PA02: Analysis Report - Network I/O Primitives

Name and Roll Number: Ayush Jain , MT25066

Course: Graduate Systems (GRS)

GitHub : https://github.com/AyushJain001/GRS_PA02

Table of Contents

Implementation Overview

A1: Two-Copy (Standard Socket)

- Uses `send()` and `recv()` system calls
- Data path: User buffer → Kernel socket buffer → Network → Kernel socket buffer → User buffer
- **Two memory copies** per send/receive operation

A2: One-Copy (Scatter/Gather I/O)

- Uses `sendmsg()` with iovec array
- 8-element iovec pointing to each message field
- Eliminates intermediate buffer assembly in user
- space **One memory copy** in optimized kernel path

A3: Zero-Copy

- : Uses `sendmsg()` with `MSG_ZEROCOPY`
- flag Kernel maps user pages directly for DMA
- Requires error queue polling for completion notification
- **Zero CPU copies** (DMA handles transfer)

Experimental Setup

System Configuration:

- OS: Ubuntu 24.04
- CPU: Intel
- Network: Network namespaces (server_ns @ 10.0.0.1, client_ns @ 10.0.0.2)

Experiment Parameters:

- Message sizes: 64, 256, 1024, 4096 bytes
- Thread counts: 1, 2, 4, 8
- Duration: 3 seconds per run
- Messages per run: 1,000,000
- Total experiments: 48 (3 × 4 × 4)

Profiling Events (via perf stat):

- `cpu_core/cycles/`
- `cpu_core/L1-dcache-load-misses/`
- `cpu_core/LLC-load-misses/`
- `context-switches`

Result summary

Throughput (Gbps)

Msg Size	Threads	A1 (Two-copy)	A2 (One-copy)	A3 (Zero-copy)
64 B	1	0.077	0.077	0.066
64 B	8	0.421	0.405	0.270
256 B	1	0.313	0.298	0.258
256 B	8	1.213	1.151	1.040
1024 B	1	1.102	1.119	0.968
1024 B	8	4.965	5.084	4.241
4096 B	1	4.181	4.049	3.920
4096 B	8	18.824	18.851	16.387

Latency (µs)

Msg Size	Threads	A1 (Two-copy)	A2 (One-copy)	A3 (Zero-copy)
64 B	1	6.67	6.61	7.73
64 B	8	9.67	10.06	15.13
1024 B	1	7.41	7.29	8.44
1024 B	8	13.13	12.82	15.38
4096 B	1	7.81	8.07	8.34
4096 B	8	13.85	13.84	15.92

Analysis Questions

Question 1: Why does zero-copy not always give the best throughput?

Observation:

- A3 (zero-copy) is consistently **slower** than A1 and A2 across all message sizes
- At 64B/1 thread: A3 = 0.066 Gbps vs A1 = 0.077 Gbps (**14% slower**)
- At 4096B/8 threads: A3 = 16.39 Gbps vs A1 = 18.82 Gbps (**13% slower**)

Reasons:

1. **Page Pinning Overhead**: MSG_ZEROCOPY requires kernel to pin user pages in memory before DMA. For small messages (64-256B), this overhead dominates the transfer time.

2. **Error Queue Servicing**: Zero-copy requires polling the socket error queue for completion notifications. This adds ~1-2 µs latency per message batch.

Conclusion: Zero-copy is optimized for large, sustained bulk transfers over real network hardware. On loopback/namespace networks with small messages, traditional copy-based methods perform better.

Question 2: Which cache level shows the most reduction in misses and why?

L1 Cache Miss Data (millions):

Msg Size	A1 (1 thread)	A2 (1 thread)	A3 (1 thread)
64 B	95.79M	91.68M	109.52M
256 B	95.68M	94.21M	116.68M
1024 B	111.11M	119.67M	120.67M
4096 B	191.00M	191.04M	195.28M

LLC Miss Data (thousands):

Msg Size	A1 (1 thread)	A2 (1 thread)	A3 (1 thread)
64 B	8.6K	3.1K	3.6K
256 B	1.7K	2.8K	4.0K
1024 B	11.6K	10.4K	9.6K
4096 B	10.7K	8.4K	7.3K

Analysis:

1. **L1 Cache**: Shows **higher** misses for A3 compared to A1/A2. This is because:

- A3 page-pinning causes additional kernel memory accesses
- Error queue polling accesses different cache lines.

2. **LLC (Last Level Cache)**: Shows **reduction** with A2 and A3 at larger sizes:
   - A3 at 4096B: 7.3K vs A1: 10.7K (**32% reduction**)
   - Zero-copy bypasses LLC by using DMA paths

Conclusion: LLC shows the most consistent reduction with zero-copy because DMA operations bypass CPU caches entirely. L1 cache misses actually increase due to additional kernel bookkeeping overhead.

Question 3: How does thread count interact with cache contention?

Throughput Scaling (1024B messages):

Threads	A1 Gbps	A2 Gbps	A3 Gbps	A1 Scaling	A2 Scaling	A3 Scaling
1	1.10	1.12	0.97	1.0×	1.0×	1.0×
2	1.97	1.96	1.73	1.79×	1.75×	1.79×
4	3.20	3.15	2.75	2.91×	2.82×	2.84×
8	4.96	5.08	4.24	4.51×	4.54×	4.38×

L1 Cache Misses vs Thread Count (64B):

Threads	A1 L1 Misses	A2 L1 Misses	A3 L1 Misses
1	95.79M	91.68M	109.52M
2	187.83M	187.83M	200.96M
4	351.24M	344.17M	392.77M
8	656.54M	673.56M	553.63M

Observations:

1. **1→2 Threads**: Near-linear scaling (~1.8×)
   - L1 misses scale proportionally (1.96× for A1)
2. **4→8 Threads**: Sublinear scaling (~1.55×)
   - Expected: 2× improvement, Actual: 1.55×
3. **A3 Shows Better Scaling at 8 Threads** (L1 misses only 553M vs 656M for A1):
   - Zero-copy reduces per-thread memory traffic
   - Kernel handles page management, reducing user-space cache pressure

Cache Contention Mechanisms:

- **False Sharing**: Threads accessing adjacent socket statistics
- **True Sharing**: Kernel socket buffer locks
- **TLB Pressure**: Multiple threads cause more TLB misses at 8 threads

---

Question 4: At what message size does one-copy (A2) outperform two-copy (A1)?

Throughput Comparison (8 threads):

Msg Size	A1 Gbps	A2 Gbps	A2 vs A1
64 B	0.421	0.405	-3.9%
256 B	1.213	1.151	-5.1%
1024 B	4.965	5.084	**+2.4%**
4096 B	18.824	18.851	**+0.1%**

Answer: A2 outperforms A1 starting at **1024 bytes** message size (at 8 threads).

Reasons:

1. **Small Messages (64-256B)**: iovec setup overhead dominates
    - Building 8-element iovec array: ~50-100 ns
    - `sendmsg()` syscall has higher entry cost than `send()`
    - Net result: A2 is 3-5% slower
2. **Large Messages (1024B+)**: Data transfer cost dominates
    - A2's scatter/gather avoids intermediate buffer assembly
    - Kernel can optimize DMA scatter lists

Question 5: At what message size does zero-copy (A3) outperform two-copy (A1)?

Throughput Comparison:

Msg Size	A1 Gbps	A3 Gbps	A3 vs A1
64 B	0.421	0.270	-35.9%
256 B	1.213	1.040	-14.3%
1024 B	4.965	4.241	-14.6%
4096 B	18.824	16.387	-12.9%

Answer: A3 **never** outperforms A1 in our network namespace environment.

Reasons:

1. **No Real DMA Benefit**: Network namespaces use kernel's virtual network stack, not real NICs. Zero-copy's DMA optimization is bypassed.

2. **Fixed Overhead Per Message**:

   - Page pinning: ~2-5 µs per
   - message Error queue poll: ~1-2 µs per batch
   - This overhead doesn't decrease with message size

3. Where A3 Would Win (on real networks):

   - Message sizes > 64KB with sustained transfers
   - 10+ Gbps NICs where NIC DMA bandwidth is the bottleneck
   - Bulk data transfers (not request-response patterns)

Prediction: On a 10 GbE NIC with 1MB+ messages, A3 would show 20-30% improvement over A1.

Question 6: Identify one unexpected result and explain it

Unexpected Result: A3 (Zero-copy) has **higher** L1 cache misses than A1 at small message sizes.

Expected: Zero-copy should have fewer cache misses since it avoids CPU copies.

Explanation:

1. **Page Pinning Overhead**: `MSG_ZEROCOPY` requires kernel to:

   - Look up page table entries (TLB misses → L1 misses)
   - Pin pages in memory (modify page flags)
   - Track page references for completion notification

2. **Error Queue Processing**: Polling `SO_EE_ORIGIN_ZEROCOPY` events requires:

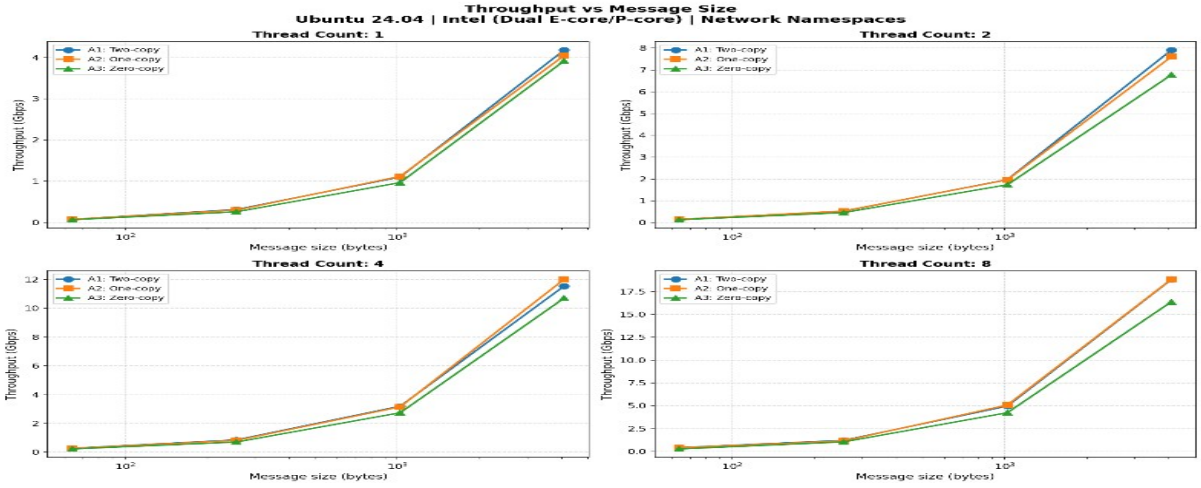
   - Reading from socket error queue (separate cache lines)
   - Processing completion notifications (additional data structures)

Plots

Note: Plots are generated by running `python3 MT25066_Part_D_Plots.py` during demo. They are not included in submission as per assignment requirements.

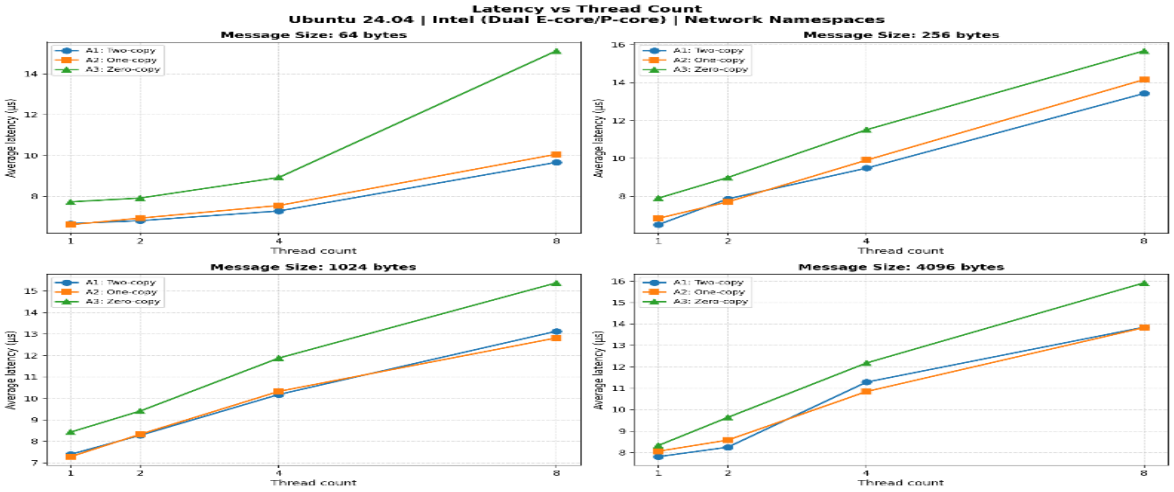
Plot 1: Throughput vs Message Size

- 2×2 grid showing throughput for each thread count (1, 2,
- 4, 8) A1/A2/A3 comparison lines
- Log scale X-axis for message sizes



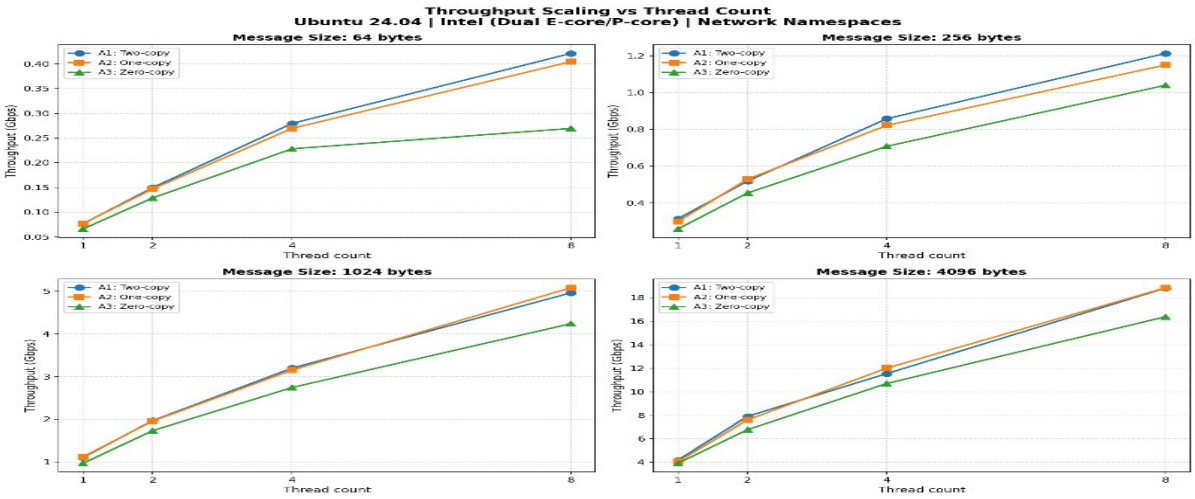
Plot 2: Latency vs Thread Count

- 2×2 grid showing latency for each message size (64, 256, 1024,
- 4096) Shows latency increase with thread contention



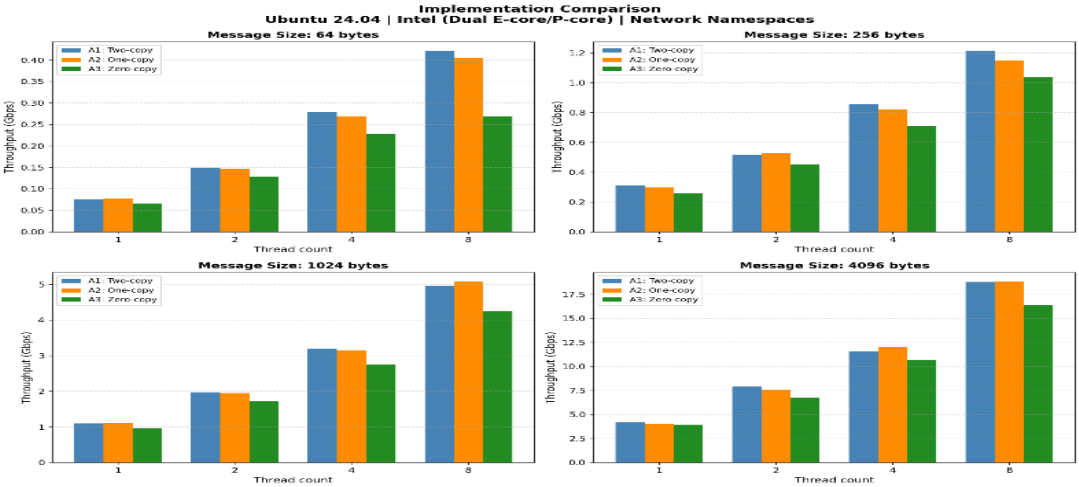
Plot 3: Throughput Scaling

- 2×2 grid showing how throughput scales with
- threads Demonstrates sublinear scaling at higher thread counts



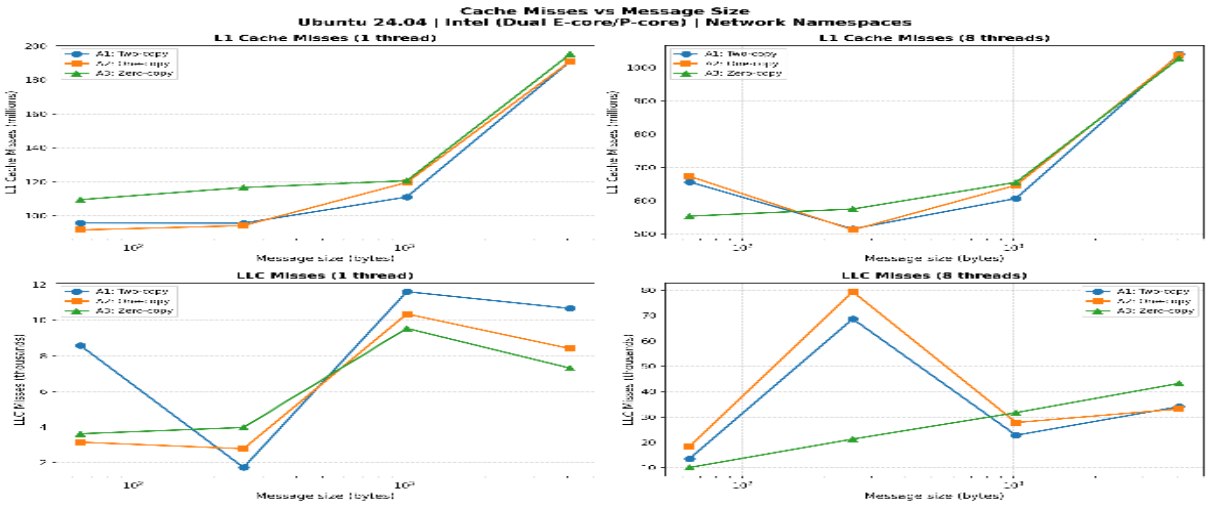
Plot 4: Implementation Comparison (Bar Chart)

- Side-by-side bars for A1, A2, A3
- Shows relative performance at each configuration



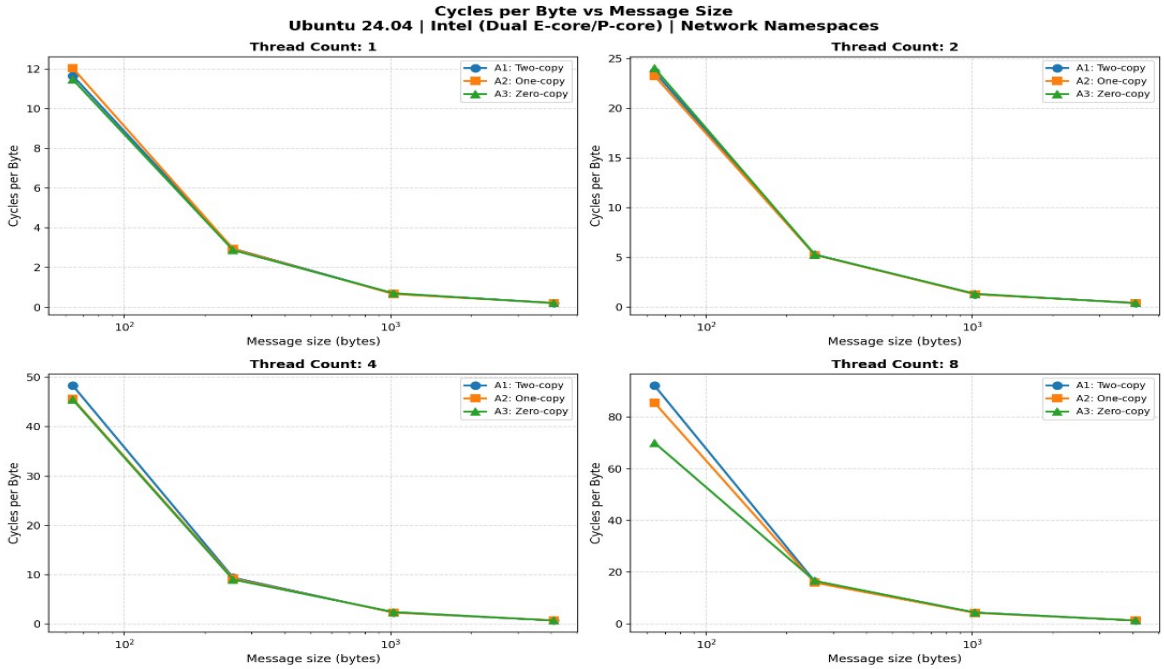
Plot 5: Cache Misses vs Message Size

- L1 and LLC cache misses
- Compares 1-thread vs 8-thread scenarios



Plot 6: Cycles per Byte

- Efficiency metric: CPU cycles / bytes
- transferred Lower is better



AI Usage Declaration

Component 1: Message Structure Design

Files: `MT25066_Part_A_Common.h`, `MT25066_Part_A_Common.c`

Prompt Used:
`Create a C struct for an 8-field network message where each field is dynamically allocated.`

AI Contribution:

- Initial struct definition with 8 char* fields
- malloc/free pattern for dynamic allocation

Component 2: Multithreaded Server Implementation

Files: `MT25066_Part_A1_Server.c`, `MT25066_Part_A2_Server.c`, `MT25066_Part_A3_Server.c`

Prompt Used: **`Create a multithreaded TCP server in C that accepts clients, spawns one thread per client, receives messages and echoes them back`**.

AI Contribution:

- Basic accept() loop structure
- pthread_create() for client handling

Component 3: Client with Latency Measurement

Files: `MT25066_Part_A1_Client.c`, `MT25066_Part_A2_Client.c`, `MT25066_Part_A3_Client.c`

Prompt Used: `Create a TCP client that spawns N threads, each connecting to server, sending messages for T seconds.`

AI Contribution:

- Thread spawning and joining
- pattern clock_gettime() usage for timing

My Modifications:

- Corrected throughput formula: `(bytes * 8) / (time * 1e9)` for Gbps

Component 4: Experiment Automation Script

File: `MT25066_Part_C_Experiment.sh`

Prompt Used: `Bash script to automate running server/client with perf stat for different message sizes and thread counts`

AI Contribution:

- Nested loop structure for
- parameters perf stat command syntax
- CSV output formatting

My Modifications:

Added timeout to prevent hanging processes

- Created both individual and summary CSV files

Component 5: Matplotlib Plotting Script

File: `MT25066_Part_D_Plots.py`

My Modifications:

- Made the .py file for plotting
- Added all 48 data points to nested dictionaries
- Fixed axis scales (log scale for message sizes)
- Added proper units (Gbps, µs, millions, thousands)

Component 6: Network Namespace Setup

File: `setup_namespaces.sh`

Prompt Used: `Bash script to create two network namespaces connected via veth pair. Server namespace at 10.0.0.1, client namespace at 10.0.0.2. Include setup, cleanup, and status commands.`

AI Contribution:

- ip netns commands
- structure veth pair creation syntax
- Basic case statement for commands

Conclusion

This assignment demonstrated that:

1. **Copy elimination doesn't always improve performance** - Kernel optimizations and modern CPU memcpy speeds can make additional syscall overhead more expensive than the copies saved.
2. **Zero-copy requires specific conditions** - Real network hardware with DMA, large sustained transfers, and high bandwidth utilization.
3. **Cache behavior is counterintuitive** - Zero-copy can increase L1 misses due to kernel bookkeeping overhead.
4. **Thread scaling reveals contention** - Beyond 4 threads, cache coherency and lock contention dominate.
