Task 5: Comparison of Thread vs. UDP Communication Overheads

1. Methodology

To evaluate the performance overhead of different communication mechanisms used in concurrent systems, a benchmark simulation was developed using Python 3.12. The simulation measured the latency of transmitting a 64-byte payload across three distinct communication models:

- Shared Memory (Threads): Intra-process communication using a shared variable protected by a threading Lock. This simulates the communication method used in Task 1 of the coursework.
- User Datagram Protocol (UDP): Connectionless inter-process communication using the socket library to send packets to the loopback address (127.0.0.1). This simulates the communication method used in Task 4.
- Web Sockets: Connection-oriented communication over TCP, simulating standard web-based data transfer.

The benchmark performed 10,000 operations for each method to calculate the average latency per operation, smoothing out minor system fluctuations.

2. Data

The following quantitative data was collected from the benchmark suite running on the local laboratory hardware:

Communication Method	Total Time (10,000 ops)	Average Latency per Op
Threads (Shared Memory)	0.1695 s	16.95 µs
UDP (Inter-Process)	0.3245 s	32.45 µs
Web Sockets	0.8350 s	83.50 µs

3. Data Analysis

The data reveals a clear hierarchy in communication performance overheads:

- Threads vs. UDP: UDP communication was found to be approximately 1.9x slower than Thread-based communication (32.45 / 16.95 $\approx$ 1.91).
    - *Reasoning:* Threads operate within the same virtual address space, meaning data transfer is essentially a memory write operation protected by a user-space mutex. UDP requires the Operating System kernel to intervene, copying data from user space to kernel space and processing it through the network stack.
- Threads vs. Web Sockets: Web Sockets were approximately 4.9x slower than Threads (83.50 / 16.95 $\approx$ 4.92).
    - *Reasoning:* Web Sockets rely on TCP, introducing overhead for packet acknowledgement and ordering. Additionally, the WebSocket protocol adds its own framing layer (headers and masking), which requires more CPU cycles to process than a raw UDP packet.

Note on Execution Environment (Python vs. C++)

It is critical to note that the relative performance gap differs significantly depending on the language used.

- C++ Performance: In a separate C++ test performed on the same hardware, Thread latency was measured at 0.0434 µs vs. UDP at 4.9936 µs. This represents a massive ~115x difference, because C++ compiles to machine code where memory locks are nearly instantaneous.
- Python Performance: In Python, the interpreter overhead and the Global Interpreter Lock (GIL) significantly increase the base cost of thread operations (to 16.95 µs), masking the relative "weight" of the network stack overhead.

4. Conclusion

The investigation confirms that Shared Memory (Threads) is the most efficient method for communication when components reside within the same process. However, the performance advantage is most dramatic in compiled languages like C++, where thread synchronization is orders of magnitude faster than network calls. UDP provides a viable middle-ground for inter-process communication (IPC), while Web Sockets introduce the highest overhead due to protocol complexity.