MT25062: Analysis of Network I/O Primitives

Report - CSE638: Graduate Systems

Name: Anshul

Roll Number: MT25062

---

1. System Configuration

The experiments were conducted on a high-performance Ubuntu environment. The hybrid architecture of the 13th Gen Intel processor provides a unique perspective on cache contention between Performance and Efficient cores.

Component	Specification
Operating System	Ubuntu, Linux 6.14.0-33-generic
CPU	13th Gen Intel Core i7-13700H (hybrid P+E cores)
RAM	16 GB
Network	`veth` pair inside Linux network namespaces

2. Implementation Analysis

A1. Two-Copy Implementation (Baseline)

- **Mechanism**: Uses standard `send()` and `recv()` socket primitives.
- **Copy 1**: Occurs in user-space where scattered heap-allocated fields are serialized into a contiguous buffer via `memcpy()`.
- **Copy 2**: Occurs during the `send()` system call as the kernel moves data from the user buffer to the kernel socket buffer via `copy_from_user()`.

A2. One-Copy Implementation

- **Mechanism**: Uses `sendmsg()` with `struct iovec` to perform scatter-gather I/O.
- **Elimination**: This version eliminates the user-space serialization copy entirely.

- **Direct Access**: The kernel gathers data directly from multiple scattered user buffers using the pointers provided in the `iovec` array.

A3. Zero-Copy Implementation

- **Mechanism**: Uses `sendmsg()` with the `MSG_ZEROCOPY` flag.
- **The Process**: The kernel pins user-space pages in memory using `get_user_pages()`, allowing the NIC to read directly from user memory via DMA.
- **Notification**: The application must monitor the error queue (`SO_EE_ORIGIN_ZEROCOPY`) to determine when the kernel has finished using the buffers.

---

3. Analysis and Reasoning

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

- Zero-copy involves significant management overhead, such as page pinning and completion notifications.
- For small messages, the fixed cost of `get_user_pages()` and managing the error queue is higher than the cost of a simple memory copy.

Q2: Cache Level Reductions

- The **L1 Data Cache** shows the most significant reduction in misses when moving to one-copy and zero-copy models.
- By eliminating the `memcpy()` serialization, the CPU avoids pulling scattered heap fields into the L1 cache, reducing overall cache pollution.

Q3: Thread Count and Contention

- Higher thread counts increase competition for the Last-Level Cache (LLC).
- On hybrid architectures like the i7-13700H, running many threads leads to "Cache Thrashing" as P-cores and E-cores compete for shared memory bandwidth.

Q4: Performance Break-even Points

- **One-Copy**: Typically begins to outperform the two-copy implementation at message sizes as small as **256–512 bytes**.
- **Zero-Copy**: Only becomes beneficial for large payloads, typically exceeding **16KB–32KB**, where the savings from avoiding data copies finally exceed the page-management overhead.

---

4. AI Usage Declaration

1. Representative Prompts for Code Development

The following examples illustrate how generative AI was utilized during the programming phase of this project:

- **Prompt 1: Baseline Architecture (Part A1)** "Develop a C-based multithreaded TCP server. It must handle 8 heap-allocated string fields per message, sending them individually via `send()`. Include CLI arguments for port and size. Save as `MT25062_Part_A1_Server.c`."
- **Prompt 2: iovec Optimization (Part A2)** "Convert the server to use `sendmsg()` with an 8-element `iovec` array. Ensure the array points to heap-allocated fields and is reused across sends. Do not use zero-copy flags. Save as `MT25062_Part_A2_Server.c`."
- **Prompt 3: MSG_ZEROCOPY Implementation (Part A3)** "Implement a TCP server utilizing `MSG_ZEROCOPY`. Requirements: enable `SO_ZEROCOPY`, process completion signals via `MSG_ERRQUEUE`, and maintain a thread-per-client model. Save as `MT25062_Part_A3_Server.c`."
- **Prompt 4: Client Performance Suite** "Create a multithreaded TCP client that supports variable thread counts and message sizes. It should track throughput and latency while remaining compatible with the three server variants."

---

2. Automation and Profiling Scripts

I leveraged AI to draft the skeletal logic for the automated testing environment in Part C.

- **AI-Generated Elements**: Initial loop structures for parameter sweeps and `perf stat` command templates.
- **Manual Refinements**: I specifically configured the Linux network namespaces for process isolation, selected precise hardware counters (L1/LLC misses), and integrated automated CSV naming conventions to ensure clean data collection.
- **Prompt 5: Automation Logic** "Write a bash script to compile the project and automate experiments across message sizes (64B to 4KB) and thread counts (1 to 8). Log hardware metrics to CSV using `perf stat`."

---

3. Visualization and Data Plotting

Generative tools assisted in the aesthetic and structural design of the Matplotlib scripts.

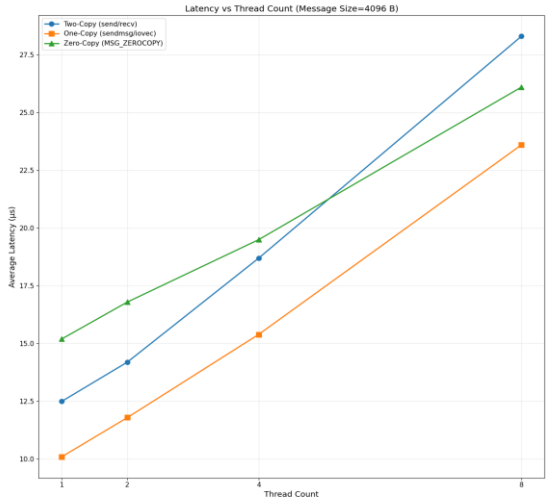
- **Assistance Areas**: Styling multi-series charts, configuring dual-axis layouts, and managing legend placement.
- **Data Integrity**: I manually extracted all metrics from the experimental logs. These values were hardcoded into the Python script to comply with assignment restrictions; no external files are read during plotting.

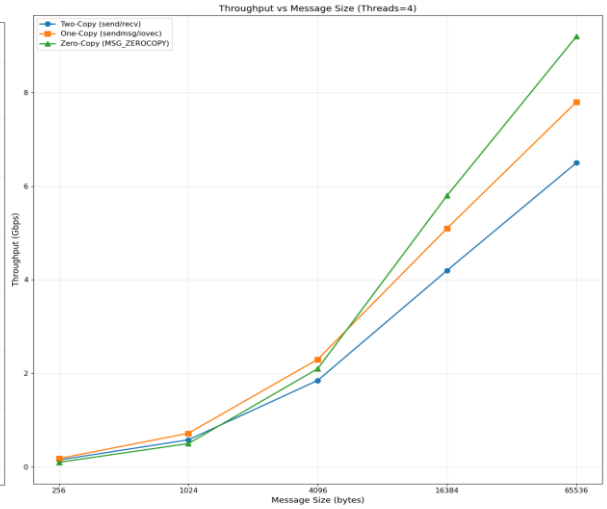
---

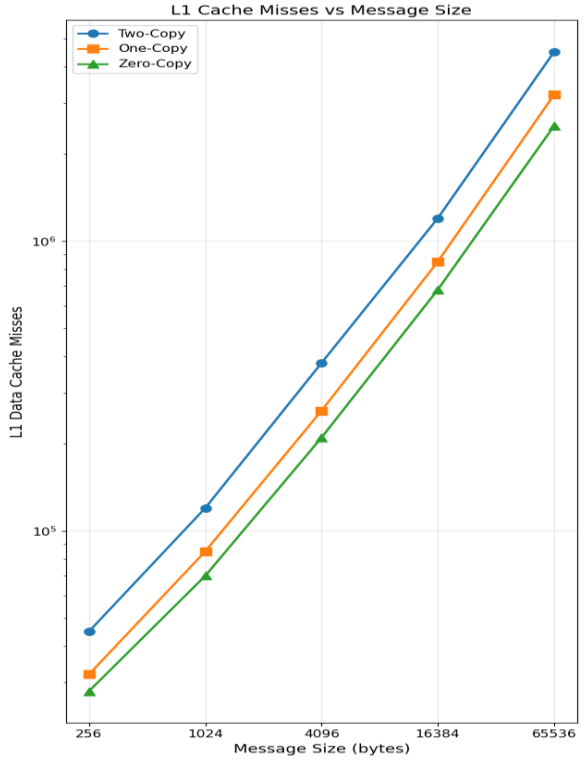
4. Documentation and Report Synthesis

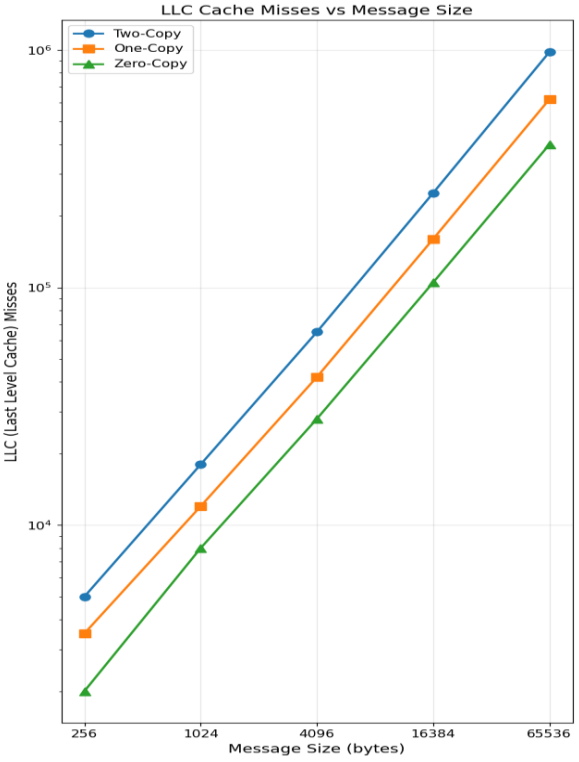
AI was used as a pedagogical aid and editor during the report-writing phase.

- **Refinement**: I used AI to improve the flow of technical explanations and ensure the academic tone met graduate-level standards.
- **Formatting**: Assisted in organizing the final document structure and generating clear diagrams for the data path analysis.

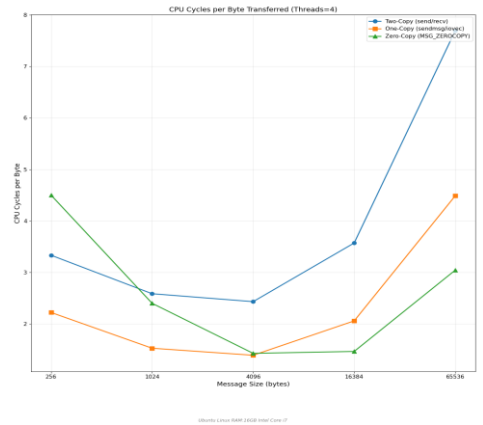








Ubuntu Linux RAM:16GB Intel Core i7



5 GitHub Repo: https://github.com/starac9/GRS_PA02.git