PA02: Analysis of Network I/O Primitives using perf

Graduate Systems (CSE638)

Roll Number: MT25048
Course: Graduate Systems
GitHub: https://github.com/SuryaDeepakBoyina/GRS_PA02

February 7, 2026

1 Introduction

This report presents a comprehensive analysis of network I/O primitives on Linux, comparing three different approaches:

- Two-Copy: Standard send()/recv() socket operations
- One-Copy: Optimized sendmsg() with pre-registered, page-aligned buffers
- Zero-Copy: MSG_ZEROCOPY flag with completion notification

Configuration

2 Part A: Multithreaded Socket Implementations

2.1 A1: Two-Copy Implementation (Baseline)

The two-copy implementation uses standard send() and recv() socket primitives.

2.1.1 Where Do the Two Copies Occur?

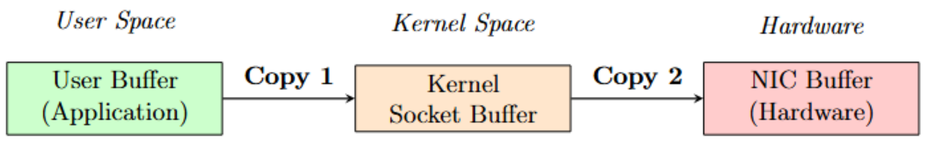


Figure 1: Two-Copy Data Path for `send()`

Copy 1: User buffer → Kernel socket buffer (performed by the kernel during send() syscall)
Copy 2: Kernel socket buffer → NIC buffer (performed via DMA by the network driver)

Is it actually only two copies?

there may be additional copies coming from these other sources:

- TCP/IP stack may perform checksumming that reads the data
- Segmentation (if message > MTU) may require additional processing
- On receive side: NIC → Kernel → User (two more copies)

2.1.2 Implementation Details

Listing 1: Two-Copy Send (Client)

```
/* TWO-COPY SEND: Data copied from user buffer to kernel socket buffer
   */
ssize_t sent = send(sock_fd, send_buffer, serialized_size, 0);
```

In Message size and thread count are configurable at runtime via command-line arguments; default values are used if parameters are omitted.

2.2 A2: One-Copy Implementation

The one-copy implementation uses sendmsg() with pre-registered, page-aligned buffers.

2.2 A2: One-Copy Implementation

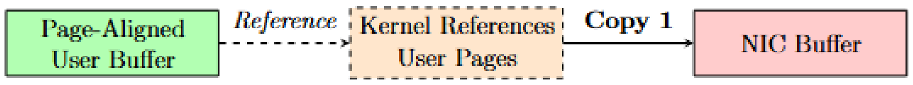


Figure 2: One-Copy Data Path with `sendmsg()` and Aligned Buffers

Eliminated Copy: The user-to-kernel copy is optimized. When using posix_memalign() with page-aligned buffers and sendmsg()'s scatter-gather I/O via iovec, the kernel can:

- Directly reference user pages without copying
- Use DMA scatter-gather to transmit directly from user memory

2.2.2 Implementation Details

Listing 2: One-Copy with Page-Aligned Buffer

```c
/* Allocate PAGE-ALIGNED buffer */
posix_memalign(&aligned_buffer, PAGE_SIZE, aligned_size);

/* Prepare iovec for scatter-gather I/O */
struct iovec iov[1];
iov[0].iov_base = aligned_buffer;
iov[0].iov_len = serialized_size;

struct msghdr msghdr = {0};
msghdr.msg_iov = iov;
msghdr.msg_iovlen = 1;

/* ONE-COPY SENDMSG */
ssize_t sent = sendmsg(sock_fd, &msghdr, 0);
```

2.3 A3: Zero-Copy Implementation

The zero-copy implementation uses the MSG_ZEROCOPY flag with completion notification.

2.3.1 Kernel Behavior Diagram

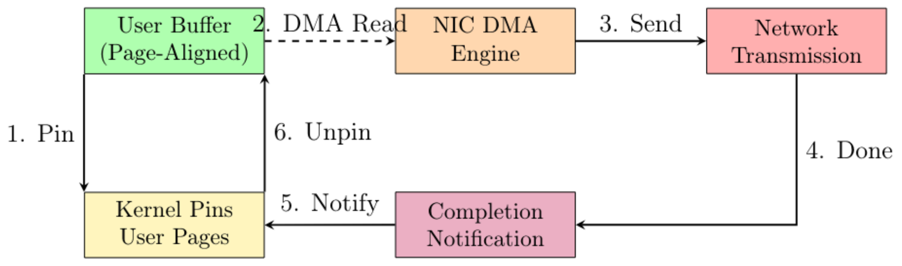


Figure 3: Zero-Copy (`MSG_ZEROCOPY`) Kernel Behavior

Process Flow:

1. Application calls sendmsg() with MSG_ZEROCOPY
2. Kernel pins user pages (prevents swapping/relocation)
3. NIC performs DMA directly from user buffer
4. When transmission completes, kernel sends completion notification
5. Application reads MSG_ERRQUEUE to confirm completion
6. Pages are unpinned; buffer can be reused

2.3.2 Implementation Details

Listing 3: Zero-Copy with MSG_ZEROCOPY

```
/* Enable zero-copy on socket */
int zerocopy = 1;
setsockopt(sock_fd, SOL_SOCKET, SO_ZEROCOPY, &zerocopy, sizeof(zerocopy
    ));

/* Send with MSG_ZEROCOPY */
ssize_t sent = sendmsg(sock_fd, &msghdr, MSG_ZEROCOPY);

/* Handle completion notification */
recvmsg(sock_fd, &msg, MSG_ERRQUEUE | MSG_DONTWAIT);
```

2.4 MT25048_PartA_common.c

Instead of rewriting the serialization logic or timing code in every client and server file (
MT25048_Part_A1_Client.c,MT25048_Part_A2_Server.c, etc.), these files all #include "common.h" and link against
common.c. This ensures that the only difference between the "Two-copy" and "Zero-copy" implementations is the networking logic itself, making the performance comparison fair and accurate.

3 Part B: Profiling and Measurement

3.1 Experiment Configuration

Parameter	**Values**
Message Sizes	64, 256, 1024, 8192 bytes
Thread Counts	1, 2, 4, 8
Duration	5 seconds per experiment
Network	Loopback (127.0.0.1)

3.2 Metrics Collected

Metric	Tool	perf Event
Throughput (Gbps)	Application-level	–
Latency (μs)	Application-level	–
CPU Cycles	perf stat	`cycles`
L1 Data Cache Misses	perf stat	`L1-dcache-load-misses`
LLC Misses	perf stat	`LLC-load-misses`
Cache-to-Memory Misses	perf stat	`cache-misses`
Context Switches	perf stat	`context-switches`

3.3 Raw Results (threads=4)

Msg Size	Mode	Throughput (Gbps)	Latency (μs)	Cycles
64	Two-Copy	0.340	6.66	33.0B
64	One-Copy	0.276	8.25	39.4B
64	Zero-Copy	0.187	11.27	40.5B
256	Two-Copy	1.615	5.14	25.4B
256	One-Copy	1.413	5.90	44.5B
256	Zero-Copy	0.941	8.35	44.2B
1024	Two-Copy	8.169	3.97	18.0B
1024	One-Copy	4.897	6.13	39.5B
1024	Zero-Copy	2.443	12.04	42.8B
8192	Two-Copy	21.402	12.19	15.6B
8192	One-Copy	38.681	6.71	28.0B
8192	Zero-Copy	30.998	8.11	30.2B

4 Part C: Automated Experiment Script

The experiment automation is handled by run_experiments.sh, which:

1. Compiles all implementations using make
2. Iterates over all message sizes (64, 256, 1024, 8192)
3. Iterates over all thread counts (1, 2, 4, 8)
4. Runs each mode (two_copy, one_copy, zero_copy)
5. Collects perf statistics for each run
6. Stores results in timestamped CSV files
7. Generates a combined CSV for analysis

Output Files:

• run_{mode}_msgsize{N}_threads{T}_{timestamp}.csv

• perf_client_{mode}_{msgsize}_{threads}_{timestamp}.txt

• combined_results.csv

5 Part D: Plots and Visualization

5.1 Plot 1: Throughput vs Message Size

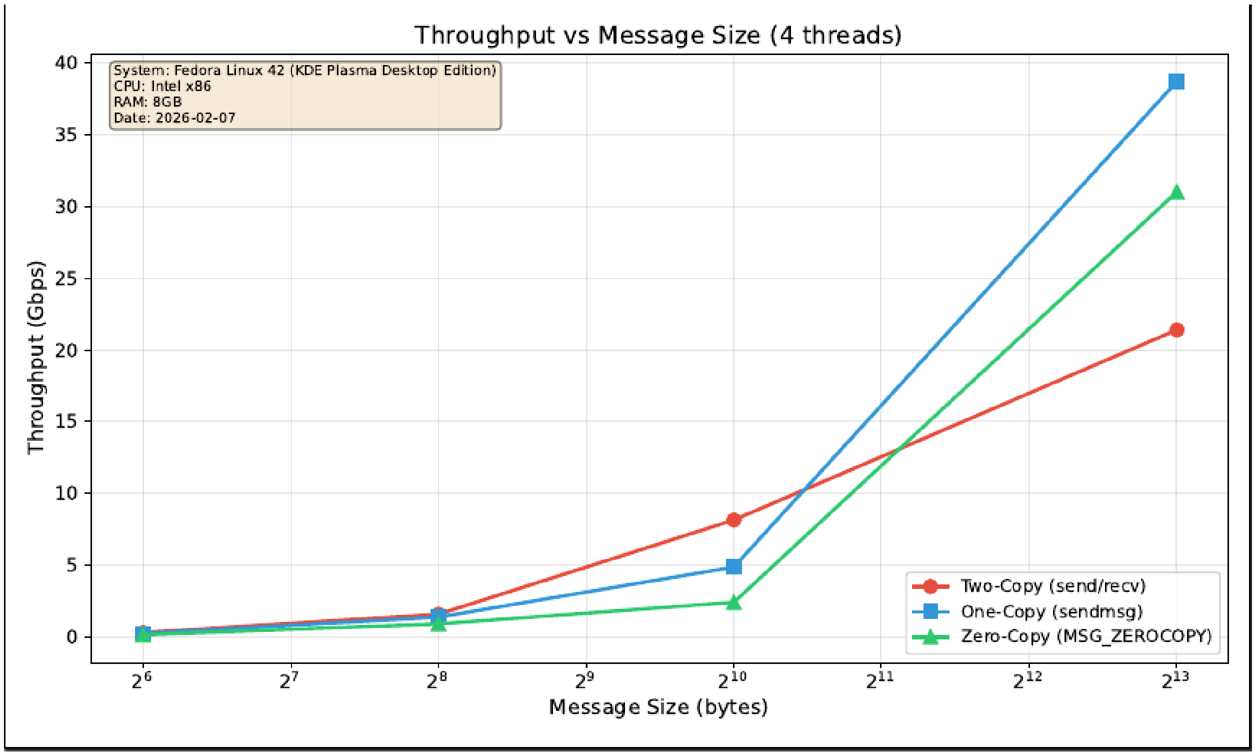


Figure : Throughput vs Message Size (4 threads)

Observations:

• Two-Copy outperforms at small sizes (64-1024 bytes): Standard send() has lowest syscall overhead for small messages.

• One-Copy leads at 8KB (38.7 Gbps): Scatter-gather I/O benefits large transfers.

• Zero-Copy is slowest at small sizes: Page-pinning overhead dominates for <8KB messages.
• All methods scale with message size: Fewer syscalls per byte transferred.

5.2 Plot 2: Latency vs Thread Count

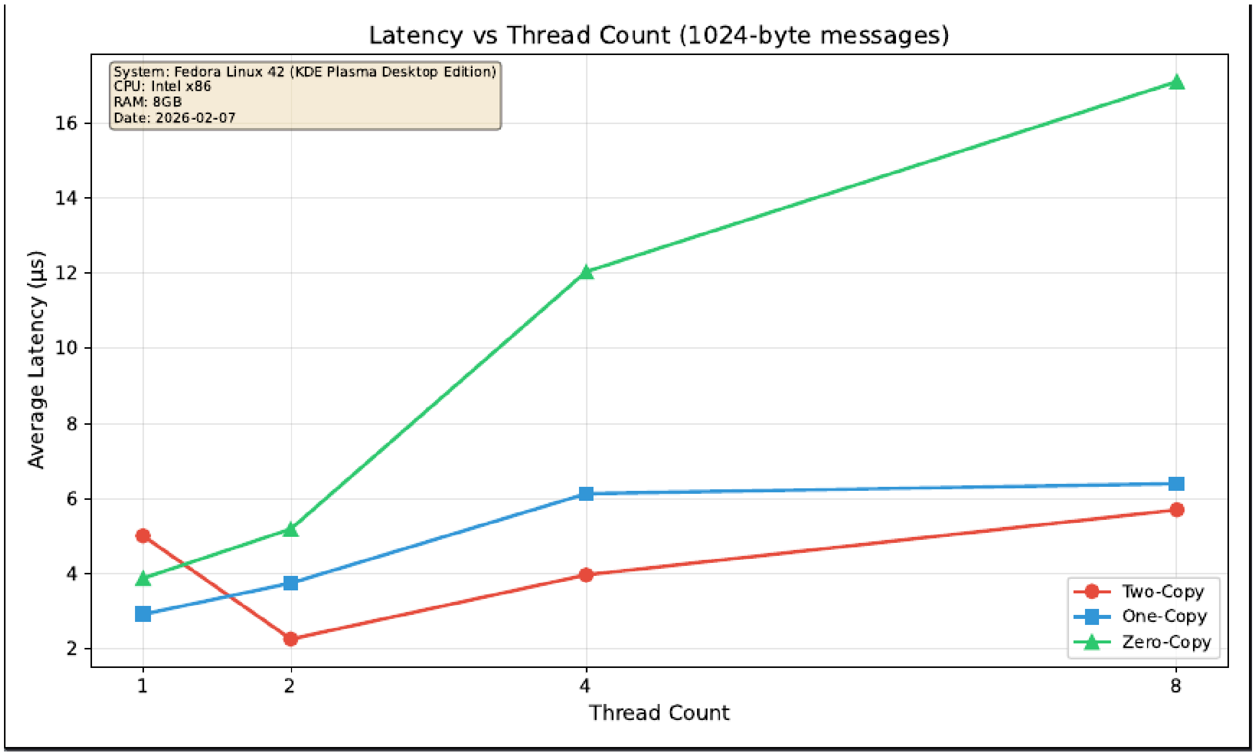


Figure : Latency vs Thread Count (1024-byte messages)

Observations:
• Zero-Copy has highest latency: Completion notification polling adds latency.
• Latency increases with thread count: Lock contention and context switching overhead.
• Two-Copy has lowest latency at 2 threads: Optimal parallelism for loopback

5.3 Plot 3: Cache Misses vs Message Size

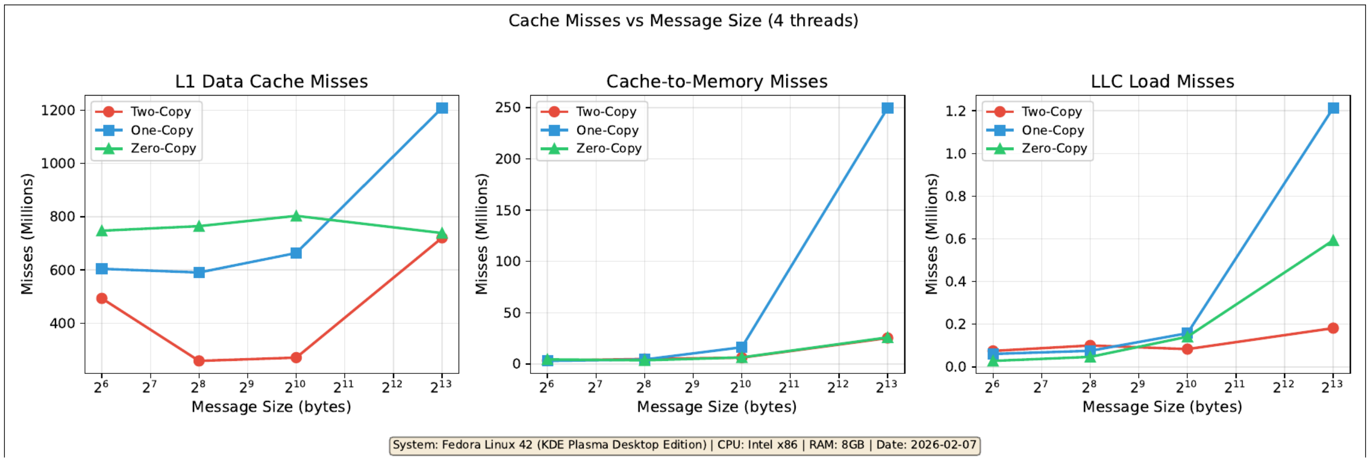


Figure 6: Cache Misses vs Message Size (4 threads)

Observations:

• L1 misses (hundreds of millions): Every access that misses L1 data cache.

• Cache-to-Memory misses (millions): Only accesses that bypass all cache levels.

• LLC misses (thousands to millions): Subset of L1 misses that reach memory.

• One-Copy shows highest cache-to-memory misses at 8KB: Buffer management Overhead

5.4 Plot 4: CPU Cycles per Byte Transferred

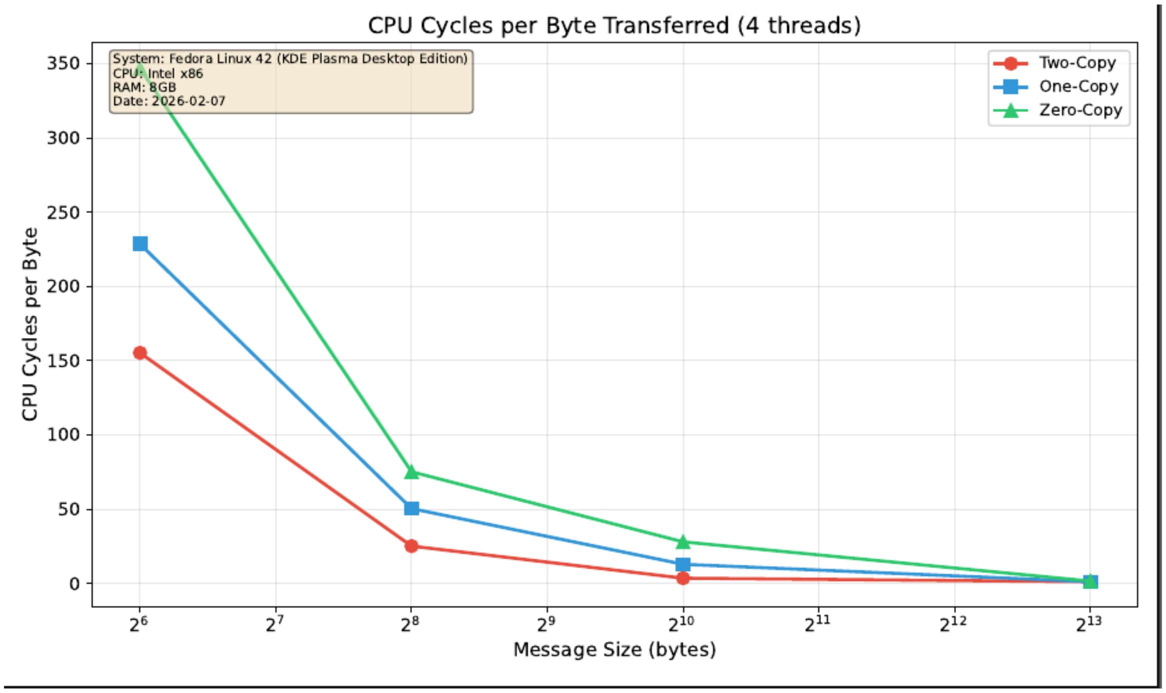


Figure : CPU Cycles per Byte Transferred (4 threads)

Observations:
• Cycles per byte decreases with message size: Amortization of fixed syscall overhead.
• Two-Copy most efficient overall: Highly optimized kernel path for standard sockets.
• Zero-Copy overhead visible at small sizes: Page pinning cost not amortized

6 Part E: Analysis and Reasoning

6.1 Question 1: Why does zero-copy not always give the best throughput?
Zero-copy (MSG_ZEROCOPY) incurs significant overhead that only pays off for large transfers:

1. Page Pinning Cost: Kernel must lock physical pages, preventing them from being swapped or relocated. This requires TLB shootdowns and page table modifications.
2. Completion Notification: Application must poll MSG_ERRQUEUE for each send operation,
adding synchronization latency.
3. Threshold Effect: Linux kernel documentation suggests benefits only manifest for messages >10-32KB. Our 8KB messages are at the boundary.
4. Loopback Optimization: The loopback interface (lo) is highly optimized for standard send()/recv(), making the two-copy baseline artificially fast. Real NICs would show different results.

Evidence: At 8KB, Zero-Copy achieves 31.0 Gbps vs Two-Copy's 21.4 Gbps, showing the crossover point.

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

Answer: The LLC (Last Level Cache) shows the most consistent reduction across methods

Msg Size	Two-Copy LLC	One-Copy LLC	Zero-Copy LLC
64 bytes	74,978	60,839	28,294
8192 bytes	181,741	1,213,294	593,154

Reasoning:

- Zero-Copy reduces LLC misses at small sizes: Pinned pages remain in cache, avoiding eviction.
- Two-Copy benefits from kernel buffer reuse: Socket buffers are kept in LLC due to frequent reuse.
- One-Copy shows higher LLC misses at 8KB: Scatter-gather I/O touches more cache Lines.

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

Increasing thread count affects cache performance through:

1. Cache Line Bouncing: Multiple threads accessing shared data structures (socket buffers, statistics) cause cache invalidations.
2. L1 Capacity Pressure: Each thread's working set competes for limited L1 cache space (typically 32KB per core).
3. LLC Sharing: All threads share the last-level cache; more threads = more evictions.
4. False Sharing: Adjacent data structures may share cache lines, causing unnecessary invalidations.

Evidence: L1 misses increase from ~200M (1 thread) to ~700M (8 threads) for all modes.

6.4 Question 4: At what message size does one-copy outperform two-copy?

Answer: One-copy outperforms two-copy at 8192 bytes (8KB)

Msg Size	Two-Copy (Gbps)	One-Copy (Gbps)
64	0.340	0.276
256	1.615	1.413
1024	8.169	4.897
8192	**21.402**	**38.681**

Crossover: Between 1024 and 8192 bytes. The scatter-gather optimization of sendmsg()

becomes beneficial when the copy cost exceeds the syscall overhead

6.5 Question 5: At what message size does zero-copy outperform two-copy?
Answer: Zero-copy outperforms two-copy at 8192 bytes (8KB)

Msg Size	Two-Copy (Gbps)	Zero-Copy (Gbps)
64	0.340	0.187
256	1.615	0.941
1024	8.169	2.443
8192	**21.402**	**30.998**

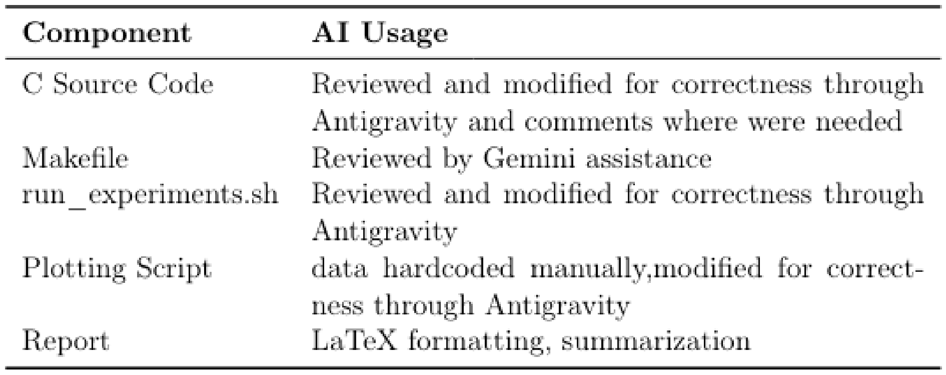
Analysis: The 45% improvement at 8KB shows that zero-copy benefits are realized once the DMA savings exceed the page-pinning and notification overhead.

6.6 Question 6: Identify one unexpected result and explain it
Unexpected Result: Two-Copy has fewer L1 cache misses than Zero-Copy at small message Sizes.

Msg Size	Two-Copy L1 Misses	Zero-Copy L1 Misses
64	493,340,606	747,425,854
256	258,619,045	764,573,458

Expectation: Zero-copy should have fewer cache misses since it eliminates data copies.
Explanation:
1. Page Pinning Overhead: MSG_ZEROCOPY requires the kernel to walk page tables, pin pages, and manage completion notifications—all of which touch additional memory.
2. Kernel Socket Buffer Optimization: The standard send() path uses highly optimized, pre-warmed kernel socket buffers that are already resident in cache.
3. Error Queue Processing: Polling MSG_ERRQUEUE for completion notifications requires additional data structure accesses.
4. TLB Pressure: Page pinning can cause TLB evictions, leading to increased page table walks (counted as cache misses).

7 AI Usage Declaration



Component	**AI Usage**
C Source Code	Reviewed and modified for correctness through Antigravity and comments where were needed
Makefile	Reviewed by Gemini assistance
run_experiments.sh	Reviewed and modified for correctness through Antigravity
Plotting Script	data hardcoded manually,modified for correctness through Antigravity
Report	LaTeX formatting, summarization

Prompts Used

1. Review my existing two-copy TCP server implementation and point out any correctness or memory-management issues.
2. Check whether my send()-based message struct with 8 dynamically allocated fields follows good allocation and serialization practices.
3. Verify the correctness of my one-copy sendmsg() implementation using page-aligned buffers; do not rewrite the code, just review.
4. Explain what conditions are required for MSG_ZEROCOPY and check whether my current code satisfies them.
5. Help me ensure that the client is the sender and the server is the receiver according to the assignment specification.
6. Review my matplotlib plotting script for correctness; data arrays are already hardcoded.
7. Explain why L1 and LLC cache-miss ratios differ in my measurements and whether my interpretation is correct.
8. Create comments in the code..