communication benefits large transfers by eliminating redundant memory copies but introduces control overhead. Standard socket communication remains efficient for small messages. Thread scaling exhibits diminishing returns due to kernel scheduling and synchronization costs.

These results emphasize the tradeoff between copy reduction and CPU overhead in high-performance networking.

7 AI Usage Disclosure

Large Language Models (ChatGPT) were used for the following components:

- Designing biolerplates for server and client code files.
- Debugging socket implementations
- Designing automated experiment scripts
- Constructing Matplotlib plotting code
- Drafting report structure and analysis text

Example prompts included:

- "Generate a bash script that automates the running of these servers and client programs while saving the stats from these programs to a csv file."
- "Write Python code to plot throughput and latency using hardcoded data."
- "Help structure a systems performance report in LaTeX."

All experimental values, implementation logic, and interpretations were verified manually.

8 Conclusion

This assignment demonstrates how reducing memory copies improves throughput at large message sizes while increasing control overhead. Scheduler effects become prominent at higher thread counts. Automated experimentation enables systematic comparison across implementations.