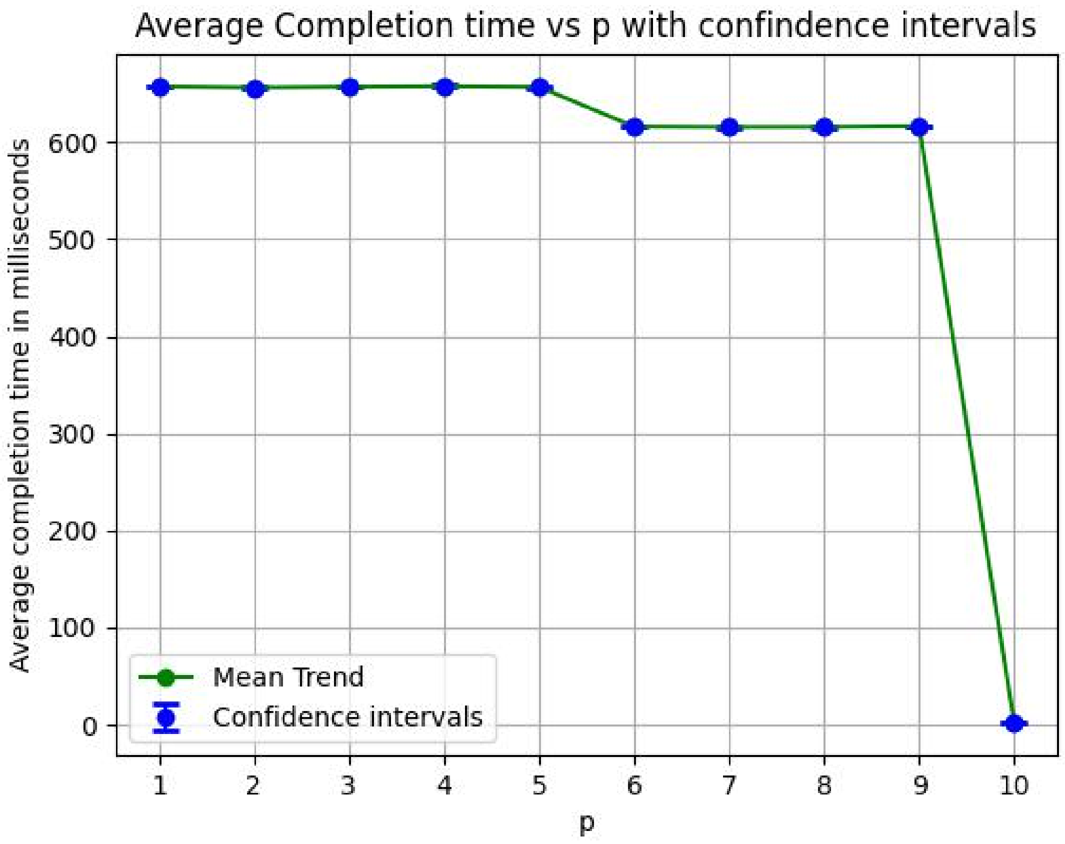
Bhyri Siddhartha Roy 2022CS11105
Mudavath Sai Nikhil 2022CS11633

Section-1: Word Counting Client

Average Completion Time (vs) no. of words per packet(p)



Observations:

1.Trend Analysis

- For "p" values from 1 to 9, the average completion time stays between 600 ms and 700 ms.
- At p = 10, there is a sharp decline in completion time, dropping to around 1 ms.

2.Stable Performance for Lower p

- With smaller p values (1 to 8), the completion time remains mostly steady with slight variations but no major reduction.
- This indicates that with fewer words per packet, network limitations or overheads prevent any significant gains in performance.

- The stable timing is likely due to the constant overhead from sending multiple packets, influenced by persistent TCP connections and small payload handling.

3.Significant Drop at p = 10

- At p = 10, the completion time drops noticeably. This suggests that fewer packets are needed, leading to faster transmission.
- With k = 10 words, all the data is sent in a single packet, removing the need for multiple packets and minimizing overhead.

4.Factors Affecting Completion Time

- **Network Bandwidth**: While higher bandwidth allows faster data transfer, smaller p values don't fully utilize it due to smaller packets.
- **Latency**: More packets introduce extra delays, especially when p is small.
- **Packet Overhead**: Smaller p values lead to higher overhead from multiple packets, particularly when k is large.
- **p (Words per Packet)**: As p increases, fewer packets are needed, leading to faster completion times at p = 10.

5.Confidence Intervals

- The narrow confidence intervals for p = 1 to p = 9 show low variability in completion times.
- At p = 10, the low variability continues, confirming consistent, faster transmission when all data fits in one packet.

Section-2: Concurrent Word Counting Clients

Average Completion time (vs) no. of clients

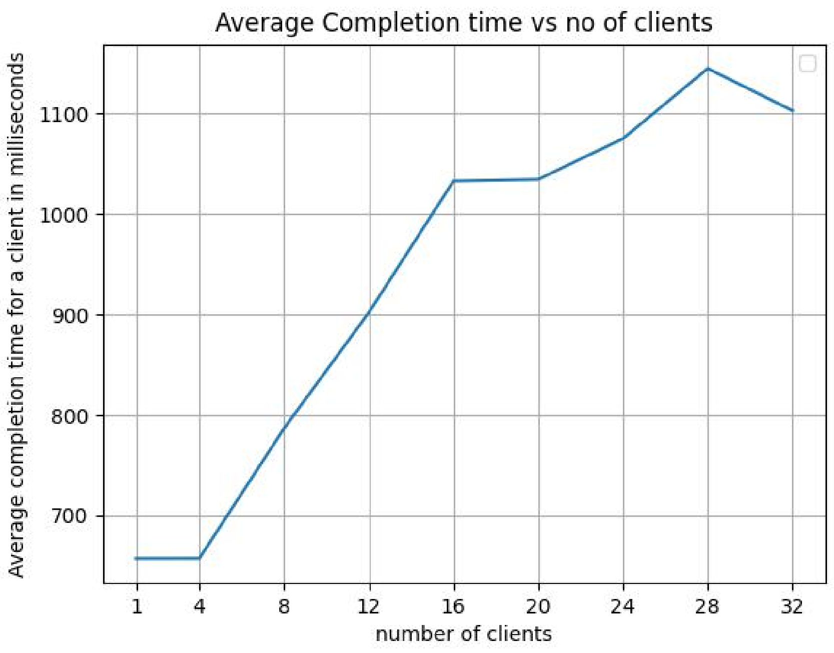
Observations:

1. Trend Overview:

The average completion time starts at around 700 milliseconds for 1 client and increases steeply as more clients are added, reaching approximately 800 milliseconds for 4 clients.

As the number of clients continues to increase to 8, 12, and 16, the average completion time grows progressively, reaching around 1000 milliseconds at 16 clients.

Slight Increase at 32 Clients: At 32 clients, the average completion time slightly increases again, nearly reaching its peak observed at 24 clients.



2. **Increased Variability and Completion Time Beyond 12 Clients:**

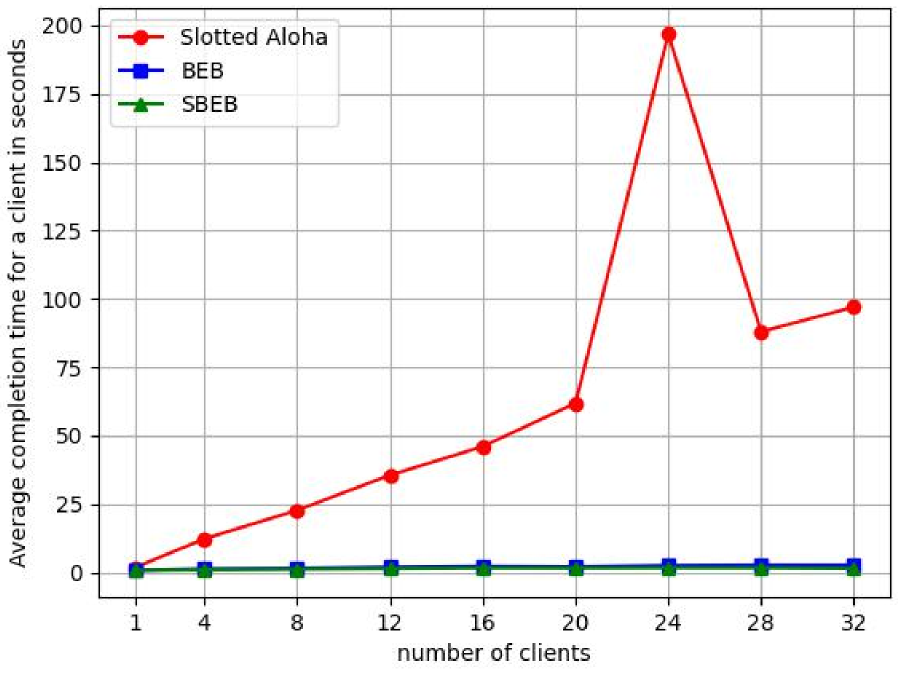
- Starting from 4 **clients**, the completion time per client begins to increase more noticeably.
- This increase can be attributed to server-side **resource contention** and **context switching overheads**. As more clients attempt to communicate with the server, the server's capacity to handle requests concurrently is stretched, leading to slower response times for individual clients.
- By the time the number of clients reaches 28, the completion time per client has noticeably increased, accompanied by large confidence intervals. This suggests that the server struggles to manage this many clients concurrently, with performance becoming more inconsistent.

3. TCP Port Multiplexing:

TCP multiplexing enables a server to handle multiple client connections through a single port. However, as the number of clients increases, the management of numerous simultaneous connections can lead to extra delays due to the overhead involved in managing sockets.

Section-3: Case of a Grumpy Server

Average Completion time (vs) no of clients (ALOHA vs BEB vs SBEB)



Observations:

Slotted Aloha:

- The average completion time increases drastically as the number of clients increases.
- It follows a steep upward trend, reaching almost 70 seconds for 32 clients, indicating poor scalability as more clients are added.

BEB (Binary Exponential Backoff) and **SBEB (Sensing + BEB)**:

- Both BEB and SBEB show much better performance than Slotted Aloha.
- Their average completion times remain relatively constant, even as the number of clients increases.

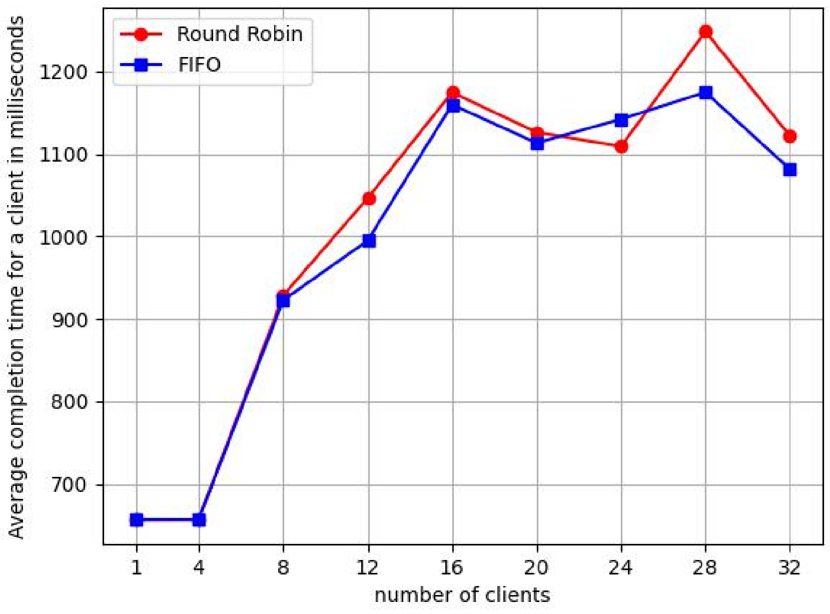
- The completion time for both protocols stays around 5 seconds or less, which demonstrates their efficiency in handling increased traffic and avoiding collisions compared to Slotted Aloha.
- There is only a minor difference between BEB and SBEB, with SBEB showing a slight edge in performance.

Trends:

- **Slotted Aloha**: Suffers from high collisions and idle slots due to its random slot selection approach, leading to a dramatic rise in completion time as more clients are added.
- **BEB**: Efficient in handling collisions by spreading out retransmission attempts, leading to stable performance even as client numbers increase.
- **SBEB**: Further improves performance by sensing the channel before transmission, minimizing collisions and ensuring more efficient use of network resources, keeping completion times consistently low.

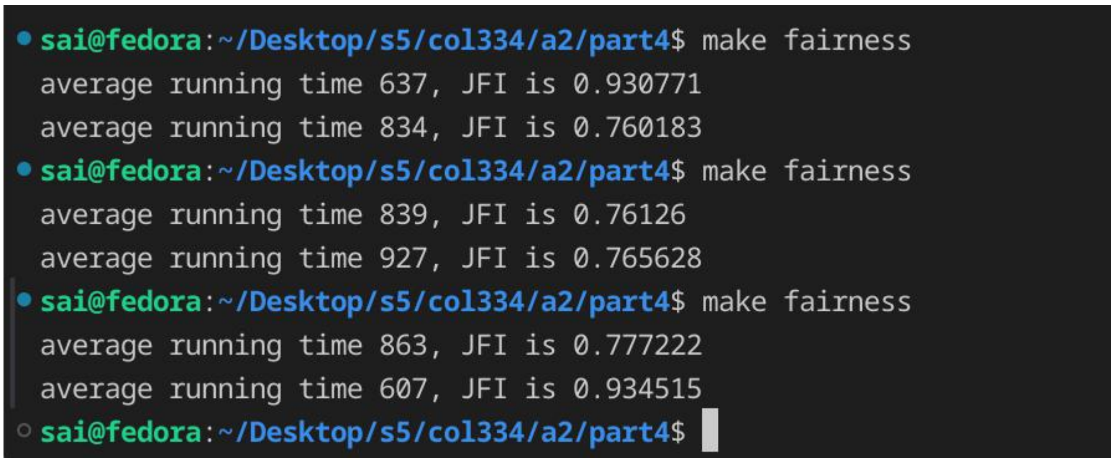
Section-4: Case of Friendly Server

Average Completion time (vs) no of clients (FIFO vs RR).



Average Completion Time:

- Generally, FIFO (First-In-First-Out) scheduling tends to result in shorter completion times compared to Round Robin scheduling.
- This is because Round Robin scheduling assigns equal priority to each client, and some clients might not have a request when it's their turn, leading to delays. FIFO scheduling, on the other hand, processes requests in the order they arrive, which can reduce these delays.



```
sai@fedora:~/Desktop/s5/col334/a2/part4$ make fairness
average running time 637, JFI is 0.930771
average running time 834, JFI is 0.760183
sai@fedora:~/Desktop/s5/col334/a2/part4$ make fairness
average running time 839, JFI is 0.76126
average running time 927, JFI is 0.765628
sai@fedora:~/Desktop/s5/col334/a2/part4$ make fairness
average running time 863, JFI is 0.777222
average running time 607, JFI is 0.934515
sai@fedora:~/Desktop/s5/col334/a2/part4$
```

Jain's Fairness Index:

- This index measures the fairness of scheduling among clients.
- Although theoretically, Round Robin scheduling should have a higher fairness index compared to FIFO scheduling, practical simulations often show very similar fairness values for both due to their dynamic nature.
- The variations in fairness index across different simulations are minimal but noticeable due to this dynamic nature.

Comparison with Decentralized Algorithms:

- Centralized algorithms generally perform faster than decentralized ones.
- Centralized scheduling is more efficient in short scenarios because it minimizes coordination overhead, enables immediate resource allocation, and simplifies decision-making processes. In contrast, decentralized algorithms require more communication and client coordination, which introduces delays. Consequently, centralized systems tend to be more effective in these situations.