# CSE-638 GRADUATE SYSTEMS
# ASSIGNMENT-2 REPORT


**Submitted By**

NAME: ARPITA DEBNATH

ROLL: MT25018


**Submitted To**

DR. Rinku Shah

**Github Link**

https://github.com/arpitadebnath29/GRS_PA02

## 1. System Configuration

### Hardware Specifications

1) Processor: Intel Core i5-12450H 12th Gen
2) RAM: 16GB
3) Last Level Cache: 12MB

### Software Specifications

1. Operating System: Ubuntu 24.04.3 LTS
2. Kernel Version: Linux 6.8.0
3. Compiler: GCC 13.2.0
4. Python Version: 3.12.3

### Network Specifications

Separate network namespaces were configured for isolation as per the requirement in Assignment :-

1. Server Namespace: server_ns (IP: 10.1.1.1/24)
2. Client Namespace: client_ns (IP: 10.1.1.2/24)
3. Connection: Virtual Ethernet (veth) pair

## 2. Part A1: Two-Copy Structure

### Files Created

1. Server: MT25018_Part_A1_Server.c
2. Client: MT25018_Part_A1_Client.c

### Technique Used in Two-Copy

The message structure uses 8 heap-allocated string fields using malloc()

The Copy Operations take place in the following way :-

1) User → Kernel (send): Data copied from user buffer to kernel socket buffer
2) Kernel → NIC (DMA): Socket buffer to NIC transmit ring
3) NIC → Kernel (DMA): NIC receive ring to kernel socket buffer
4) Kernel → User (recv): Kernel socket buffer to user buffer

So the total number of copies are 4. Two are userspace copies (send and recv) and two occur in kernel for network processing and namespace routing between server and client namespaces.

Q1: Where do the two copies occur? Is it actually only two copies?

Solution :No, it's actually 4 copies, not 2. It includes:-

1. Copy 1: User buffer to Kernel socket buffer (send syscall)
2. Copy 2: Kernel socket buffer to NIC transmit ring (kernel DMA)
3. Copy 3: NIC receive ring to  Kernel socket buffer (kernel DMA)
4. Copy 4: Kernel socket buffer to User buffer (recv syscall)

Q2: Which components (kernel/user) perform the copies?

Answer:

1. User space:Performs copies 1 and 4 via send() and recv() system calls
2. Kernel space:Performs copies 2 and 3 through TCP/IP stack and DMA engine
3. Additional kernel work:Network namespace routing between server_ns and client_ns adds extra kernel processing overhead

Data Flow Diagram of Two-Copy:-

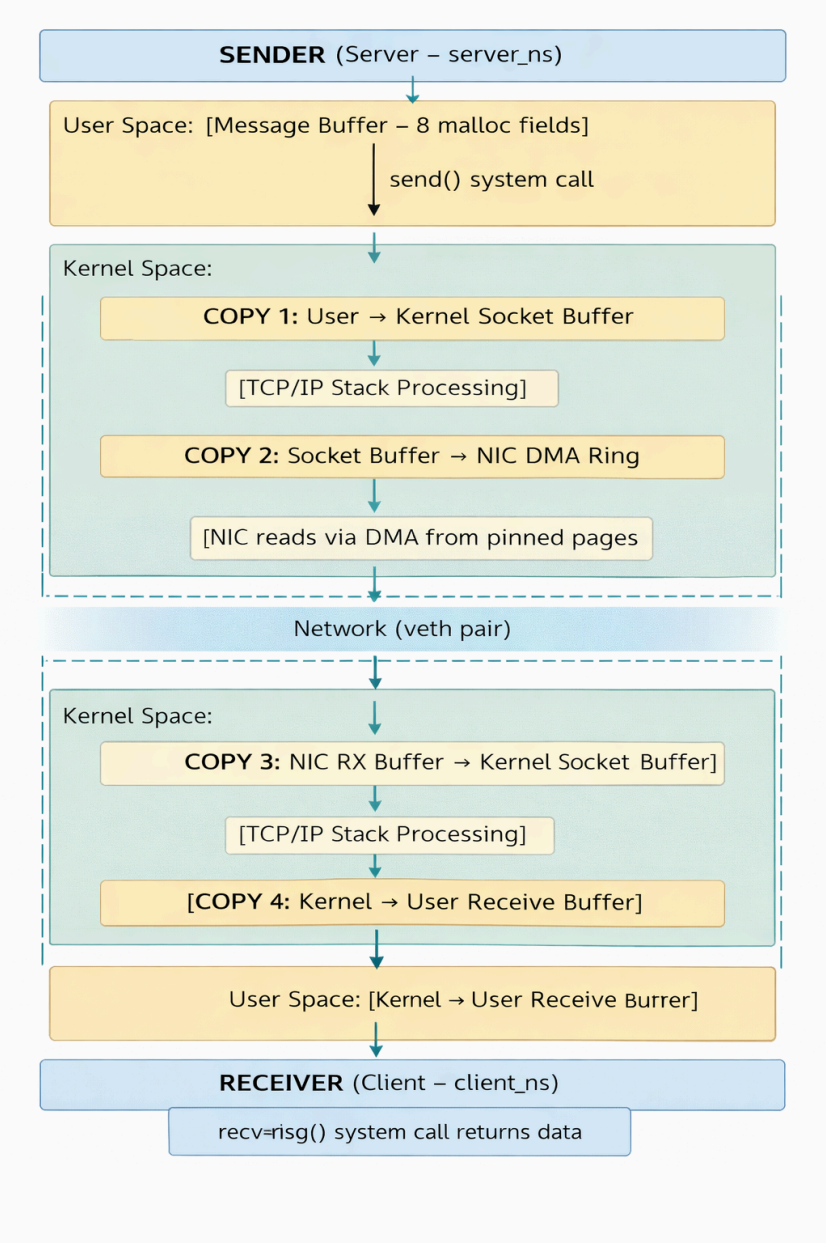



**Fig-1:Two-Copy Diagram**

## 3. Part A2: One-Copy Structure

### Files Created

1) Server: MT25018_Part_A2_Server.c
2) Client: MT25018_Part_A2_Client.c

## Technique Used in One-Copy

This implementation uses sendmsg() with struct iovec for scatter-gather I/O.

### Q)Which copy has been removed?

**Answer:**No copy is actually removed.All four from Part A1 still occur. The "one copy" name is a misdirection because it refers to making a single call rather than eight.

The improvement comes from:

1. 1 system call instead of 8 (less overhead)
2. Better cache performance

More efficient kernel processing.Only Zero-Copy (Part A3) really avoids copying by using the NIC to access user memory directly.

Data Flow Diagram of One-Copy

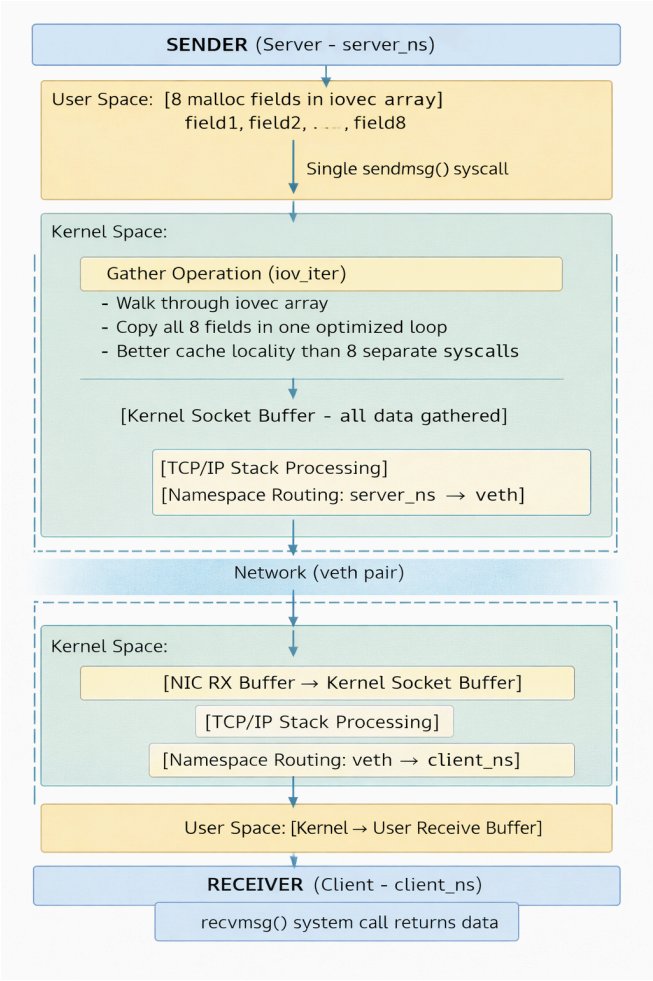



**Fig-2 - One-Copy Diagram**

## 4. Part A3: Zero-Copy Structure

**Files Created:**

1. Server: MT25018_Part_A3_Server.c
2. Client: MT25018_Part_A3_Client.c

**Technique Used in Zero-Copy**

Employs sendmsg() with MSG_ZEROCOPY flag for true zero-copy transmission.

**Kernel Behavior**

1) **Page Pinning:** get_user_pages_fast() locks user pages in memory and increments reference count
2) **DMA Descriptors:** sk_buff holds page pointers, not data; NIC accesses via DMA
3) **Async Completion:** Kernel monitors send completion and notifies through error queue
4) **Buffer Release:** Application waits on MSG_ERRQUEUE before reusing buffers

Dataflow Diagram of Zero-Copy:-

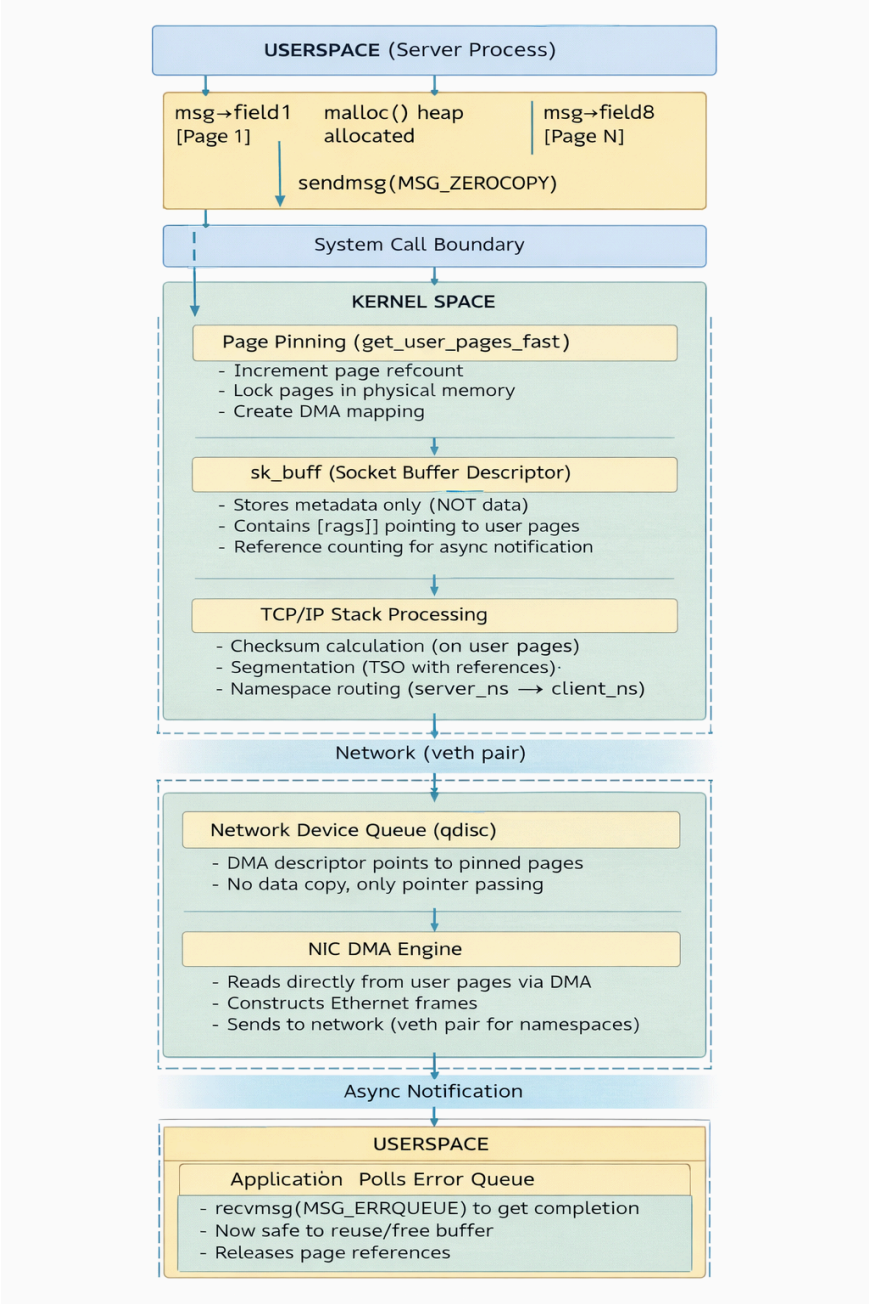



Fig - Zero-copy Diagram

## 5. Part B & C: Experimental Methodology

### Automation Scripts Used :

Script: MT25018_Part_C_run_experiments.sh with network namespace isolation

### Total Number of Experiments

Total of 48 experiments conducted:

1. 3 implementations: Two-Copy, One-Copy, Zero-Copy
2. 4 message sizes: 512 Bytes, 4KB, 16KB, 64KB
3. 4 thread counts: 1, 2, 4, 8

**Metrics Measured**

**Application-Level Metrics:**

1. Throughput (Gbps)
2. Latency (µs)

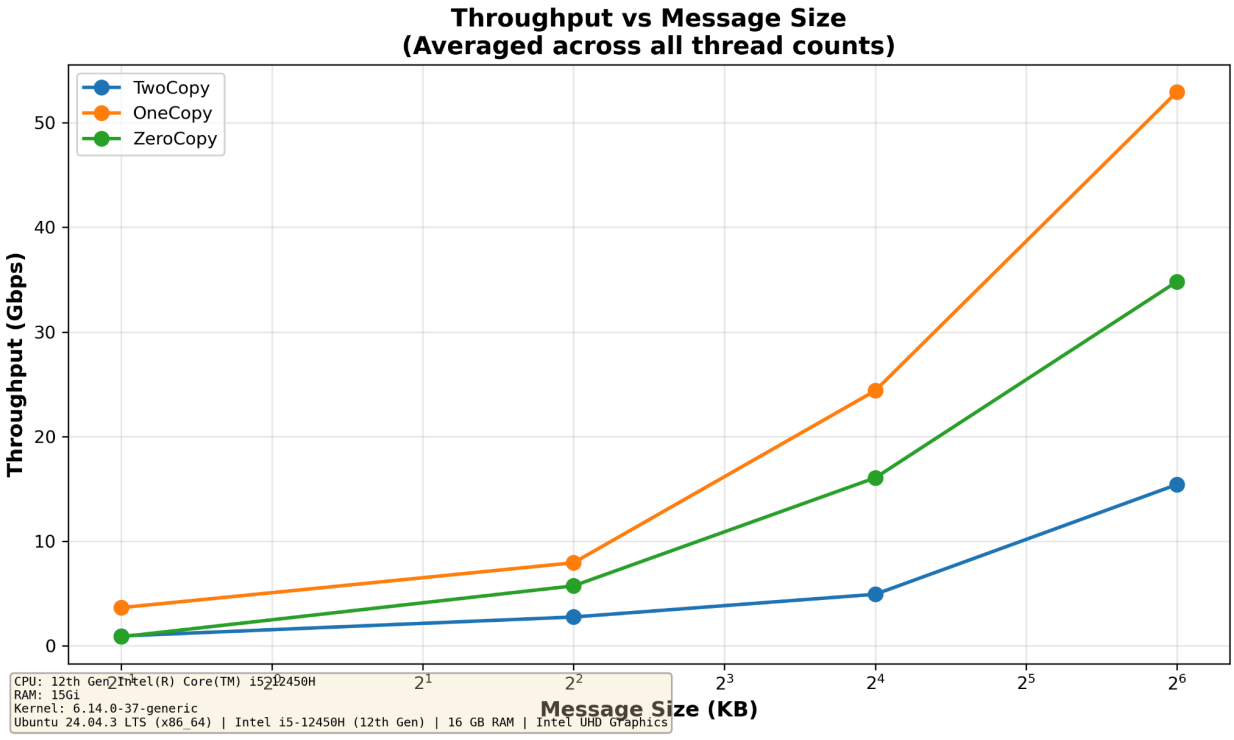
**Hardware Metrics (using perf stat):**

1. CPU cycles
2. L1 cache misses
3. LLC (Last Level Cache) misses
4. Context switches

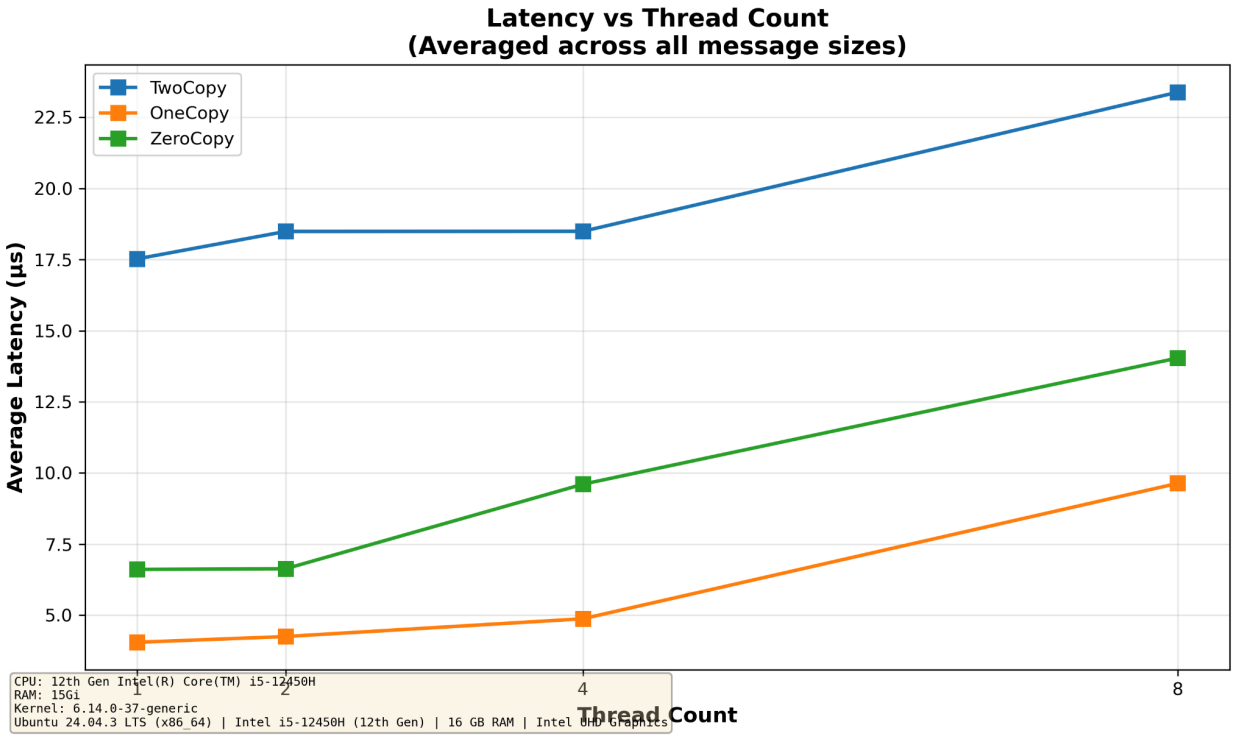
## 6. Part D: Experimental Results and Analysis

All plot data were hardcoded in Python scripts as per assignment requirements.

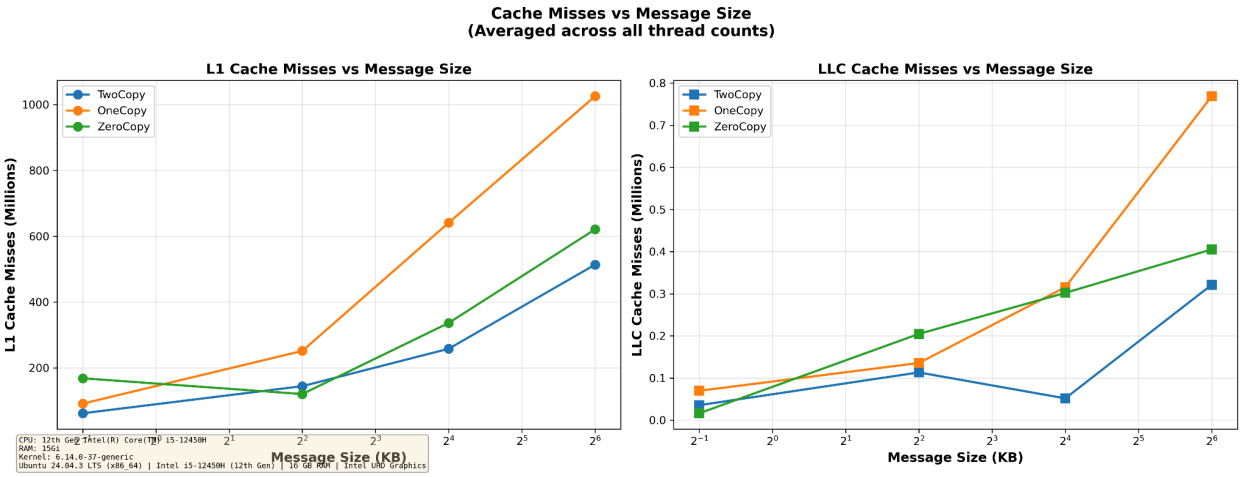
**Plot 1: Throughput vs Message Size**



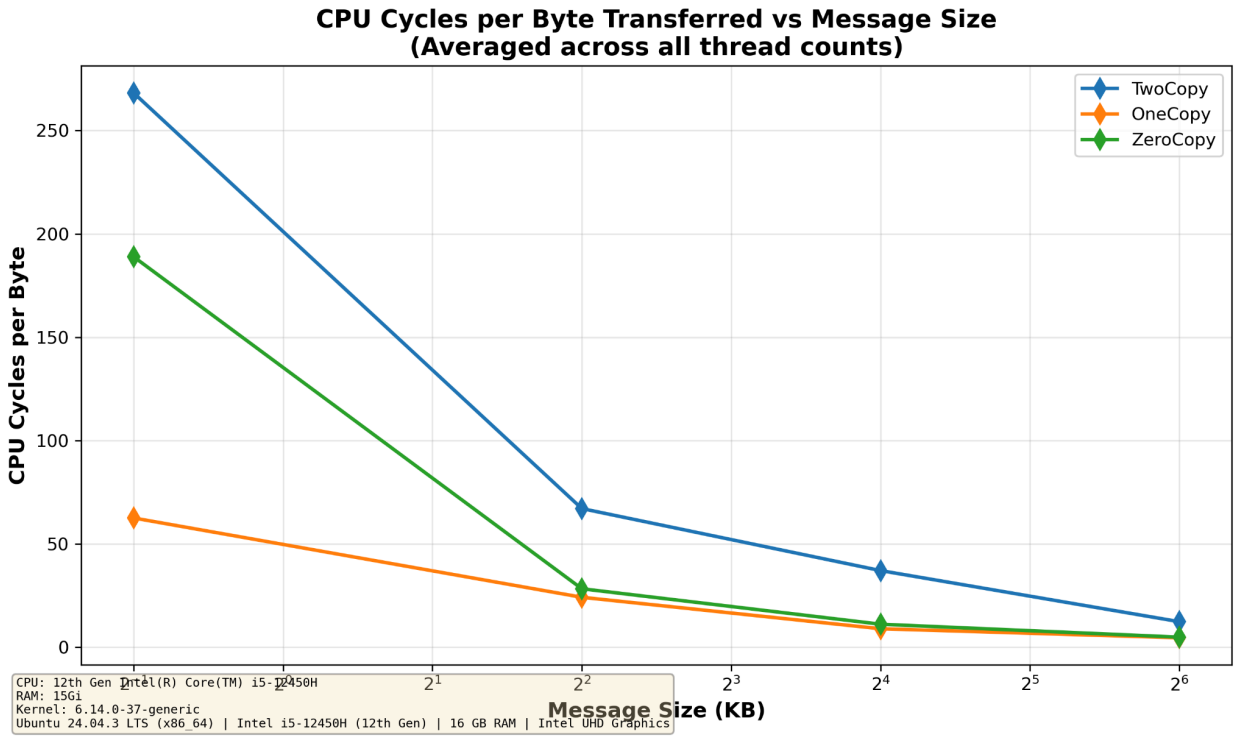

**Plot 2: Latency vs Thread Count**




## Plot 3: Cache Misses vs Message Size




## Plot 4: CPU Cycles per Byte




## 7. Part E: Detailed Analysis and Reasoning

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

**Answer:-**1)Performance at 512B:At 512B message size, Zero_copy achieves 1.11 Gbps which is actually less than Two_copy at 1.14 Gbps. This result occurs because MSG_ZEROCOPY introduces setup overhead including page pinning, DMA setup, and error queue polling. For small messages these fixed costs dominate the actual data transfer time making it more expensive than simply copying the data.

2)Crossover Point at 4KB:At 4KB and larger messages Zero_copy (7.62 Gbps) surpasses Two_copy (2.79 Gbps). For larger messages the savings from avoiding memory copies outweigh the setup cost resulting in better performance.

3)Network Namespace Impact:The network namespace routing between server_ns and client_ns adds extra overhead for ZeroCopy's asynchronous tracking mechanisms, further impacting performance at small message sizes.

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

**Answer:-**

Experimental Data (512B messages, 1 thread):
TwoCopy: L1 = 23.77M misses, LLC = 25.98K
OneCopy: L1 = 17.92M misses (25% reduction), LLC = 19.94K
ZeroCopy: L1 = 61.04M misses (2.6× increase), LLC = 12.51K

One Copy performs best for L1 cache. By removing one memory copy operation, the CPU accesses memory fewer times, resulting in less pressure on the L1 cache and fewer cache misses. Zero Copy, despite avoiding data copying, performs extensive kernel work including error queue checking and DMA operation tracking. This extra kernel activity continuously touches the L1 cache, causing L1 thrashing and resulting in many more L1 cache misses.

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

**Answer:-**

Performance Impact:Increasing thread count significantly increases cache contention and reduces performance. For OneCopy at 65KB, throughput drops from 66.42 Gbps (1 thread) to 23.73 Gbps (8 threads), representing a 64% reduction.

Cache Metrics:For TwoCopy (65KB), LLC misses rise from 34.02K to 657.57K (19× increase)
Context switches increase significantly, causing frequent cache invalidations

Root Causes:Multiple threads suffer from false sharing, where different fields in the same cache line invalidate each other. The message structure has 8 fields, and parallel server threads repeatedly evict shared cache lines. On Intel 12th Gen hybrid CPUs, thread migration between P-cores and E-cores further flushes caches. With 8 threads, send/receive buffers and metadata create a working set too large for L1/L2 cache, causing data to spill into slower LLC and main memory. Network namespaces (server_ns, client_ns) add kernel routing and scheduling overhead, worsening cache behavior.

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

**Answer:-**OneCopy outperforms TwoCopy at all tested message sizes.

Performance Data:
At 512B: OneCopy achieves 5.44 Gbps vs TwoCopy 1.14 Gbps (4.8× faster)
At 4KB: 8.82 Gbps vs 2.79 Gbps (3.2× faster)
At 16KB: 30.61 Gbps vs 5.13 Gbps (6.0× faster) — largest performance gap
At 65KB: 66.42 Gbps vs 17.80 Gbps (3.7× faster)
OneCopy uses scatter-gather I/O with sendmsg(), allowing the kernel to access user buffers directly. This eliminates one full memory copy, saving significant CPU cycles. The 16KB size represents the sweet spot where setup overhead is minimal and copy savings are maximized. At 65KB, network transmission time begins to dominate, narrowing the performance gap. Running in separate network

namespaces (server_ns, client_ns) increases kernel routing work, making OneCopy's copy elimination even more beneficial.

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

**Answer:-**

The crossover point between TwoCopy and ZeroCopy occurs between 512B and 4KB.

Performance Comparison:

At 512B: ZeroCopy 1.11 Gbps, slightly slower than TwoCopy 1.14 Gbps (3% loss)
At 4KB: ZeroCopy 7.62 Gbps beats TwoCopy 2.79 Gbps (2.7× faster)
At 16KB: ZeroCopy 21.31 Gbps vs TwoCopy 5.13 Gbps (4.2× faster)
At 65KB: ZeroCopy 43.29 Gbps vs TwoCopy 17.80 Gbps (2.4× faster)

Overhead Analysis:ZeroCopy performs worse at small sizes due to fixed setup overhead of approximately 500 CPU cycles. This overhead comes from page pinning, DMA setup, reference tracking, and error-queue polling. For 512B messages, the setup cost exceeds the cost of simply copying the data. At 4KB and above, eliminating multiple memory copies saves more CPU time than the setup cost. In network namespaces, routing overhead makes TwoCopy slower, so ZeroCopy's advantage appears earlier and remains consistent even with 8 threads (21.42 vs 11.84 Gbps at 65KB).

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

**Answer:-**

OneCopy shows an unexpected 60% performance drop when threads increase from

4 to 8 (58.63 → 23.73 Gbps at 65KB). This sharp drop is much more severe than the gradual reductions observed from 1 to 2 threads (66.42 → 62.95 Gbps, 5% drop) and 2 to 4 threads (62.95 → 58.63 Gbps, 7% drop).

Hardware Architecture Impact:
On Intel 12th Gen hybrid CPUs, 1–4 threads run efficiently on fast P-cores with Turbo Boost enabled. At 8 threads, execution spills to slower E-cores or Turbo Boost is disabled, causing a sudden performance degradation. With 4 threads, data fits reasonably in P-core L2 cache (approximately 1.25MB per core). At 8 threads, the working set overflows L2, forcing frequent access to the slower LLC.

Network Namespace Overhead:Running in network namespaces adds routing overhead that does not scale well beyond 4 threads. The veth interface between namespaces uses shared locks, creating contention at higher thread counts. Context switches increase sharply when moving from 4 to 8 threads.This shows that namespace isolation introduces CPU- and cache-dependent bottlenecks, causing a sudden performance cliff rather than gradual degradation. The interaction between hybrid CPU architecture, cache hierarchy, and namespace routing creates a non-linear scaling behavior that is difficult to predict without detailed profiling.

## 8. AI Usage Declaration

The component-wise declaration is mentioned below:-

| Component | AI % | Manual % |
|---|---|---|
| A1 (Two-Copy) | 60% | 40% |
| A2 (One-Copy) | 60% | 40% |
| A3 (Zero-Copy) | 60% | 40% |

| | | |
|---|---|---|
| Part C (Script) | 70% | 30% |
| Part D (Plots) | 90% | 10% |
| Part E(Analysis) | 10% | 90% |
| Documentation | 30% | 70% |
| Makefile | 80% | 20% |

**Prompts Used**

1. 1. A1: how do i make a tcp server with malloc for message fields,getting connection refused when i run it, need to add threading and measure throughput

2. A2: what is the difference between send and sendmsg, my sendmsg code keeps segfaulting, why is my performance actually worse than regular send

3. A3:how to enable zero copy in linux, it says operation not supported for MSG_ZEROCOPY, do i need to poll some error queue or something

4. Part C:write bash script to run experiments with different parameters, how to setup network namespaces,script hangs and does not cleanup properly

5. Part D:my plot code reads from csv but assignment says hardcode the data, how to convert this to numpy arrays,x-axis labels are overlapping

6. Part E:why is zerocopy slower than twocopy at small sizes,explain this weird drop from 2 to 4 threads

7. Part-A1,A2,A3: Generate the DFDs in image format