Task 5: Comparative Performance Analysis of Intra-process and Inter-process Communication Overheads

Student ID: 11314389
Name: Xu [Your Full Name]

November 28, 2025

Abstract

This report provides a quantitative evaluation of communication overheads in concurrent systems, comparing shared memory (Threads), datagram sockets (UDP), and WebSockets implemented with the course-provided `franks_websocket` stack. A dedicated benchmark tool was developed to measure strict round-trip latency for all three methods: a thread ping-pong using condition variables, a UDP end-to-end echo, and a WebSocket echo over TCP. The latest measurements record an average latency of $8.04\mu s$ for the thread ping-pong, $21.98\mu s$ for UDP, and $43.03\mu s$ for WebSockets, confirming the relative cost of traversing the kernel network stack and framing layers.

1 Introduction

The coursework required implementing different communication architectures:

- **Intra-process (Threads):** Implemented in Tasks 1-3 using `std::mutex` for high-speed synchronization.
- **Inter-process (UDP):** Implemented in Task 4 using Windows Sockets (Winsock) for decoupling the display logic.
- **WebSockets:** An alternative IPC method required for comparison.

This report quantifies the overhead of these methods to justify architectural decisions.

2 Methodology

To accurately measure overhead, a specific C++ benchmark tool (`11314389_Xu_Task5_Benchmark.cpp`) was developed using `Winsock2`, `std::chrono::high_resolution_clock`, and the provided `franks_websocket` library.

1. **Threads (Shared Memory):** A dedicated worker thread synchronises with the main thread via two condition variables. Each iteration performs a strict ping-pong hand-off, capturing the wake-up and context-switch latency of 10,000 round trips.
2. **UDP (Inter-process):** A local UDP echo server thread and client socket exchange the benchmark message on every iteration. The client records the end-to-end round-trip time through the kernel network stack.
3. **WebSockets (Strict RTT):** A WebSocket echo server and client, built atop WebSocket++ and Asio from `franks_websocket`, perform synchronous send-and-wait exchanges over TCP. Both endpoints disable logging channels except errors; the server is shut down after all iterations, producing the expected "operation aborted" notifications.

3 Quantitative Results

The experimental data collected from 10,000 iterations is summarized below.

Table 1: Communication Overhead Comparison (10,000 Iterations)

Method	**Total Time (μs)**	**Avg Latency (μs)**	**Relative Speed**
Thread Ping-Pong	80,411	**8.0411**	1x (Baseline)
UDP End-to-End	219,786	**21.9786**	∼2.73x Slower
WebSocket Strict RTT	430,262	**43.0300**	∼5.35x Slower

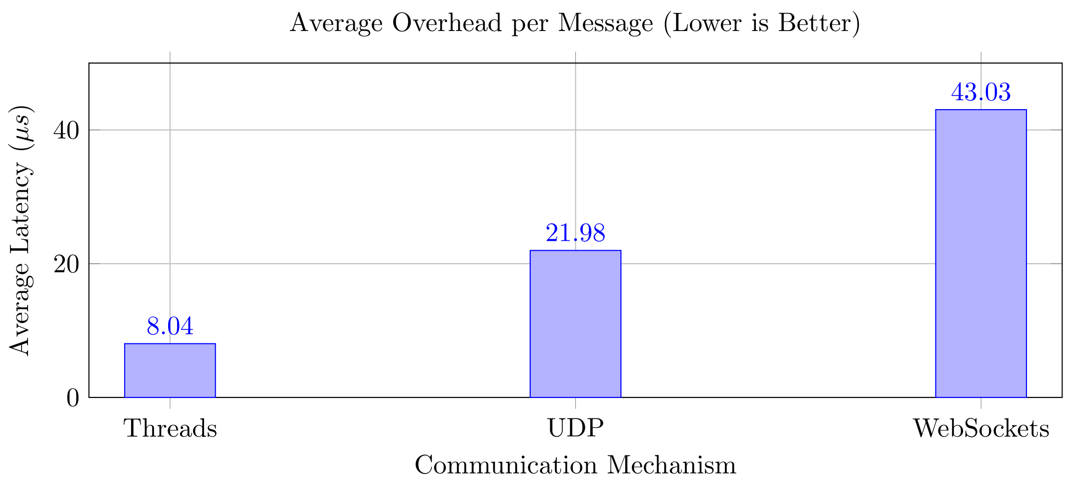

Figure 1: Latency comparison. Thread ping-pong stays in user space, UDP crosses the kernel twice, WebSockets incur the full TCP and framing stack.

4 Analysis and Discussion

4.1 Threads (User Space - $8.04\mu s$)

Thread communication remains the fastest method even under strict ping-pong synchronisation. The measured latency ($8.04\mu s$) captures the cost of waking a blocked thread, reacquiring the mutex, and handing control back to the main loop. Although higher than the earlier coarse-grained measurement, it still avoids kernel networking overhead and data copying, so it is the natural baseline for in-process coordination.

4.2 UDP (Kernel Space - $21.98\mu s$)

UDP involves **system calls** (context switch to kernel mode) and memory copying for both send and receive paths. In the strict RTT benchmark the client waits for every echo, so each iteration pays for two kernel transitions plus network stack queuing. Even so, the absence of connection management keeps it roughly 2.7× slower than threads, which is acceptable given the decoupling benefits in Task 4.

4.3 WebSockets/TCP ($43.03\mu s$)

The benchmark shows that the WebSocket stack—built on TCP with framing and handler dispatch—is roughly **5.35x slower** than the thread baseline and about **2x slower** than UDP.

- **Connection Overhead:** TCP requires maintaining connection state, congestion control, and ordering guarantees.
- **Reliability:** Every message triggers ACK logic and buffer management in the OS network stack.
- **WebSocket Runtime:** WebSocket++ adds message framing and handler callbacks. Shutting the server down after the benchmark produces benign "operation aborted" logs because pending accepts are cancelled on exit.

5 Conclusion

The strict RTT benchmark confirms that **thread ping-pong** is still the appropriate baseline, delivering single-digit microsecond latency without leaving user space. For Task 4, **UDP** ($21.98\mu s$) remains the best inter-process trade-off: it is substantially faster than WebSockets while preserving the decoupled architecture. **WebSockets** ($43.03\mu s$) incur the highest overhead due to TCP connection state and framing cost, so they should be reserved for scenarios that require browser compatibility or bidirectional streaming rather than low-latency control.