Project 2: Simulation of a Network Router Queue

Arian Djahed

December 10, 2024

1 Simulation Design

The six python files that make up my whole simulation architecture consist of five separate classes for the various objects referenced throughout the simulation and another python script with the actual simulation logic and with the logger that outputs the data we are trying to procure.

The first of these classes is `Engine.py`, which—as the name suggests—simulates the routing engine that the packets go to once they leave the queue. It comes complete with parameters to indicate the current packet, the service time, and whether or not the engine is busy (along with getters and setters for each of these). There is also the `Packet.py` and `Event.py` classes; the former has parameters for packet ID, packet size, arrival time, and dequeue time; while the latter has parameters for event time and event type (and each have getters and setters for their respective parameters).

Then, there is the `PacketQueue.py` class, which specifically calls upon the aforementioned `Packet` class to make a queue specifically for packets and no other object. It comes complete with dedicated `enqueue` and `dequeue` functions that properly increment/decrement the number of items in the queue. I also designed it such that one could choose between either having an unlimited packet queue size or a limited packet queue size; one could achieve the former by setting `maxItems` to zero when instantiating the object, and one could achieve the latter by setting the same variable to any positive integer. The `EventQueue.py` class is similar, except it calls upon the `Event` class to make a queue specifically for events; as such, there is no limit to the queue by default as this was never a requirement for the event queue. In addition, here, the `enqueue` function behaves like a priority queue while the `dequeue` function behaves like a regular queue, as per the instructions.

Lastly, there is the `QueueSim.py` file, which is where the actual simulation happens, more specifically within the aptly-named `run_simulation` function. Upon running this file, which also imports the classes from all other files, the user is prompted to choose which test case to run between the three under which we were instructed to simulate our code. Then, thanks to the imported `logging` library, the results are then sent to `QueueSim.log` instead of printing them straight out onto the terminal. In all 3 of the test cases, the service rate is kept at a constant 100 bits per second, while the arrival rate is 0.001 packets per microsecond in the first case, 0.01 packets per microsecond in the second, and 0.1 packets per microsecond in the third.

2 Algorithms Used

Here, I will give "General Algorithm I" (as it is called in the instructions) as a pseudocode version of my python implementation in `QueueSim.py`:

2.1 General Algorithm I

```
FUNCTION run_simulation(num_packets, arrival_rate, service_rate)
    INITIALIZE event_queue, packet_queue, and engine objects

    SET total_wait_time to 0
    SET total_busy_time to 0
    SET max_queue_size to 0
```

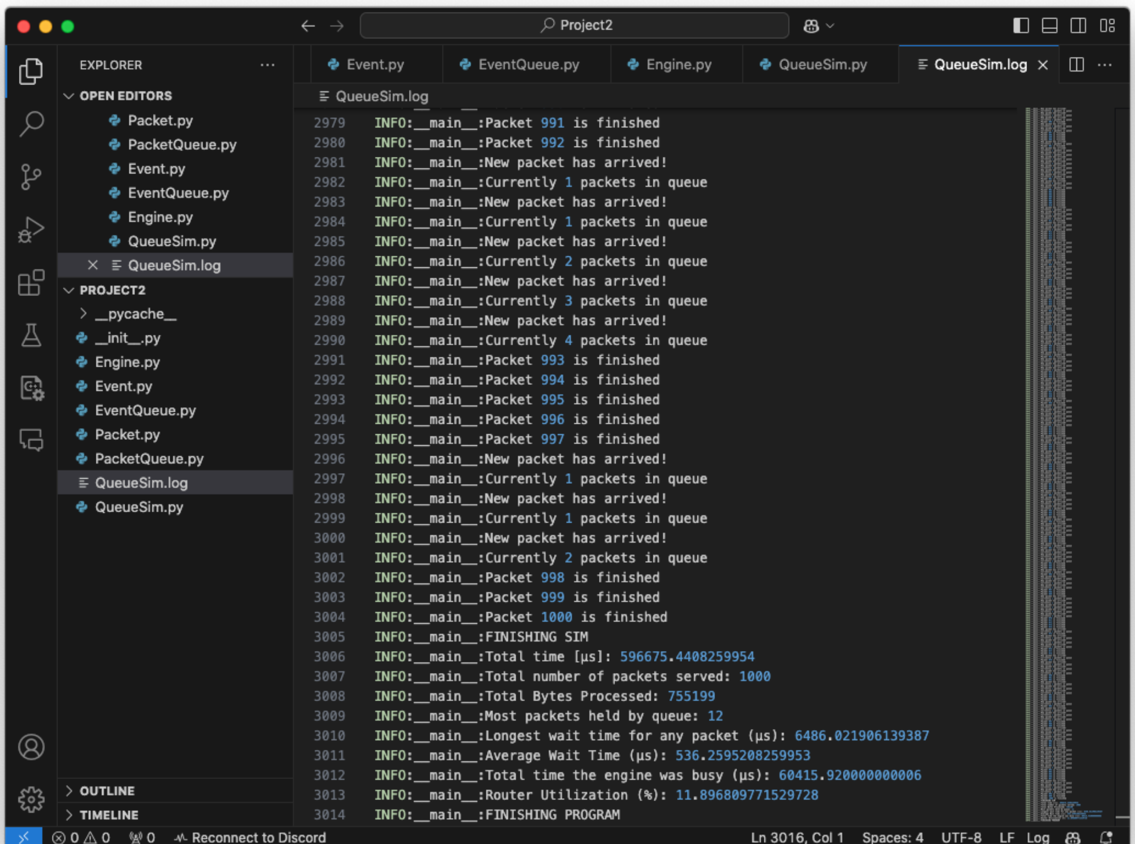
```
SET longest_wait_time to 0
SET total_bytes to 0

SET current_time to 0
SET arrival time based on arrival_rate
ENQUEUE new event based on arrival time onto event_queue
SET processed_packets to 0

WHILE processed_packets is less than num_packets
    DEQUEUE event from event_queue

    IF event_queue is empty
        SET arrival time based on arrival_rate
        ENQUEUE new event based on arrival time onto event_queue
    END IF

    IF current event type is "arrival"
        PRINT that a new packet has arrived
        INITIALIZE new packet object with random packet size
        ENQUEUE new packet to packet_queue
        SET new arrival time
        ENQUEUE new event based on arrival time onto event_queue

        IF engine is NOT busy
            DEQUEUE next packet to be serviced from packet_queue
            SET service_time to packet size divided by service_rate
            ENQUEUE new event based on service_time onto event_queue
            SET engine's busy status to TRUE
        END IF

    ELSE IF current event type is "departure"/"service completed"
        SET processed_packet to the current packet in the engine
        SET wait_time to processed_packet's dequeue time minus its arrival time
        SET longest_wait_time to the maximum between itself and wait_time
        ADD wait_time to total_wait_time
        ADD engine's service time to total_busy_time
        ADD processed_packet's size to total_bytes
        ADD one to processed_packets

        IF packet_queue is not currently empty:
            DEQUEUE next packet to be serviced from packet_queue
            SET service_time to packet size divided by service_rate
            ENQUEUE new event based on service_time onto event_queue
        ELSE SET engine's busy status to FALSE
        END IF

        PRINT that the packet is finished
    END IF
END WHILE

SET total_time to total_wait_time plus total_busy_time
SET avg_wait_tme to total_wait_time over num_packets
SET utilization_rate to total_busy_time over current_time
```

```
    RETURN total_time, item history of packet_queue, total_bytes, max_queue_size,
           longest_wait_time, avg_wait_time, total_busy_time, utilization
END FUNCTION
```

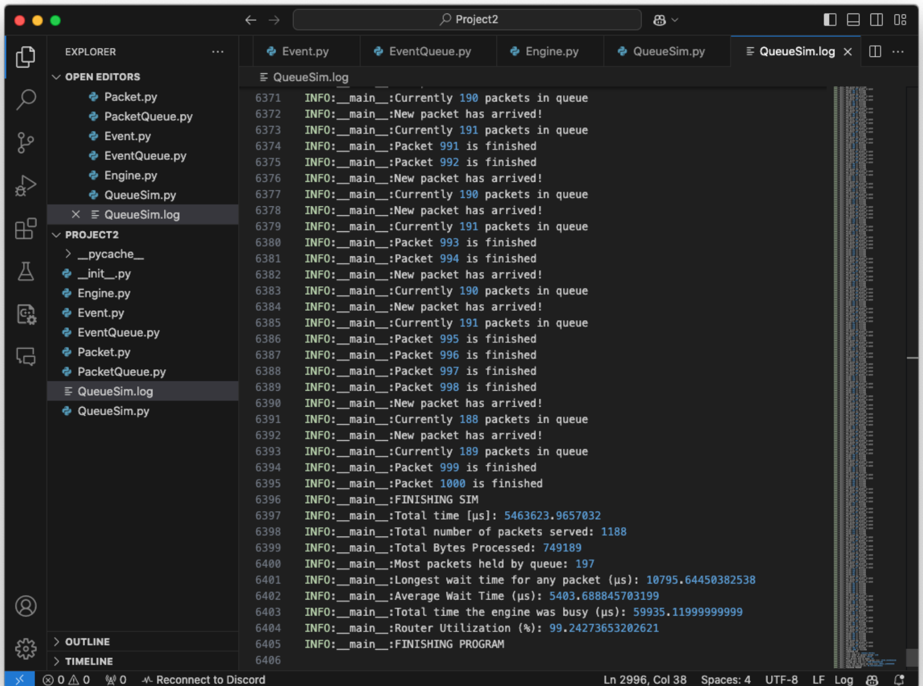
3 Sample Runs

Since I designed my program so that all the results would appear in a separate `.log` file, my sample runs will consist of screenshots of this file.

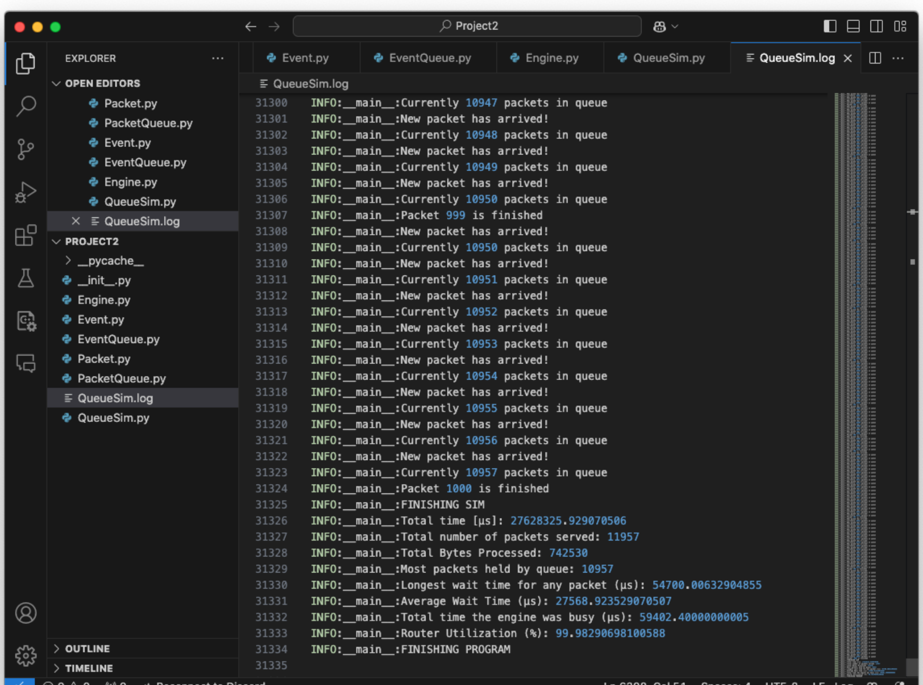
3.1 Test Case 1: Arrival Rate << Service Rate



3.2 Test Case 2: Arrival Rate close to Service Rate



3.3 Test Case 3: Arrival Rate >> Service Rate



4 Results

4.1 Data Table

Note: here r_a refers to the arrival rate and r_s refers to the service rate.

	$r_a << r_s$	$r_a \approx r_s$	$r_a >> r_s$
Total Time (μs)	596,675.44	5,463,623.97	27,628,325.93
Total number of packets served	1000	1000	1000
Total Bytes Processed	755,199	749,189	742,530
Most packets held by queue	12	197	10,957
Longest wait time for any packet (μs)	6,486.02	10,795.64	54,700.01
Average Wait Time (μs)	536.26	5,403.69	27,568.92
Total time the engine was busy (μs)	60,415.92	59,935.12	59,402.40
Router Utilization (%)	11.90	99.24	99.98

4.2 Comments

The first thing that seems to be of note is the fact that as the arrival rate increases by orders of magnitude, the total time increases by the same order of magnitude. It can also be seen that as the arrival rate increases but the service rate stays the same, the engine seems to get "overwhelmed" as a result of the packets coming in quicker but the engine not being able to service them any quicker; this is reflected in the total packets held by the queue increasing astronomically as the arrival rate increases. In addition, the fact that the longest and average wait time increase as the service rate increases also reflects this.

What is most interesting, though, is the fact that the total bytes processed and the total time the engine was busy remain virtually unchanged even as the arrival rate increases (besides some negligible perturbations). This most likely indicates that these are independent of the arrival rate. Furthermore, in the case of the total time that the engine was busy, it reveals that increasing the arrival rate whilst keeping the service rate constant—and thus "overwhelming" the engine—only affects the wait time between packet processing and not the actual time that it takes to process each packet. This, however, did affect the router utilization rate.