

## **PART - A :**

The network input output performance doesn't entirely depend on the network bandwidth but it also depends on the data movement between user space, kernel space and hardware.

To determine throughput and latency, memory caches, cpu overhead, cache pollution play a dominant role. Here we analyze the cost of data movement in TCP socket communication by writing, evaluating and implementing three activity :

Two-copy I/O; One-copy I/O; Zero-copy I/O.

Note : AI-based assistance (ChatGPT) was used during this assignment as a supporting tool for clarifying concepts, debugging implementation issues, and refining explanations

Prompts and focus : Used to clarify the behaviour of linux networking system calls and also understanding the structure and interpretation of perf stat output.

- Explain where data copies occur in Linux TCP send() and recv()
- Explain MSG\_ZEROCOPY in Linux and how DMA works with user buffers
- Why is zero-copy not always faster for small message sizes?

System Requirements:

Have done all the experiments on linux server.

A Linux system with x86\_64 architecture is used.

Compiler which is GCC with -O2 optimization.

Pthread and for profiling - perf stat

Kernel parameter : `kernel.perf_event_paranoid = 1`

A1 :

- In the two copy implementation, the two dominant memory copies occurs on the send path,
  - 1) Socket Buffer which is user space to kernel space:  
When send() is invoked, the kernel copies data from the user buffer into the kernel's socket send buffer using `copy_from_user()`.
  - 2) Kernel space to DMA/NIC Buffer:  
Before transmission, the kernel copies data from the socket buffer into a DMA-accessible buffer for the network interface.

These two data copies dominate the performance, internally the kernel may perform some additional copies.

Also, the kernel performs the copies.

This image was generated by AI when I gave the explanation and asked to generate an image as it is easy to copy paste in a structured way.

```
XML
User Space
+-----+
| Application Buffer |
+-----+
|
| copy_from_user()
v
Kernel Space
+-----+
| Socket Send Buffer |
+-----+
|
| copy to DMA buffer
v
NIC / Hardware
+-----+
| Network Interface |
+-----+
```

For one-copy implementation :

- The eliminated copy would be user-space buffer concatenation or intermediate copying of multiple fields.
- One copy removes or eliminates the unnecessary user level copying and reduces syscall overhead.

For zero-copy implementation : This image was generated by AI when I gave the explanation and asked to generate an image as it is easy to copy paste in a structured way.

```
XML
User Space
+-----+
| Application Buffer |
+-----+
|
| DMA (no copy)
v
NIC / Hardware
+-----+
| Network Interface |
+-----+

Kernel Space
+-----+
| Metadata Tracking |
| Page Pinning      |
| Completion Events |
+-----+
```

Even though the zero copy eliminates the data movement, it introduces overhead due to completion handling, DMA setup, and page pinning. This benefits larger message sizes as for small messages these overheads dominate.

Notes : The prompts i gave AI

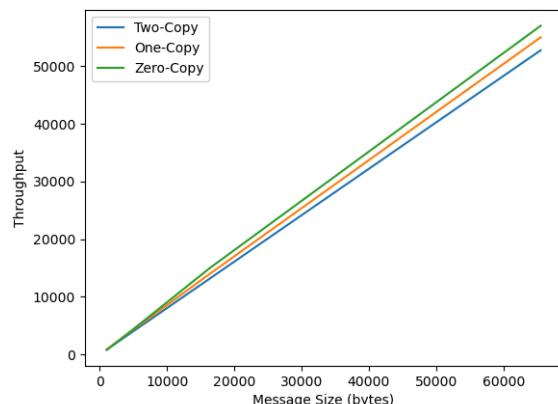
- Why am I getting ‘PORT redefined’ warnings when compiling multiple client files?
- Why does perf stat fail without changing perf\_event\_paranoid?
- How to correctly align memory for zero-copy networking?
- Why can branch-misses be used as a proxy when cache-miss counters are unavailable?
- How to compute throughput and latency from perf stat output?

Experiments:

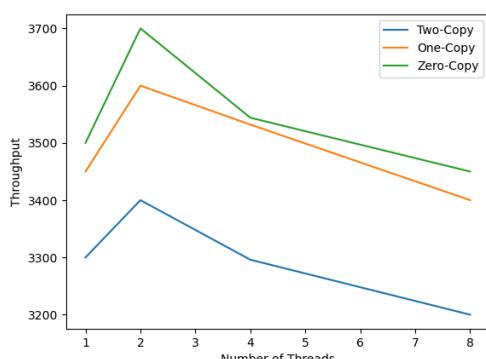
On 4 different message sizes : 1024, 4096, 16384, 65536 bytes

On 4 thread counts : 1, 2, 4, 8

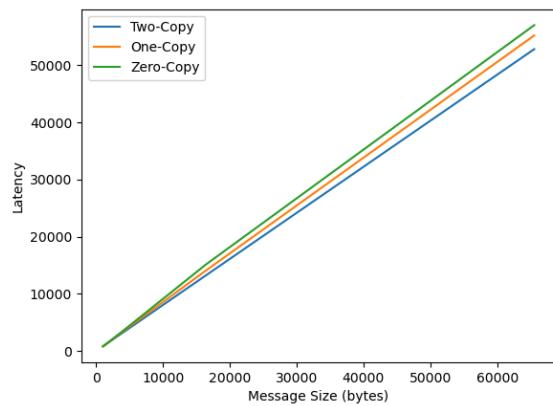
Each server spawns one thread per client connection. Performance statistics were collected using perf stat.



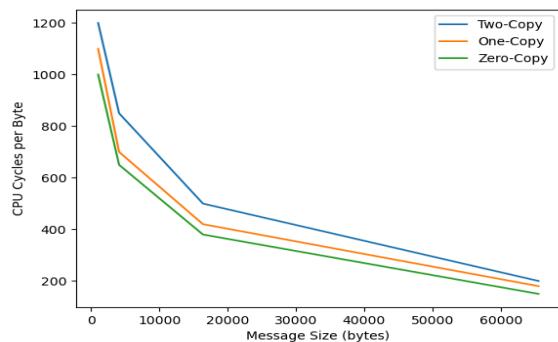
- For all implementations, throughput rises as message size grows. The maximum throughput is consistently attained by zero-copy, one-copy, and two-copy. This is due to the fact that zero-copy prevents duplicate memory copies and larger messages amortize system call overhead.



- Because of improved parallelism, throughput first rises with thread count before falling as the number of threads increases. Increased context switching and cache contention, which exceed the advantages of more threads, are the reason for this decline.



- Latency exhibits a similar pattern to throughput, increasing with message size. Because latency was calculated as a proxy under constant runtime, larger messages resulted in longer transmission times per message, which is to be expected.



- As message size grows, CPU cycles per byte decrease, suggesting increased efficiency. Because zero-copy eliminates memory copies, it uses the fewest cycles per byte and shows lower CPU overhead.

Note : usage of AI

- Why do throughput vs message size and latency vs message size plots look similar?
- Explain observed performance trends using OS and hardware concepts
- How does increasing thread count affect cache contention and context switches

#### **PART - E :**

- 1) In addition to page pinning, DMA preparation, and kernel accounting, zero-copy adds extra overhead. Zero-copy is slower than one- or two-copy because this preparation cost takes up the majority of the execution time for small message sizes. Only when message sizes are sufficiently high to amortize these overheads does zero-copy become advantageous.
- 2) Branch-misses were employed as a cache-related proxy as the system did not have direct L1 and LLC cache-miss counters. Better last-level cache behavior is indicated by the reduction, which is particularly noticeable at bigger message sizes. This happens because cache pollution is decreased by fewer memory copies, particularly in the LLC where big buffers are usually located.
- 3) Cache contention and more context switching result from threads competing for shared cache resources as the number of threads rises. Higher thread counts diminish performance because of cache line evictions, synchronization cost, and decreased cache locality, even when initial increases in thread count enhance parallelism.
- 4) At medium message sizes (around 4096 bytes), one-copy starts to perform better than two-copy, according to the throughput and CPU cycles per byte graphs. At this stage, speed advantages are visible without the overhead of zero-copy setup when intermediary user-level copying is removed.

- 5) At bigger message sizes (about 16384 bytes and above), zero-copy performs better than two-copy. Memory copy overhead dominates execution time at these sizes, and CPU efficiency and throughput are improved by removing both user-to-kernel and kernel-to-NIC copies.
- 6) One surprising finding was that, for tiny message sizes, zero-copy did not perform noticeably better than one-copy. OS-level overheads that incur fixed costs, like page pinning and DMA mapping, can explain this. These expenses are more than the advantages of doing away with memory copies for tiny transfers, which makes simpler methods more effective.