PA02 — Network I/O Primitives: Final Report

Name: Ruchir Jain
Roll No: MT25080

Abstract

This report documents the implementations, experiments, and analysis for three network I/O approaches: Two-Copy (A1), One-Copy Scatter-Gather (A2), and Zero-Copy (A3). It explains why performance differs across message sizes and thread counts, presents key plots, and answers the assignment questions in plain language.

1 Objectives

- **Goal:** Measure and compare throughput, latency, CPU cost, and cache behavior of A1/A2/A3 across message sizes and thread counts.
- **Deliverables:** Working client/server code, measurement CSV (`MT25080_measurements.csv`), plotting script (`MT25080_Part_D_Plots.py`), and this report.

2 Experimental Setup

2.1 Files Used

- `MT25080_Part_A_Client.c`
- `MT25080_Part_A1_Server.c`
- `MT25080_Part_A2_Server.c`
- `MT25080_Part_A3_Server.c`
- `MT25080_measurements.csv`
- `MT25080_Part_D_Plots.py`

2.2 Generating Plots

```
python MT25080_Part_D_Plots.py
```

3 Implementations (High-Level)

3.1 A1 — Two-Copy (Standard)

The sender builds a contiguous `send_buffer` using `memcpy` and calls `send()`. This performs two copies: assembling fields into a buffer, then copying from user space to the kernel socket buffer.

3.2 A2 — One-Copy (Scatter-Gather)

The sender uses `struct iovec` pointing to separate fields and calls `sendmsg()`. The user-space assembly copy is eliminated; the kernel copies directly from multiple user buffers.

3.3 A3 — Zero-Copy (MSG_ZEROCOPY)

The sender uses `sendmsg()` with `MSG_ZEROCOPY`. The kernel pins user pages so the NIC can DMA directly from user memory. This avoids an explicit kernel copy but introduces VM and bookkeeping overhead.

4 Key Results

- **Throughput vs Message Size:** Throughput increases with message size for all methods. A3 performs best for large messages where copying dominates. For tiny messages, A1 can match or outperform A3 due to lower setup overhead.
- **Latency and Threads:** Latency is stable at low thread counts and increases once threads exceed physical cores due to context switching and cache contention.
- **Cache Behavior:** A1 causes the most L1 cache pollution due to `memcpy`. A3 preserves cache hierarchy by avoiding user-space copies.
- **CPU Efficiency:** A3 uses the fewest CPU cycles per byte for large messages, while A1 is relatively expensive per byte.

5 Answers to Assignment Questions

Q1: Why zero-copy does not always give best throughput?

Zero-copy reduces data movement but adds fixed overheads such as page pinning, DMA mapping, and completion bookkeeping. For small messages, these overheads dominate the cost of copying.

Q2: Which cache level shows the most reduction in misses?

The L1 data cache shows the largest reduction. Zero-copy avoids touching payload bytes in user space, reducing L1 cache pollution.

Q3: How does thread count interact with cache contention?

Increasing thread count increases shared cache pressure, especially in L3. When threads exceed physical cores, context switches and cache contention reduce throughput.

Q4: When does one-copy outperform two-copy?

One-copy (A2) outperforms two-copy (A1) at small-to-moderate message sizes (around 1KB) by eliminating the user-space assembly copy.

Q5: When does zero-copy outperform two-copy?

Zero-copy outperforms two-copy when payload size is large enough that copy costs dominate, typically tens of KB (around 16–32KB), depending on hardware.

Q6: Why is zero-copy slower at 1KB?

The overhead of page pinning, DMA mapping, and completion handling dominates at small message sizes, whereas modern `memcpy` is highly optimized.

6 Plots

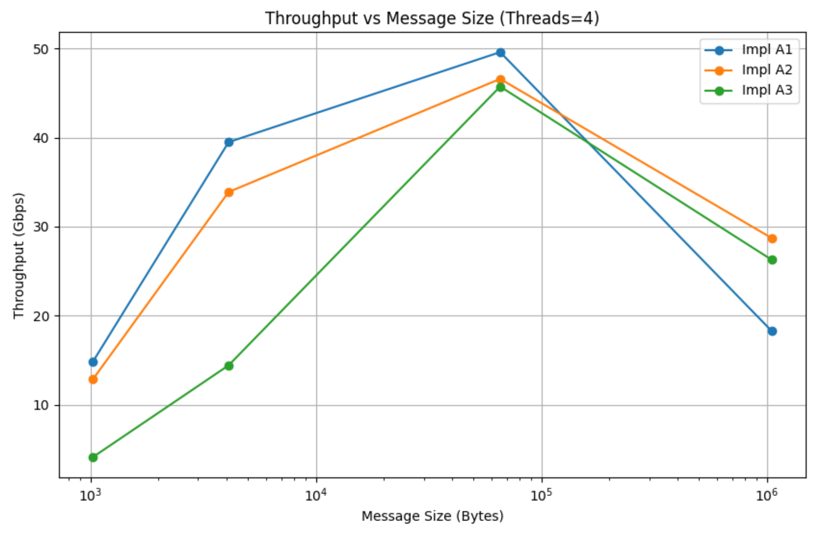

Figure 1: Throughput vs Message Size

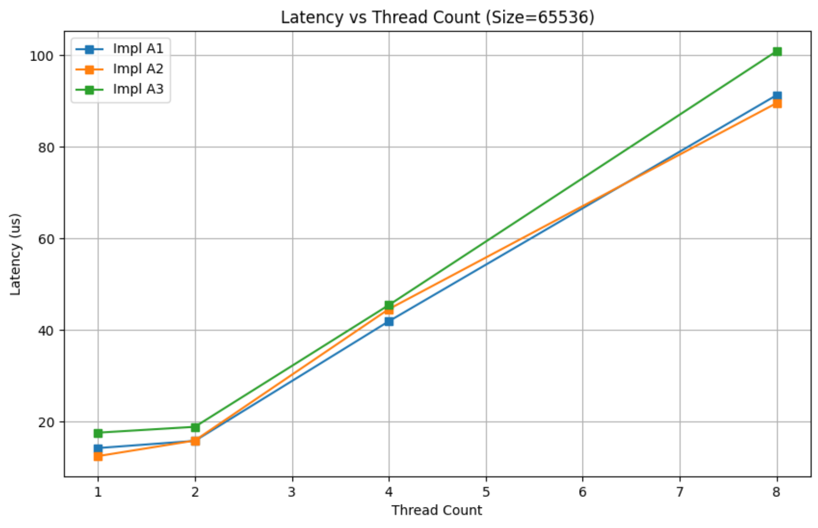

Figure 2: Latency vs Thread Count

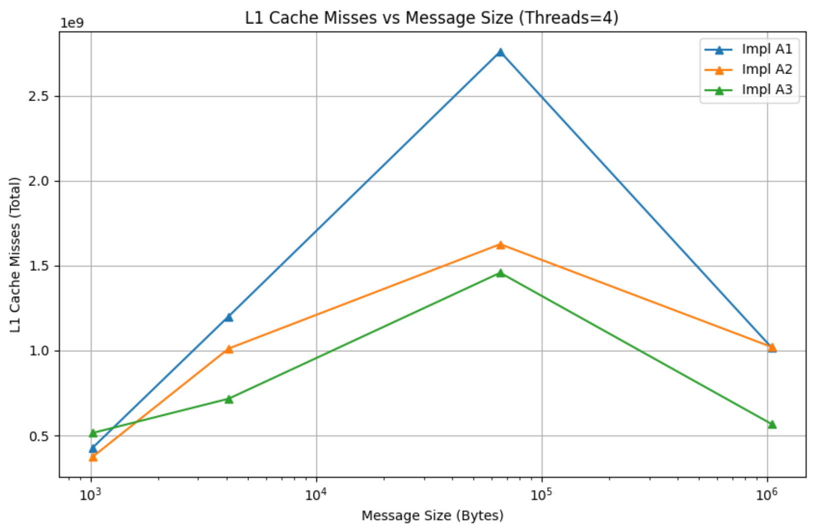

Figure 3: L1 D-Cache Misses vs Message Size

Message Size	A1 (Gbps)	A2 (Gbps)	A3 (Gbps)	Winner
1 KB	7.3510	5.7079	1.7577	A1
4 KB	20.2203	17.4305	5.8493	A1
64 KB	36.4870	41.5983	29.5257	A2
1 MB	42.7131	49.1505	39.5473	A2

Table 1: Throughput comparison for single-thread runs

7 Selected Measurement Tables

7.1 Threads = 1

7.2 Threads = 4

Message Size	A1 (Gbps)	A2 (Gbps)	A3 (Gbps)	Winner
1 KB	14.8363	12.8595	4.1272	A1
4 KB	39.4902	33.8931	14.4344	A1
64 KB	49.5895	46.5879	45.7253	A1
1 MB	18.2788	28.7142	26.2782	A2

Table 2: Throughput comparison for four-thread runs

8 AI Usage Declaration

Components

1. Code generation for initial client/server skeletons
2. Plotting script generation using matplotlib

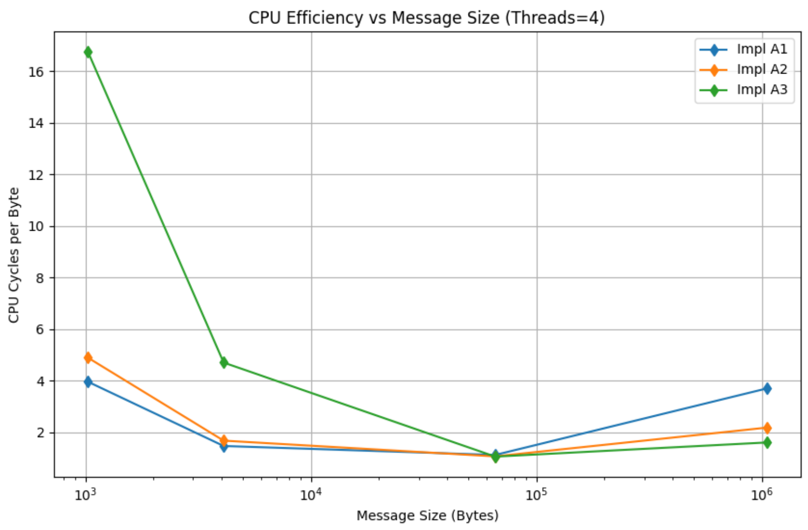

Figure 4: CPU Cycles per Byte vs Message Size

Prompts Used

- "Generate a C server using sendmsg with iovec for scatter-gather."
- "Create a python script to plot throughput vs message size from hardcoded dictionary data."
- "Explain why MSG_ZEROCOPY is slower for small messages."

Repository

https://github.com/Rjain1002/GRS_PA02