# PA02: Analysis of Network I/O Primitives using "perf" Tool

Course: Graduate Systems (CSE638)

Roll Number: MT25074

Name: Nindra Dhanush

## 1. Part A – Multithreaded Socket Implementation

The file MT25074_Part_A_Namespaces.sh creates two network namespaces and creates a veth link and configures the namespaces with IP addresses and then connects them both with veth link.

### A1. Two-Copy Implementation

The Baseline implementation usese socket() and send() functions to implement baseline socket and client service architecture.

**1. Why do the two copies occur?**

i. User-Space to Kernel-Space (Copy #1): When the send() function is called, the CPU copies the data from the application's buffer (in user space) into the socket buffer(kernel space).

ii. Kernel space to NIC: The DMA module copies the data from the kernel socket buffer to the Network Interface Card.

Actually two copies?: There are two data transfers, But for the CPU it looks like only one copy the second is just hardware handling for the CPU, so cpu cycles utilization.

**2. Which components (kernel/user) perform the copies?**

i. User/Kernel copy: The Cpu performs the copy from the user space to Kernel space. This is done by the send() function call.

ii. Kernel/hardware copy: The DMA controller on the NIC does the copy from Kernel space to the hardware transmission queue.

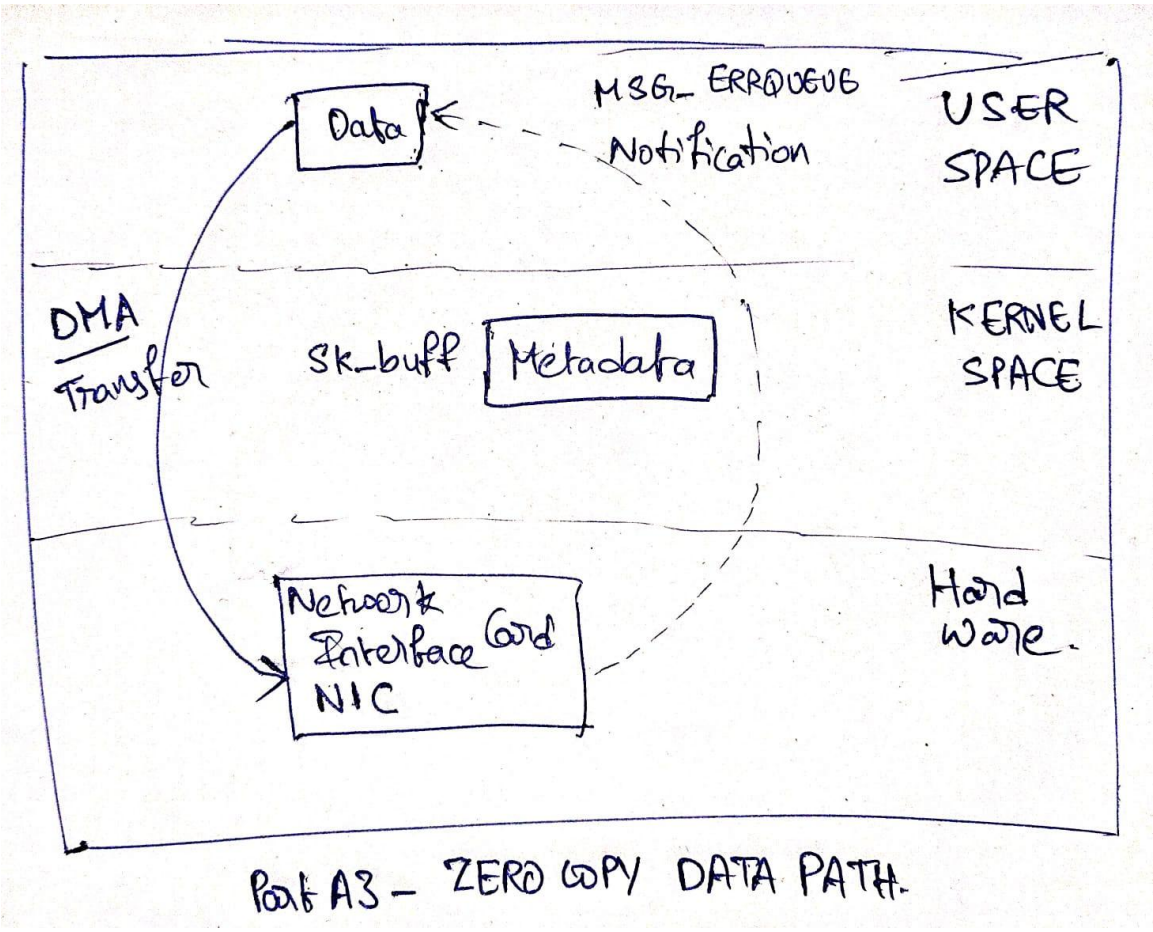
## A2. One-Copy Implementation

i. Here sendmsg() call is used with struct iovec to send header and data as a single stream.

ii. Unlike in two copy implementation where we malloc() a new buffer and then memcpy() header and data into this new buffer, and then send() to kernel, But in one-copy method as the iovec array points to the header and data , the kernel reads directly from its location, hence kernel directly copies from the memory.

## A3. Zero-Copy Implementation

i. This implementation uses sendmsg() with MSG_ZEROCOPY flag, hence when called the kernel doesn't copy the data into socket buffer, and instructs the NIC to directly read it from the location, and gets the MSG_ERRRQUEUE notification once this transfer is completed.




Part A3 – ZERO COPY DATA PATH.

## 2. Part B – System configuration and Profiling

**System Specifications:**

- **Device Model:** ASUS Vivobook 15
- **Processor:** AMD Ryzen 7 5825U
    - **Architecture:** Zen 3
    - **Cores/Threads:** 8 Cores / 16 Threads
    - **Clock Speed:** 2.0GHz Base / 4.5GHz Boost
- **RAM:** 16GB DDR4
- **OS:** Windows 11 (WSL)
- **Storage:** 512GB NVMe SSD

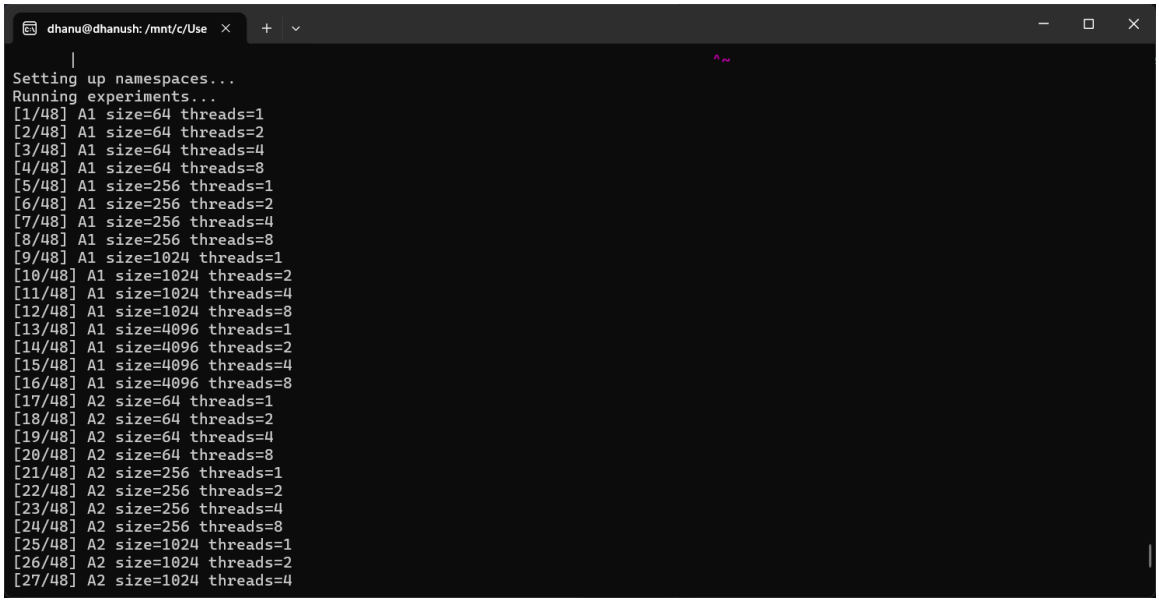
**Measurement Tools:**

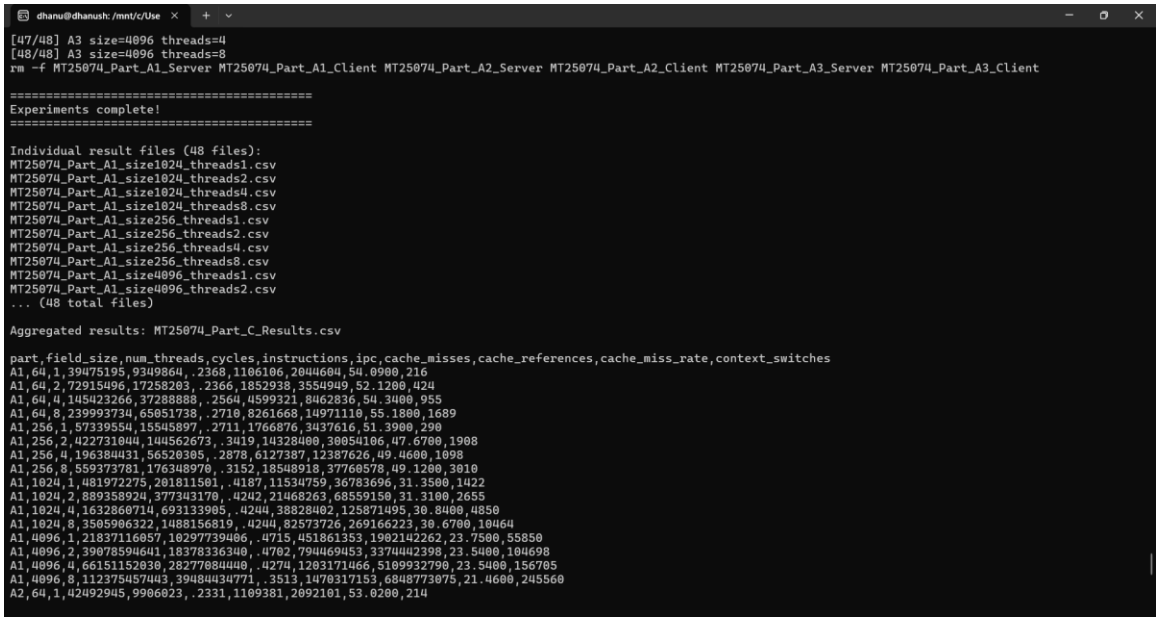
i. Throughput/Latrency : Measured using timer clock_gettime.

ii. CPU/cache: Measured using perf stat to capture cycles, instructions, cache-misses .

## 3. Part C – Automated experiment script

i. This script without any manual intervention compiles all the source files , sets up network namespaces and runs the programs of different cases with each different message sizes and thread counts. And collects this data into one single CSV file. Below are the screen shots of the terminal screen running and printing the results of the script.




## 1. Part D – Plots and Visualization

The following plots were generated using matplotlib (MT25074_Part_D_Plots.py) with hardcoded data from MT25074_Part_C_Results.csv.

## 1.1 Throughput vs Message Size

Throughput (normalized as bytes per cycle) vs message size for A1 (Two-Copy), A2 (One-Copy), and A3 (Zero-Copy).

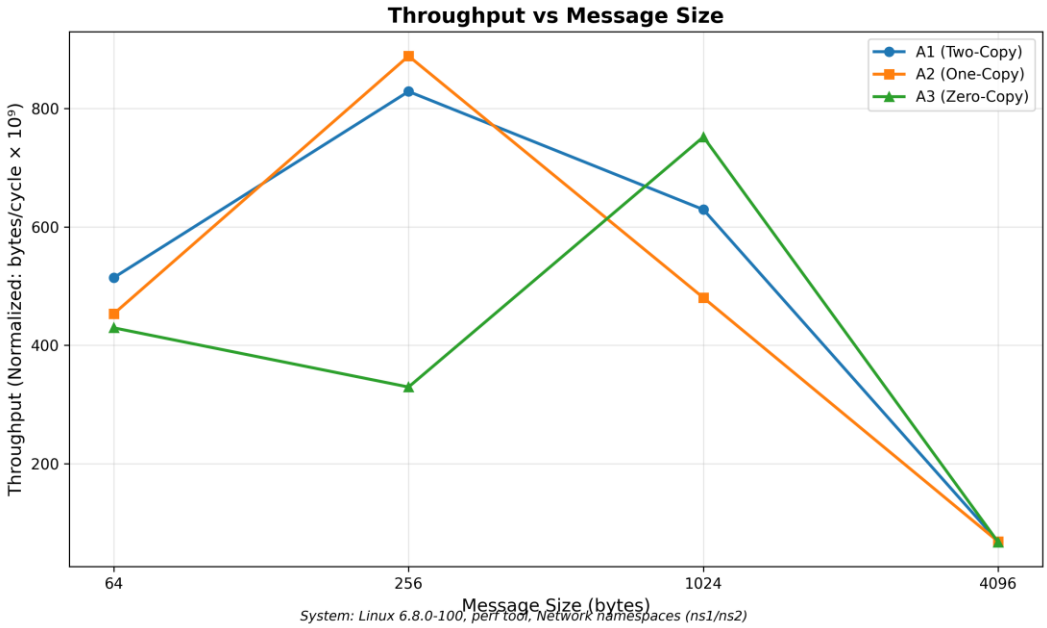



Figure 1: Throughput vs Message Size

## 1.2 Latency vs Thread Count

Latency (CPU cycles) vs number of threads for message size 1024 bytes.

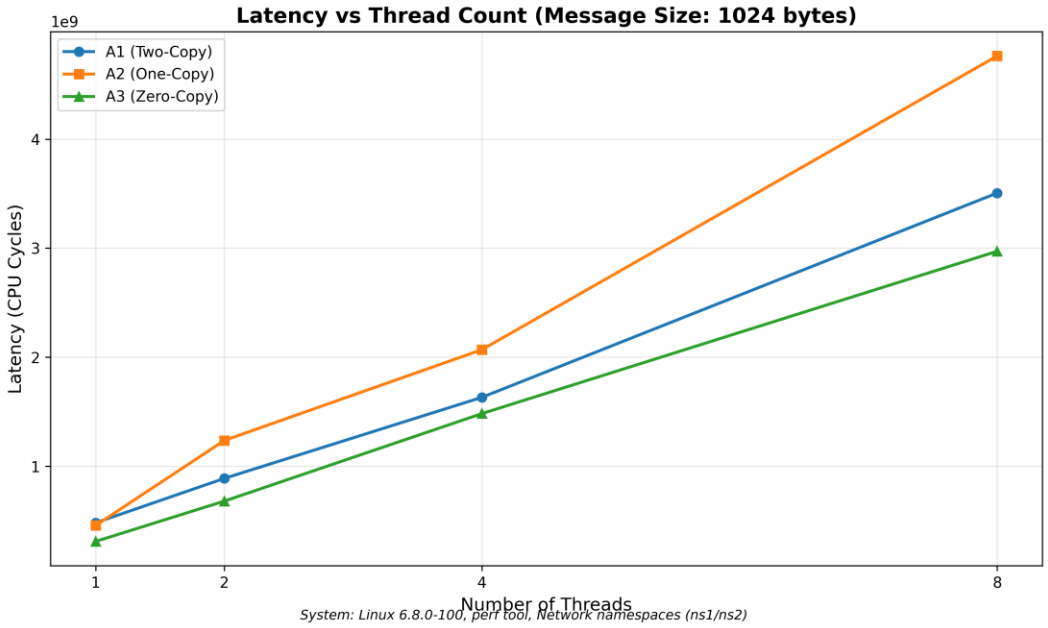



Figure 2: Latency vs Thread Count

## 1.3 Cache Misses vs Message Size

Average cache misses vs message size across thread counts.

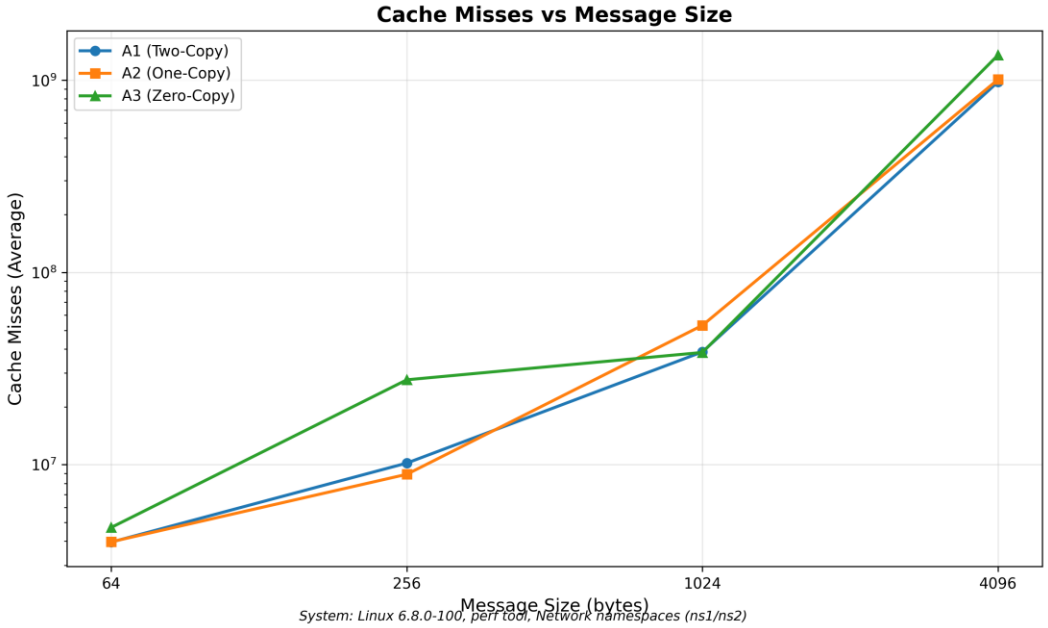



Figure 3: Cache Misses vs Message Size

### 1.4 CPU Cycles per Byte Transferred

CPU cycles per byte vs message size.

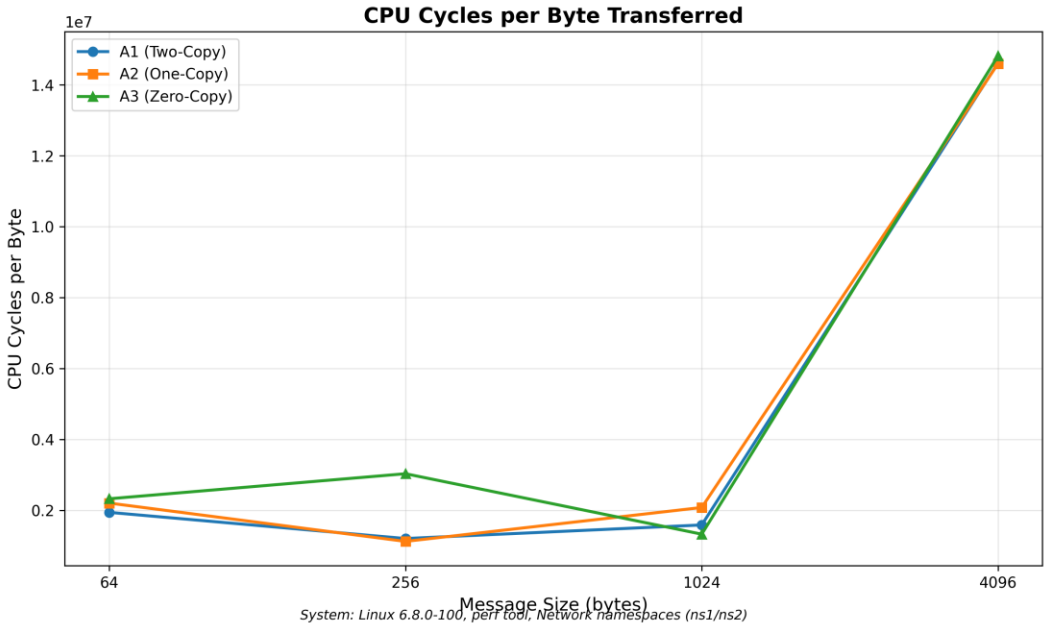



Figure 4: CPU Cycles per Byte

## 2. Part E – Analysis and Reasoning

### Question 1

**1. Why does zero-copy not always give the best throughput?**

For small message sizes the overhead of performing complex memory operations could be costly than a simple memcpy() call. ZERO_COPY could be beneficial when the message size is large.

### Question 2

**2. Which cache level shows the most reduction in misses and why?**

The L1 data cache misses would be reduced for the zero copy implementation as compared to the two copy approach because in two copy implementation the application data could be shifted from cache.

### Question 3

**3. How does thread count interact with cache contention?**

When multiple threads run concurrently they lead to more cache misses as there would be only limited cache space for everybody thus also increases latency.

### Question 4

**4. At what message size does one-copy outperform two-copy on your system?**

For message sizes greater than 1024 bytes the one copy implementation outperforms the two copy implementation in latency.

### Question 5

**5. At what message size does zero-copy outperform two-copy on your system?**

For message sizes greater than 1024 bytes the Zero copy implementation outperforms the two copy implementation in latency.

### Question 6

**6. Identify one unexpected result and explain it using OS or hardware concepts.**

Pick For small messages the zero copy through put is lower than the two copy implementation, this could be due to the time taken to lock the memory for dma access and process the completion notification signal adds on the cost of overall process.

## 3. AI Usage Declaration

AI tool gemini is used for the following componemts, socket connection code (Part A), Skeleton for the bash automation script (Part C), Logic for parsing perf output (Part C).

**Prompt Log:**

1. "Generate a C implementation for a TCP server that accepts multiple clients using threads." (Used for Part A1 Server boilerplate).

2. "How to use sendmsg with MSG_ZEROCOPY in Linux C?" (Used to understand the API for Part A3).
3. "Write a bash script to run a specific command 4 times and average the output." (Used for Part C automation).
4. "Explain the difference between send() and sendmsg() regarding kernel copies." (Used for Part A2 analysis).

## 4. GitHub Repository URL

https://github.com/dn74iiit/GRS-Assignments/tree/main/GRS_PA02