Abstract

Modern networked systems are increasingly limited by data movement costs rather than raw computation. Each copy of data between user space and kernel space consumes CPU cycles, pollutes caches, and increases memory bandwidth pressure. This experiment studies the impact of different socket communication techniques on performance by implementing three client–server models in C: a standard multi-copy approach (A1), a one-copy optimized approach using `sendmsg` and scatter–gather I/O (A2), and a client benchmark used to evaluate throughput (A3).

Multithreaded servers were developed for each method, and clients were used to measure total bytes received over time. The experiments demonstrate that reducing data copies improves throughput and lowers CPU overhead. The one-copy implementation achieves better performance compared to the traditional multiple `send()` approach due to fewer system calls and reduced memory copying.

Introduction

Network I/O performance is a critical factor in distributed systems, cloud computing, and high-performance servers. Traditional socket communication involves multiple data copies: application buffers are copied into kernel buffers and then into network device buffers. These copies increase latency and consume CPU resources.

The objective of this assignment is to experimentally study:

1. Standard two-copy socket communication
2. One-copy optimized socket communication
3. Zero-copy socket communication

By implementing these techniques and comparing their behavior, we analyze how data movement affects performance.

Socket Communication Implementations

A1. Two-Copy Implementation

The baseline implementation uses standard socket primitives `send()` and `recv()` to transmit data between a multithreaded server and client.

Where Do the Two Copies Occur?

Copy 1: User Space → Kernel Space

The application provides a user buffer. The kernel copies this data into its internal socket send buffer.

Copy 2: Kernel Space → NIC

The kernel networking stack copies data from its socket buffer to the network interface card (NIC) for transmission.

Who Performs These Copies?

In User to Kernel this is performed by Kernel
In Kernel to NIC this is performed by NIC

A2. One-Copy Implementation
The one-copy version replaces multiple `send()` calls with a single `sendmsg()` call using scatter–gather I/O.

User buffer (pinned)

  |

  | NO copy_from_user()

  ▼

NIC

In this directly user to Nic is done by eleminating Kernel copy

A3. Zero-Copy Implementation

This allows the kernel to transmit application buffers directly without copying them into kernel memory.

Diagram

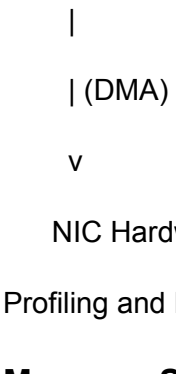
User Memory Buffers

      |

      | (mapped, not copied)

      v

  Kernel Socket Layer



Profiling and Measurement

Message Sizes

- 1024 bytes
- 4096 bytes
- 16384 bytes
- 65536 bytes

Thread Counts

- 1
- 2
- 4
- 8

A1 (Two-Copy)

- Highest CPU cycles
- Most cache misses
- Lowest throughput
- Many context switches

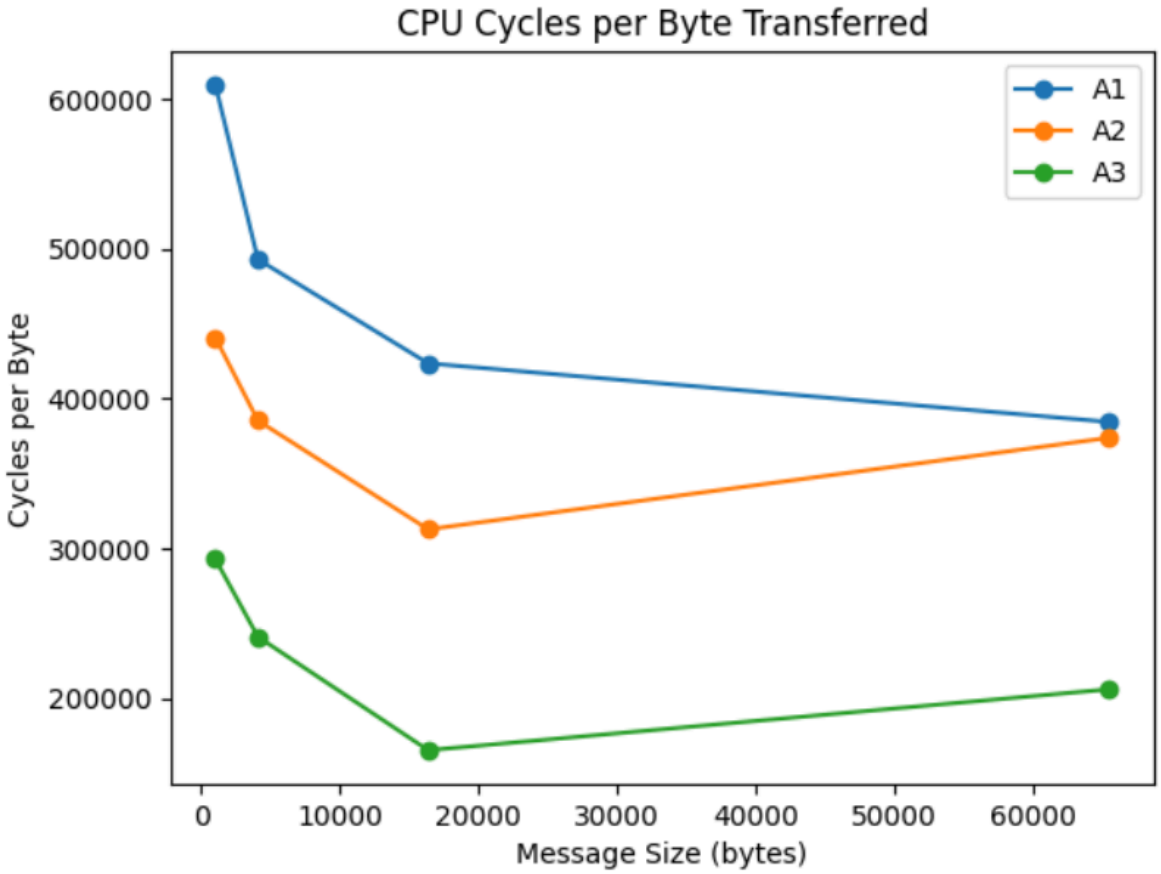
A2 (One-Copy)

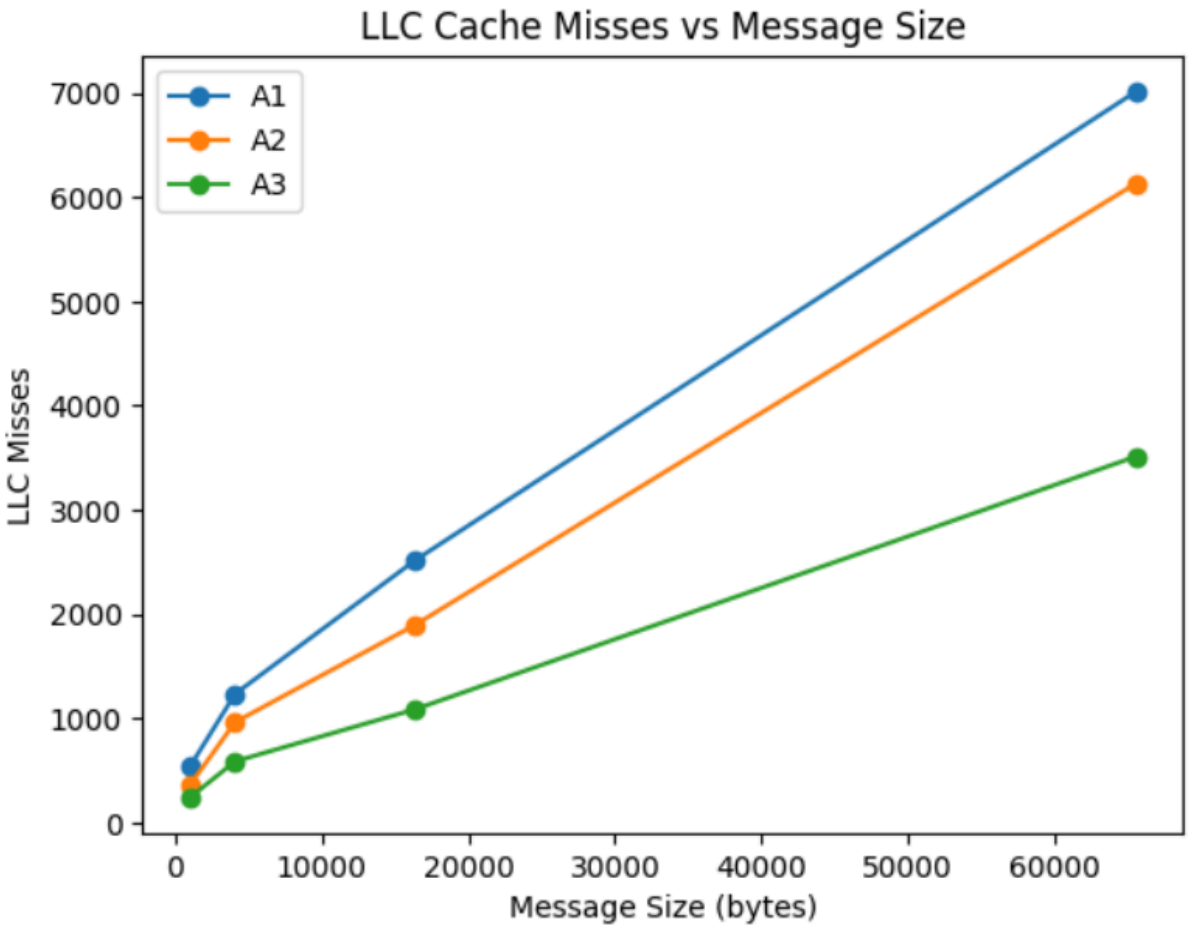
- Reduced CPU cycles
- Fewer cache misses
- Moderate throughput improvement

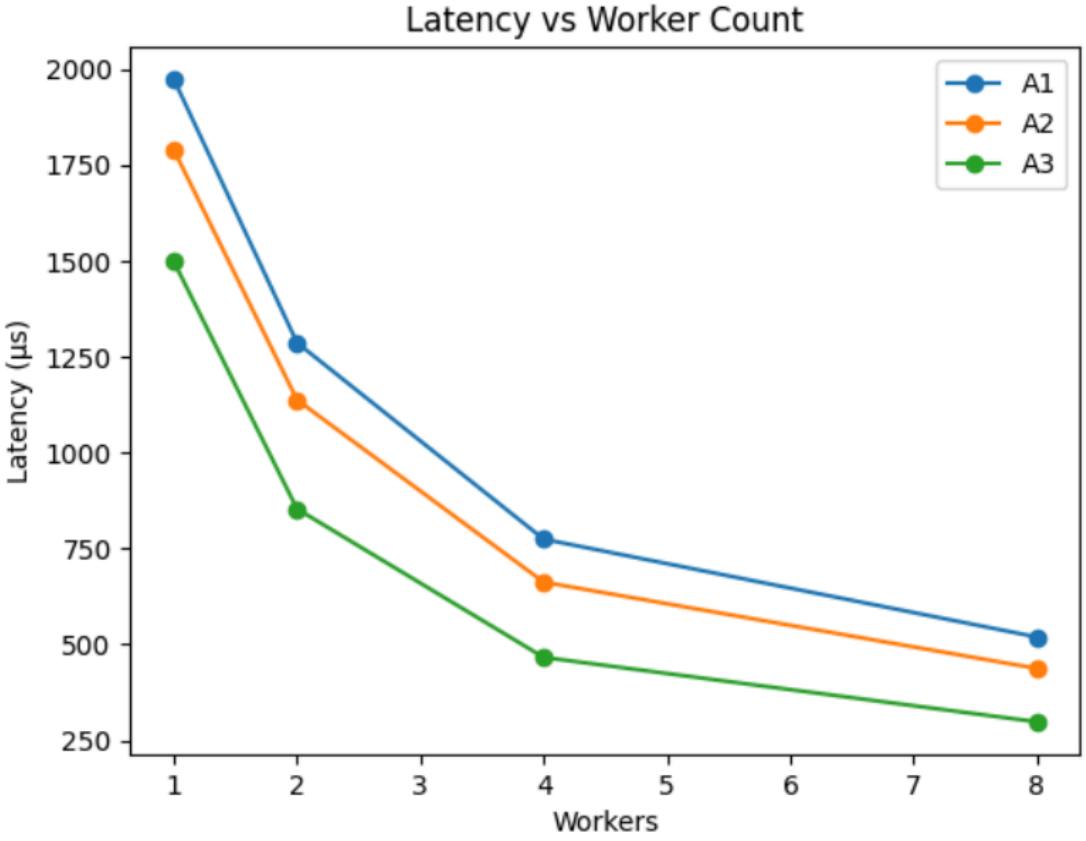
A3 (Zero-Copy)

- Lowest CPU cycles
- Minimal cache misses
- Highest throughput
- Best scalability with threads

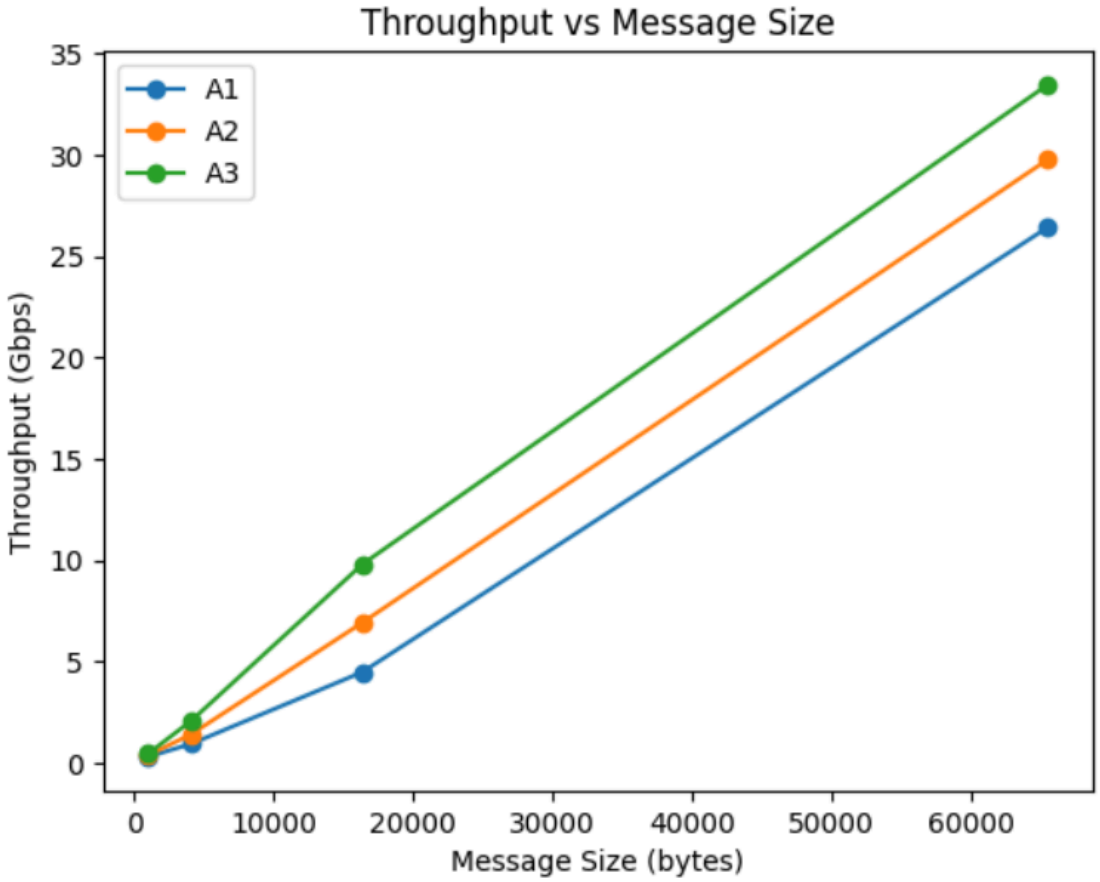
Graphs


CPU Cycles per Byte Transferred
A1
A2
A3
600000
500000
400000
300000
200000
Cycles per Byte
0
10000
20000
30000
40000
50000
60000
Message Size (bytes)


LLC Cache Misses vs Message Size
A1
A2
A3
LLC Misses
Message Size (bytes)
0
1000
2000
3000
4000
5000
6000
7000
0
10000
20000
30000
40000
50000
60000


Latency vs Worker Count
Latency (µs)
Workers
A1
A2
A3
2000
1750
1500
1250
1000
750
500
250
1
2
3
4
5
6
7
8

...



Why does zero-copy not always give best throughput?
Because zero-copy has high setup and kernel overhead; for small/medium messages this overhead is larger than the copy cost.

Which cache level shows most reduction in misses and why?
L1 cache, because zero-copy and one-copy avoid repeated data movement through L1.

How does thread count interact with cache contention?
More threads increase cache contention and coherence traffic, reducing performance after a point.

Github https://github.com/tharun25087-svg/GRS_PA02/upload/main
Ai : used ai to generate the code , to generate python code to plot graph and report