**Subject: BSDS** **CS 6650 — Programming Assignment #2**
**Name:** Samyak Shah **NUID:** 002051486

**Overview.** This assignment employs a TCP socket–based client–server robot factory and stubs. Customers order, the server makes. Expert robots require expert engineers maintained by a shared FIFO queue and expert pool thread. We measure per-order latency (µs) and end-to-end throughput (orders/s) as we vary load and expert capacity.

**Design Summary**

- **Transport & framing.** All communication is over TCP. Each message is framed as: `[4B type][4B length][payload]` with network byte order. Two message types: `ORDER` and `ROBOT`. Payloads are fixed-field structs serialized as 32-bit integers.
- **Stubs.**
  - `ClientStub.Init(ip, port)` opens one TCP socket per customer thread.
  - `ClientStub.Order(order)` marshals `Order`, sends, then blocks waiting for a `RobotInfo` response.
  - `ServerStub.ReceiveOrder()` blocks until an `Order` arrives and unmarshals it.
  - `ServerStub.ShipRobot(robot)` marshals and sends `RobotInfo`.
- **Server concurrency (regular engineers).** The server waits for connections in a loop and forks one thread per customer connection. Each engineer processes a stream of orders, set its id, and either ship immediately (regular) or or call the expert pipeline.
- **Expert workflow (special robots).** A shared **FIFO queue** holds `ExpertRequest`s. A fixed-size **expert thread pool** pops requests, simulates at least 100 µs of work, gives `expert_id`, and replies.
- Here it is ensure that Expert IDs never overlap regular IDs.
- **Client workload model.** The client launches *C* customer threads. Each thread keeps at most one in-flight order (next order issued only after a response), generating *C* concurrent RPCs. We record latency as asked in the assignment, mean/min/max across all orders, and compute throughput.
- **Correctness notes.** One connection per customer. Server threads terminate cleanly when the client closes its socket.
- **What works.** Regular and special robots; many concurrent customers; expert queue + thread pool. Each order's metrics and overall throughput and all the metrics. Cross-machine runs on Khoury Linux, different machines and ports (for exp 4).

**How to run (Khoury Linux, cross-machine).**

- On server (machine A): `./server <port> <#experts>` e.g., `./server 20000 2`.
- On client (machine B): `./client <ip> <port> <#customers> <#orders_each> <robot_type>`.
- Regular robots: `robot_type=0`. Special robots: `robot_type=1`.

**Results & Plots (one page).** Each row shows latency (mean/min/max) and throughput as customers increase ($x$-axis). Attach the eight PNGs generated by the plotting script.

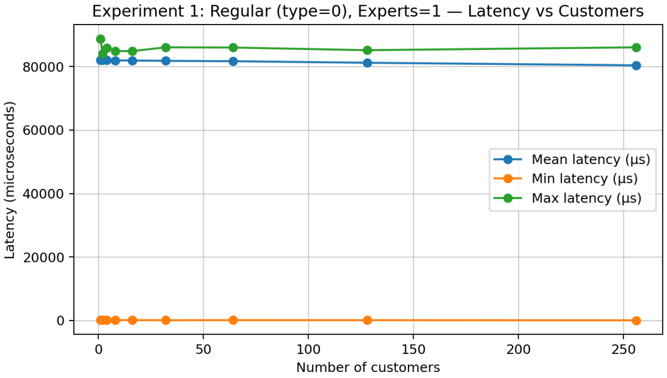


**Figure 1:** Exp1: Regular (type=0), Experts=1 — Latency

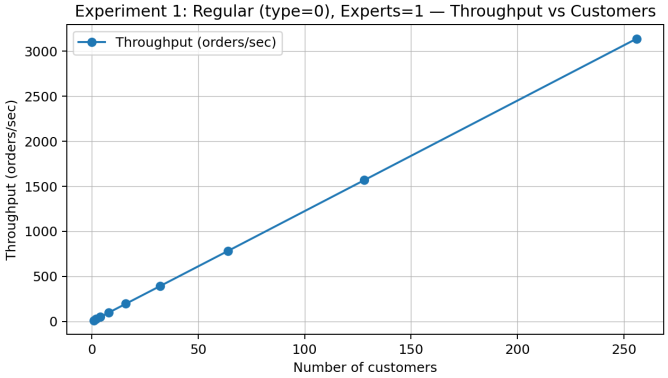


**Figure 2:** Exp1: Regular (type=0), Experts=1 — Throughput

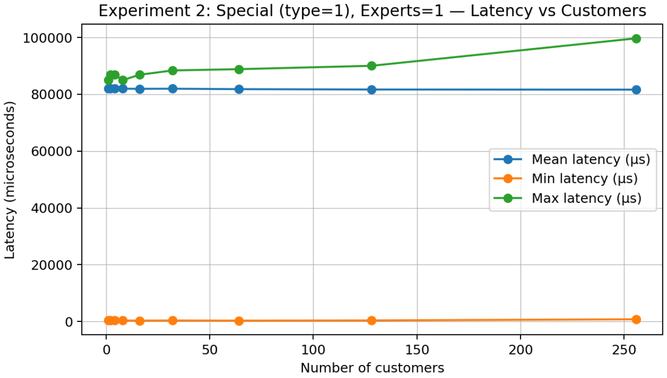


**Figure 3:** Exp2: Special (type=1), Experts=1 — Latency

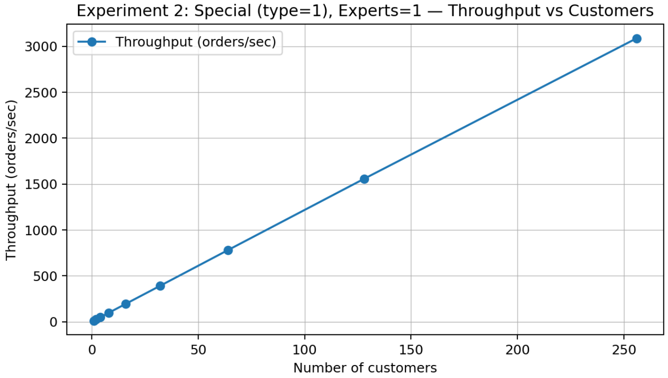


**Figure 4:** Exp2: Special (type=1), Experts=1 — Throughput

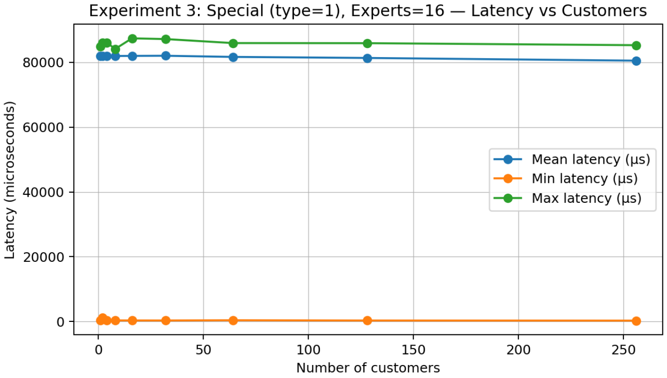


**Figure 5:** Exp3: Special (type=1), Experts=16 — Latency

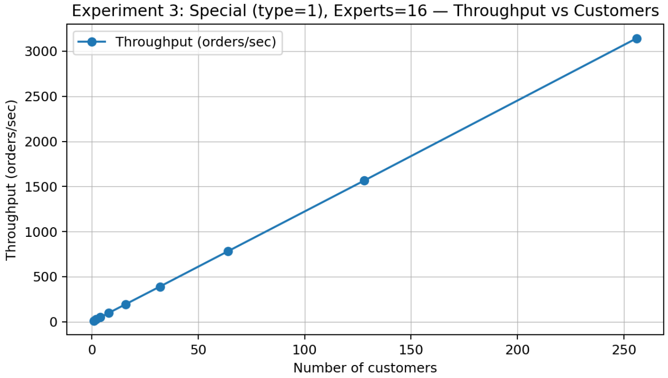


**Figure 6:** Exp3: Special (type=1), Experts=16 — Throughput

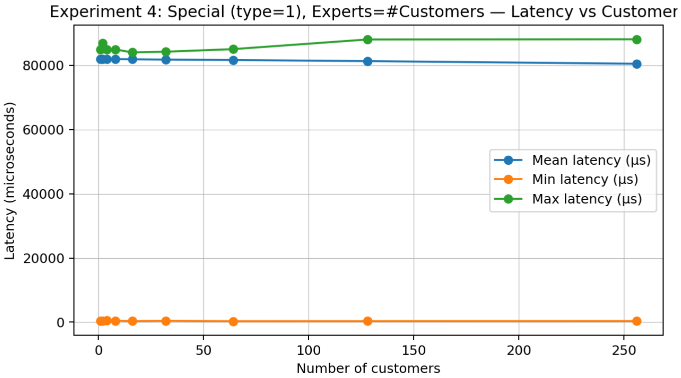


**Figure 7:** Exp4: Special (type=1), Experts=#Customers — Latency

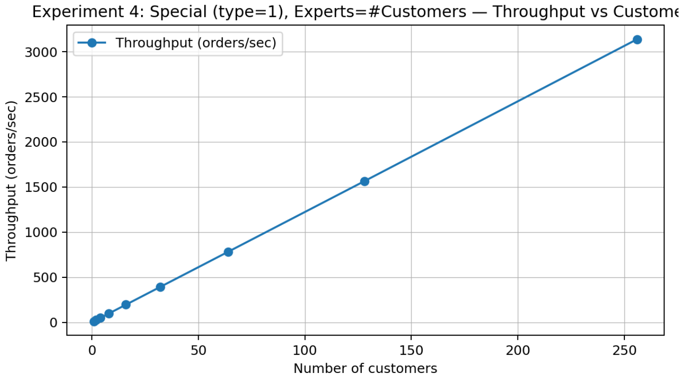
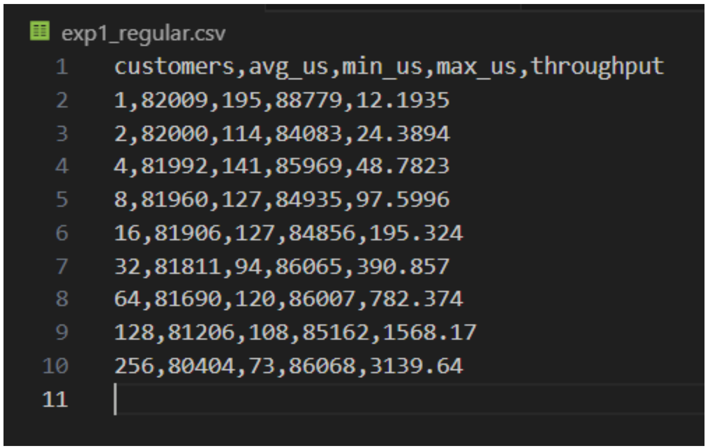


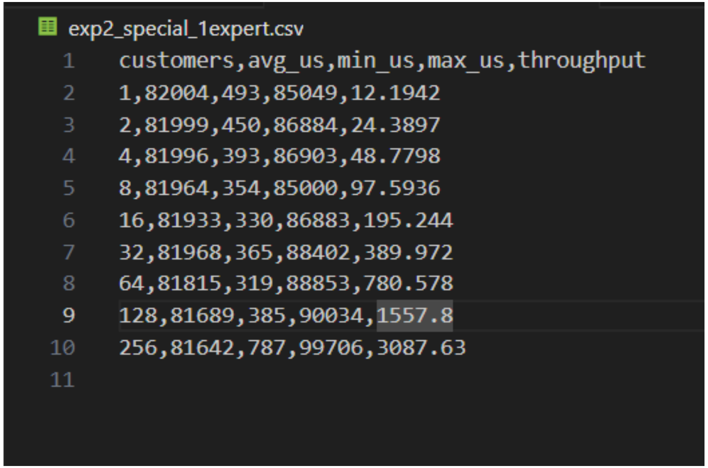
**Figure 8:** Exp4: Special (type=1), Experts=#Customers — Throughput

**Explanation:** Across Exp1–4, throughput scaled nearly linearly with customers, while mean latency stayed essentially flat at 82 ms, indicating TCP works good and the ($\leq 100\mu s$) expert step never became a bottleneck—even with just 1 expert—so 16 or customers experts match the regular baseline.



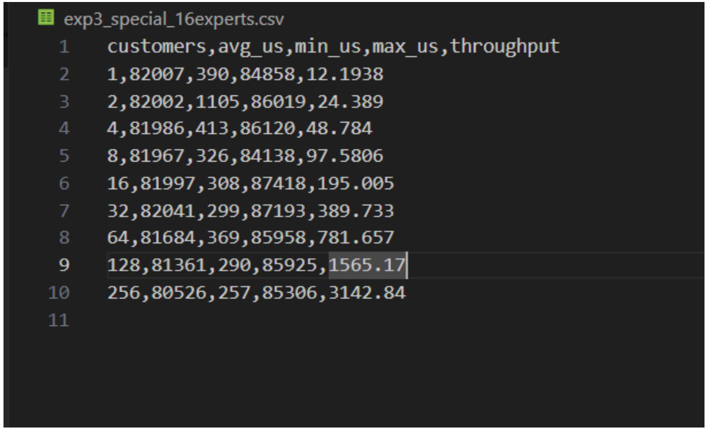
```
customers,avg_us,min_us,max_us,throughput
1,82009,195,88779,12.1935
2,82000,114,84083,24.3894
4,81992,141,85969,48.7823
8,81960,127,84935,97.5996
16,81906,127,84856,195.324
32,81811,94,86065,390.857
64,81690,120,86007,782.374
128,81206,108,85162,1568.17
256,80404,73,86068,3139.64

```



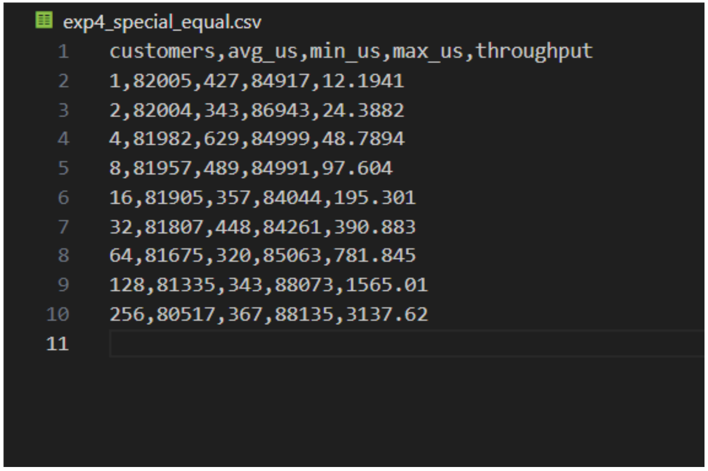
```
customers,avg_us,min_us,max_us,throughput
1,82004,493,85049,12.1942
2,81999,450,86884,24.3897
4,81996,393,86903,48.7798
8,81964,354,85000,97.5936
16,81933,330,86883,195.244
32,81968,365,88402,389.972
64,81815,319,88853,780.578
128,81689,385,90034,1557.8
256,81642,787,99706,3087.63

```



```
customers,avg_us,min_us,max_us,throughput
1,82007,390,84858,12.1938
2,82002,1105,86019,24.389
4,81986,413,86120,48.784
8,81967,326,84138,97.5806
16,81997,308,87418,195.005
32,82041,299,87193,389.733
64,81684,369,85958,781.657
128,81361,290,85925,1565.17
256,80526,257,85306,3142.84

```



```
customers,avg_us,min_us,max_us,throughput
1,82005,427,84917,12.1941
2,82004,343,86943,24.3882
4,81982,629,84999,48.7894
8,81957,489,84991,97.604
16,81905,357,84044,195.301
32,81807,448,84261,390.883
64,81675,320,85063,781.845
128,81335,343,88073,1565.01
256,80517,367,88135,3137.62

```