COL-334 Assignment 2: Scheduling and Fairness

Parts 2–4 Report

AYUSH SINGH 2023CS10322 PRITHVI SINGH 2023CS50077

September 7, 2025

Executive Summary

This report evaluates how client behavior and server scheduling affect fairness in a simple client–server system built on Mininet. Part 2 studies completion time as the number of clients increases. Part 3 demonstrates unfairness under FCFS when one client is greedy, and Part 4 shows how a round-robin scheduler restores fairness.

1 Setup & Reproducibility

- **Platform:** Ubuntu (Baadal VM), Mininet 2.x.
- **Python:** 3.10.x. Libraries: `matplotlib` (for plots).
- **Topology:** Single switch; one server host (`10.0.0.100`) and N clients (`10.0.0.1..10`). All links BW = 1 Mbps, equal delay/buffer.
- **Workload:** Static `words.txt` file of animal tokens; clients must download the *entire* file.
- **Common config:** Start offset p, chunk size k. Defaults used in our runs: server `10.0.0.100`, port 8887, $N = 10$, $p = 0$, $k = 5$.

How to run.

```
# Part 2
cd part2
make clean && make plot     # -> p2_plot.png, results_p2.csv

# Part 3 (FCFS)
cd ../part3
make clean && make plot     # -> p3_plot.png, results_p3.csv

# Part 4 (Round-Robin)
cd ../part4
make clean && make plot     # -> p4_plot.png, results_p4.csv
```

2 Client Behaviors

Normal client (baseline). Sends one request (p, k), waits for the response, then increments $p \leftarrow p + k$; repeats until `EOF`.

Greedy client. Always sends c back-to-back requests per burst:

$$(p, k), (p + k, k), \ldots, \big(p + (c - 1)k, k\big),$$

then waits for *all c responses*. If no `EOF`, updates $p \leftarrow p + c \cdot k$ and sends the next burst. (Per course clarifications: the greedy client waits for all c replies before the next burst and never reverts to normal.)

3 Server Scheduling Policies

Part 3 (FCFS). Single queue in arrival order.

Part 4 (Round-Robin). Per-client queues and a scheduler that cycles through clients in a fixed order, serving *one request per client* per round. This prevents a greedy client from monopolizing the server by flooding many back-to-back requests.

4 Metric: Jain's Fairness Index (JFI)

We first convert each client's completion time T_i to a throughput proxy $x_i = 1/T_i$. Then

$$\text{JFI} = \frac{(\sum_i x_i)^2}{n \cdot \sum_i x_i^2}, \quad 0 < \text{JFI} \leq 1.$$

Higher is better; 1.0 indicates perfect fairness.

5 Results

Part 2: Completion Time vs. Number of Clients

Observation. Mean completion time increases with more clients due to shared link/queue contention; variance typically grows as well.

Part 3 (FCFS): JFI vs. c (1–10)

Observation. As c rises (i.e., greed increases), JFI falls because FCFS admits the greedy client's back-to-back requests ahead of others.

Part 4 (Round-Robin): JFI vs. c (1–10)

Observation. JFI stays high and comparatively flat (≈ 0.8–1.0) across c because RR enforces one-request-per-round fairness.

6 Discussion & Limitations

- **Why FCFS is unfair.** Bursts from the greedy client dominate the head of the queue.
- **Why RR helps.** Turn-taking across clients prevents monopolization by any one client's burst.
- **Limitations.** RR enforces request turn-taking, not equal service *time*; if request sizes differ drastically, some residual unfairness may remain. Finite-file `EOF` effects can create small skews near completion. Mininet timing and TCP buffering add small measurement noise.

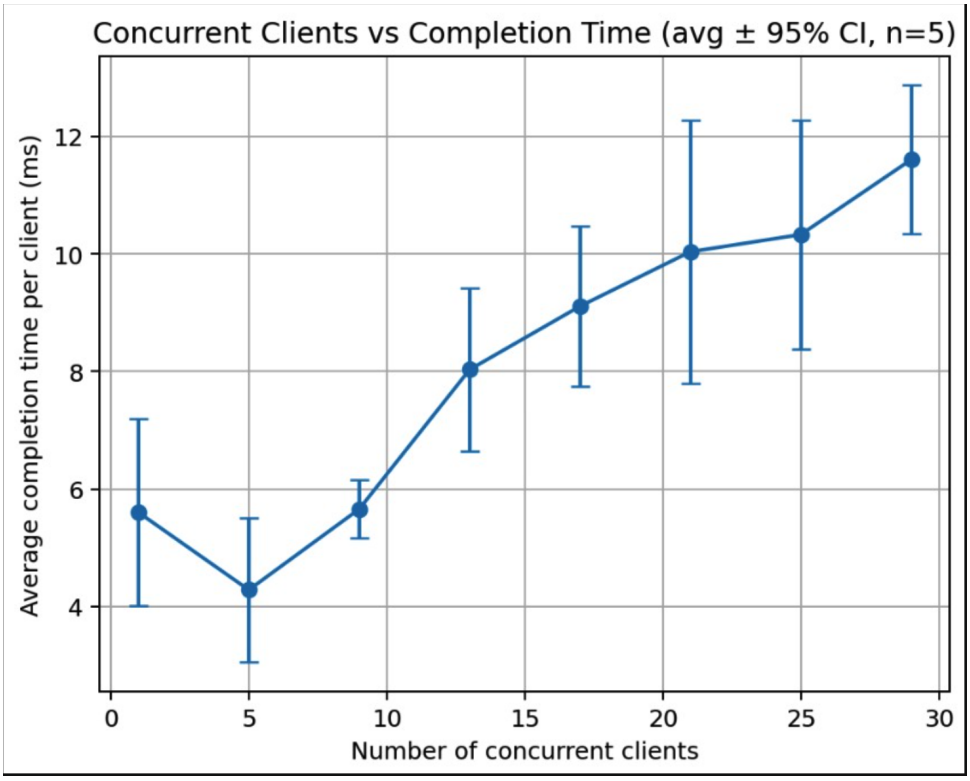

Figure 1: Avg completion time vs. clients — grows with contention.

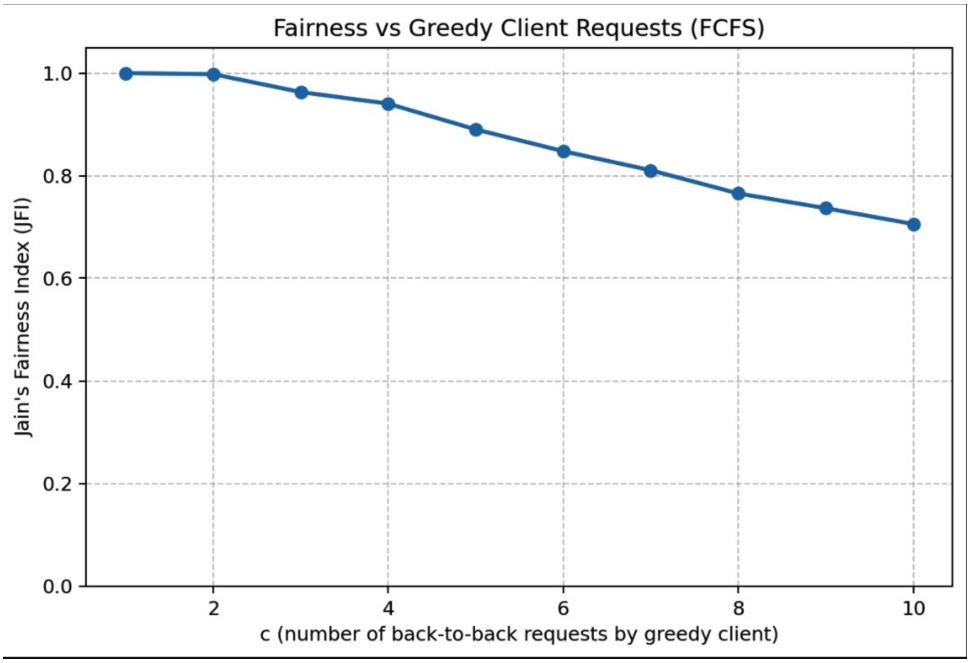

Figure 2: JFI vs. c — fairness drops as c increases

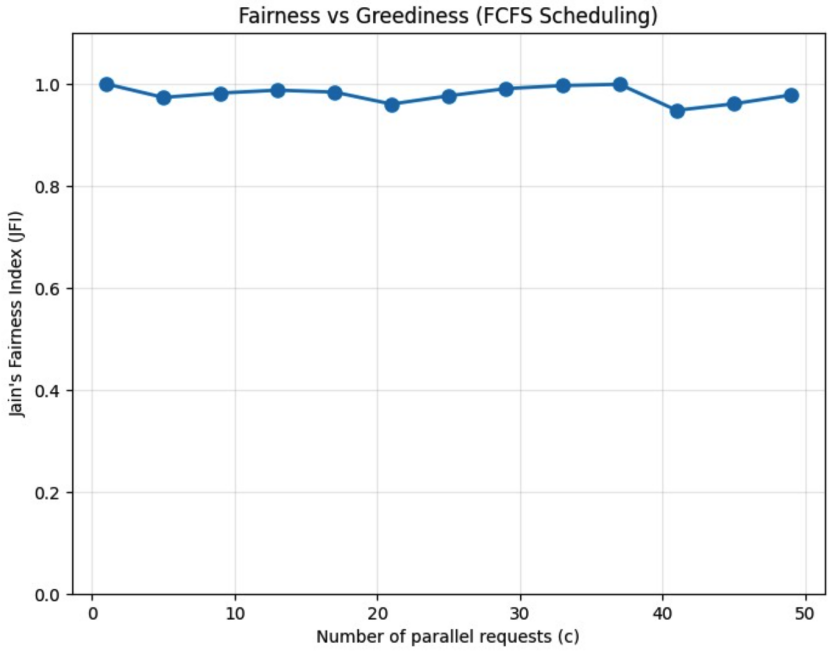

Figure 3: JFI vs. c — fairness stays high across c

7 Conclusion

FCFS becomes increasingly unfair as the greedy client's c grows. A round-robin server materially improves fairness, keeping JFI high and much less sensitive to c.

Appendix A: JFI (reference implementation)

```
def jfi(times):
    xs = [1.0/t for t in times if t > 0]
    if not xs: return 0.0
    s, s2, n = sum(xs), sum(x*x for x in xs), len(xs)
    return 0.0 if s2 == 0 else (s*s)/(n*s2)
```