|  |  |  |  |
| --- | --- | --- | --- |
| BN | Section | Student ID | اسم الطالب |
| 19 | 2 | 9221268 | سراج خالد احمد عبد العال |
| 34 | 2 | 9220461 | عبدالرحمن حاتم نصر يوسف |

# 1 System Design

## 1.1 Overview

This FreeRTOS-based program synchronizes three sender tasks and one receiver task using semaphores and timers. Each task sleeps for a period, wakes up to perform its task, and then sleeps again. Semaphores, controlled by timers, ensure tasks execute only when triggered. Sender tasks get new random periods within a specified range each cycle, using the rand() function seeded by srand(time(NULL)) to ensure randomness across program runs.

int main(int argc, char\* argv[])

{

/\*Initiates the rand() function in GenerateRandomNumber() and lets it make a different random number each time\*/

srand(time(NULL));

//Rest of code here

.

.

}

int GenerateRandomNumber(int num1\_cpy, int num2\_cpy);

{

int random = 0;

random = (rand()%(num2\_cpy – num1\_cpy)) + num1\_cpy

return random;

}

For the receiver task, it has a constant period of 100ms. It receives the messages from the queue and checks if it received 1000 messages. If so, we increase the index of the Bound Array and print the statistics of the system so far. If not, we continue normally. And after the 6th iteration the program terminates.

## 1.2 Queue Operation

A Queue of size 3 is used in the project to store the messages of the sender tasks so that the receiver task can receiver messages from multiple senders. Code snippets of a part of the Sender and Receiver tasks are shown.

//Code snippet of SenderTaskOne

else if(xQueueSend(Queue, SentMessage1, 0) == pdTRUE)

/\*Checks if the message is sent successfully

{

//Rest of code here

}

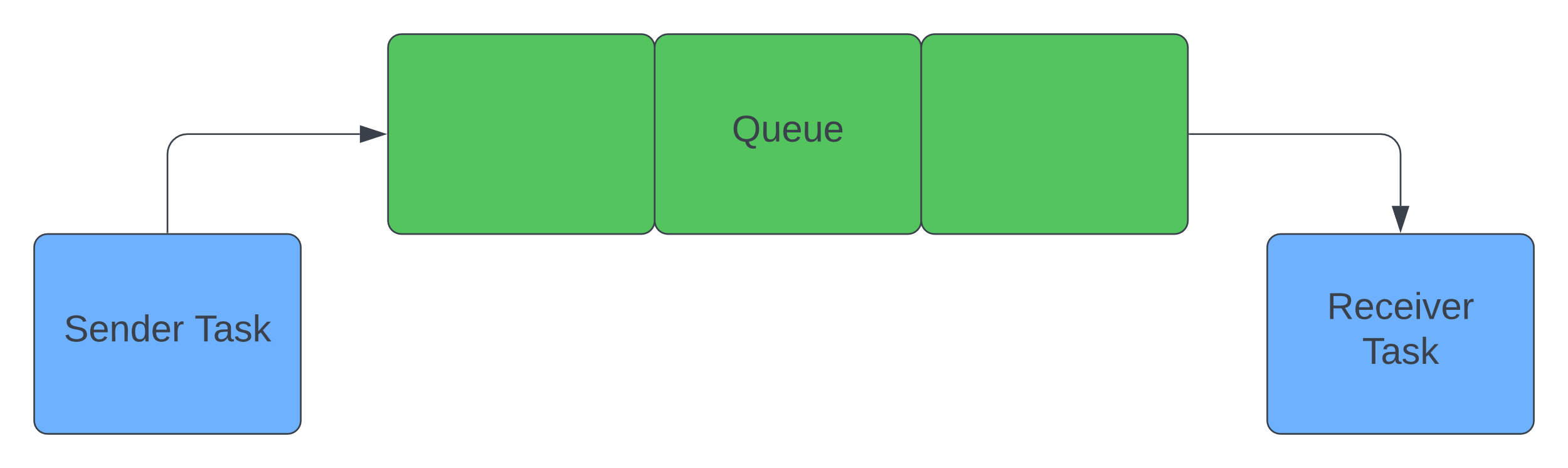


Figure 1 Showing Empty Queue Structure

//Code snippet of ReceiverTask

if (xQueueReceive(Queue, receivedMessage, 0) == pdPASS)

{

//Rest of Code here

}

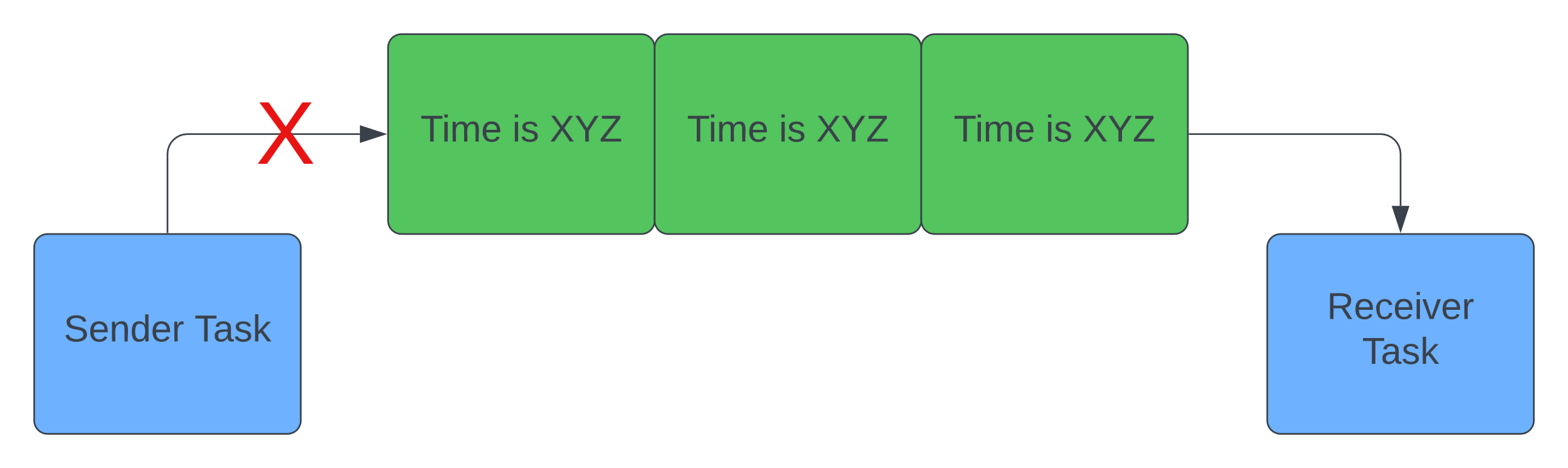
When the Queue is full the senders are blocked

Figure 2 Showing Full Queue Structure and Blocked Sender Task

## 1.3 Flowchart

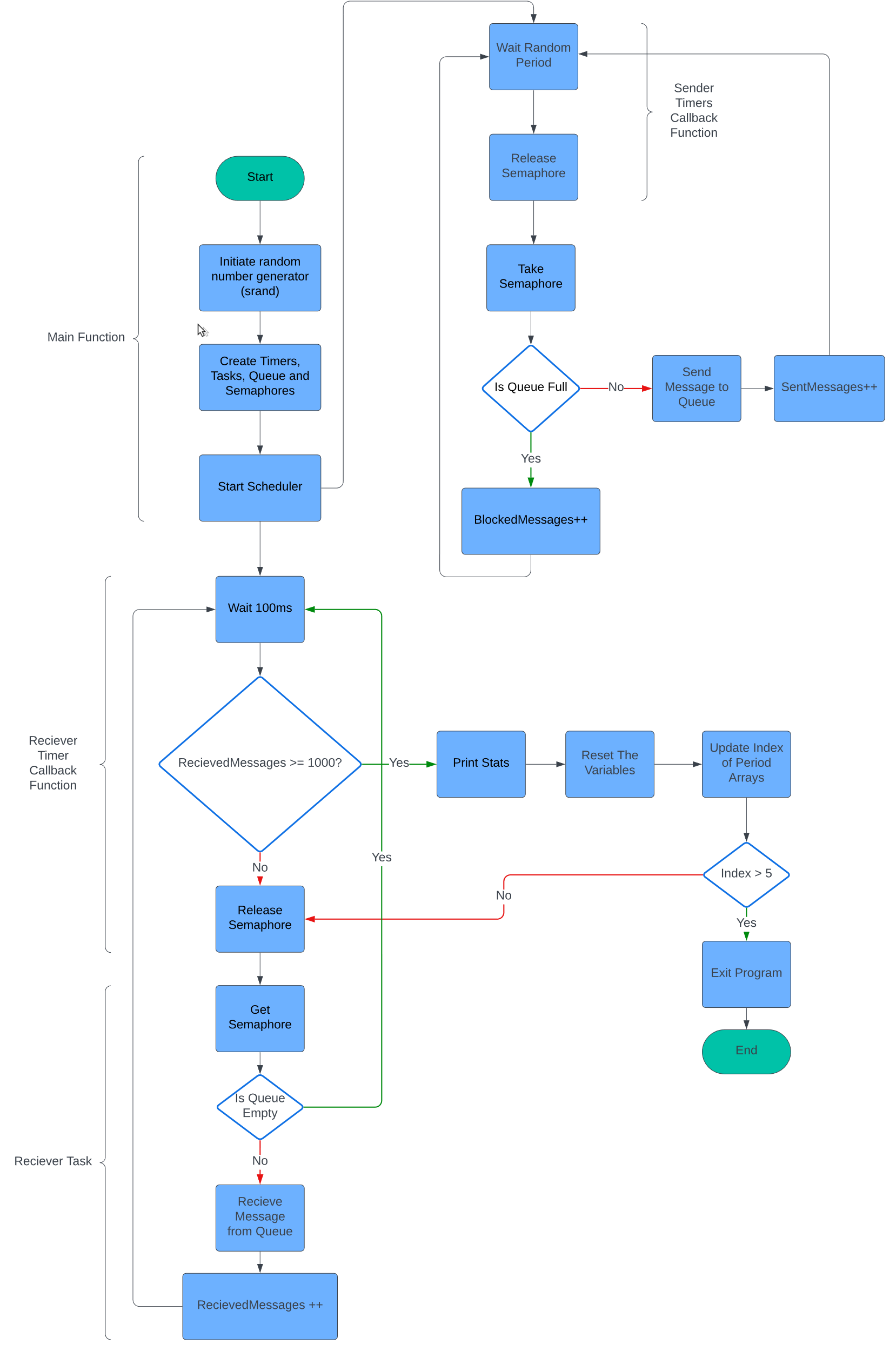


Figure 3 Flowchart of the Whole Program

# 2 Results and Discussion

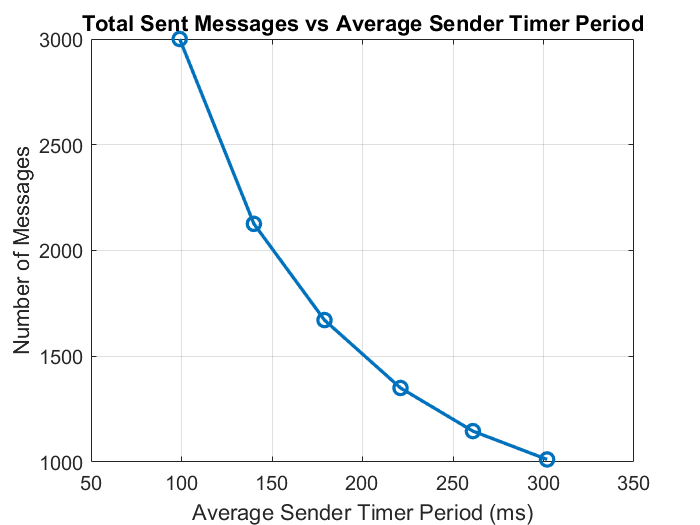
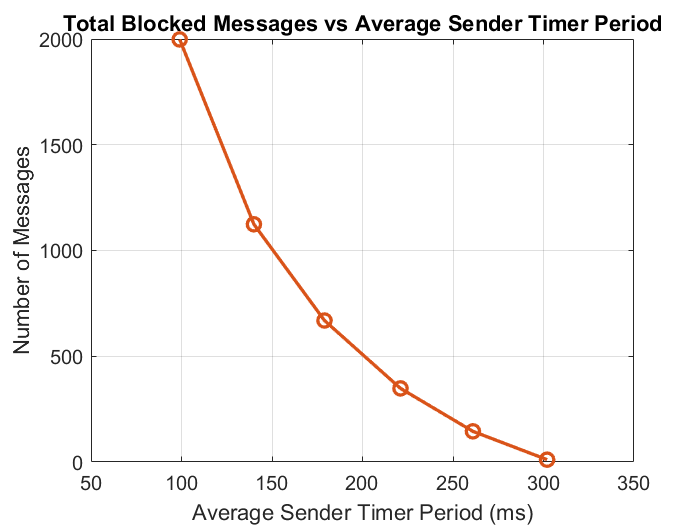
Here are the results for Queue size of 3:

Figure 5 Total Blocked Messages

Figure 4: Total Sent Messages

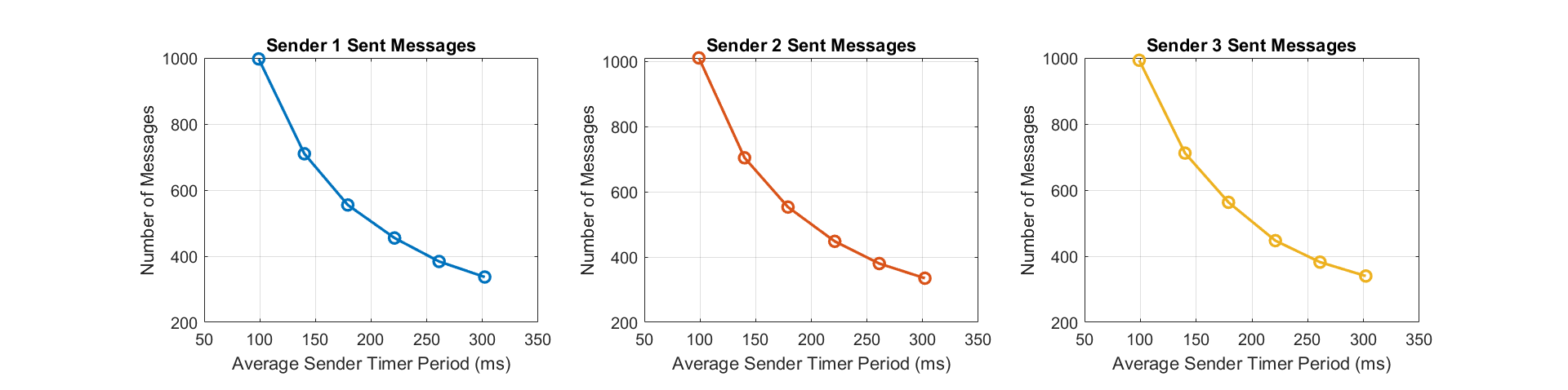
The Receiver task waits for a total of 1000 messages in each iteration. Once this count is reached, a new iteration starts with updated bounds. Initially, the three sender tasks transmit messages at a high rate due to the small values of random time periods in the early iterations. As the iterations progress and the bound values increase, the rate of message transmission decreases, resulting in a lower rate of message sending.

Figure 6: Sent Messages by each Sender Task

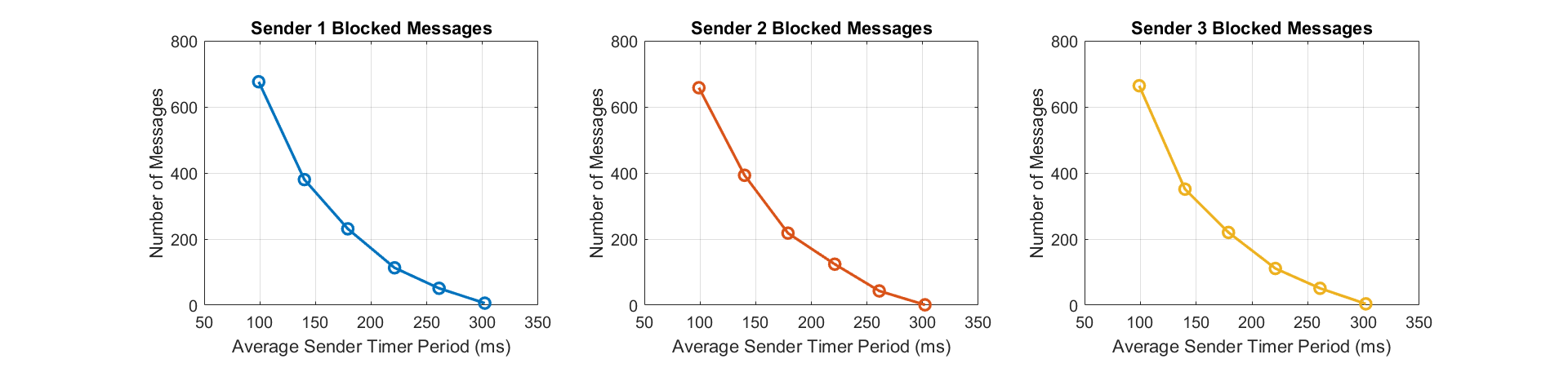
The tasks are almost sent at the same rate due to random durations taken by each task for sending messages. If a low-priority task (SenderOne() or SenderTwo()) and a high-priority task (SenderThree()) take the same amount of time, which rarely happens, the higher priority task will be sent first.

Figure 7: Blocked Messages by each Sender Task

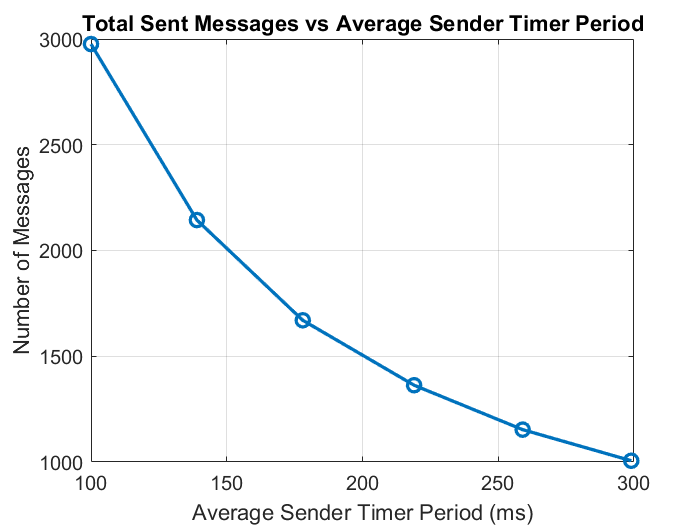
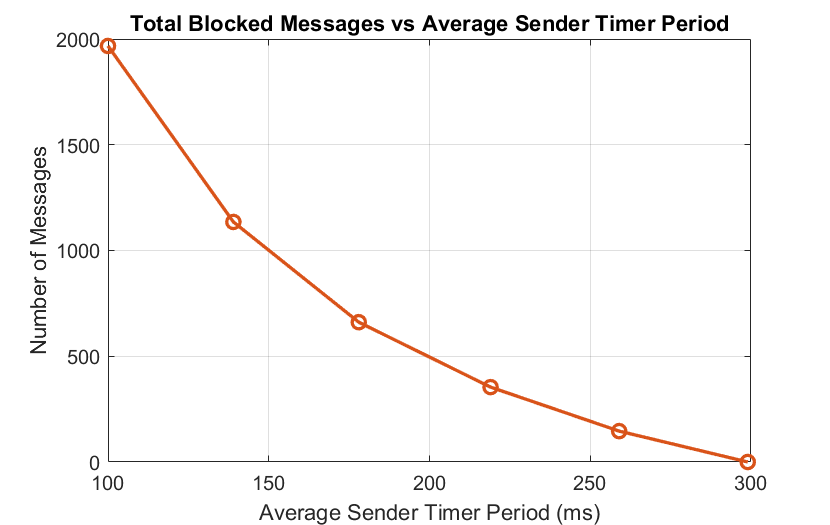
Here are the results for Queue size of 10:

Figure 9: Total Blocked Messages

Figure 8: Total Sent Messages

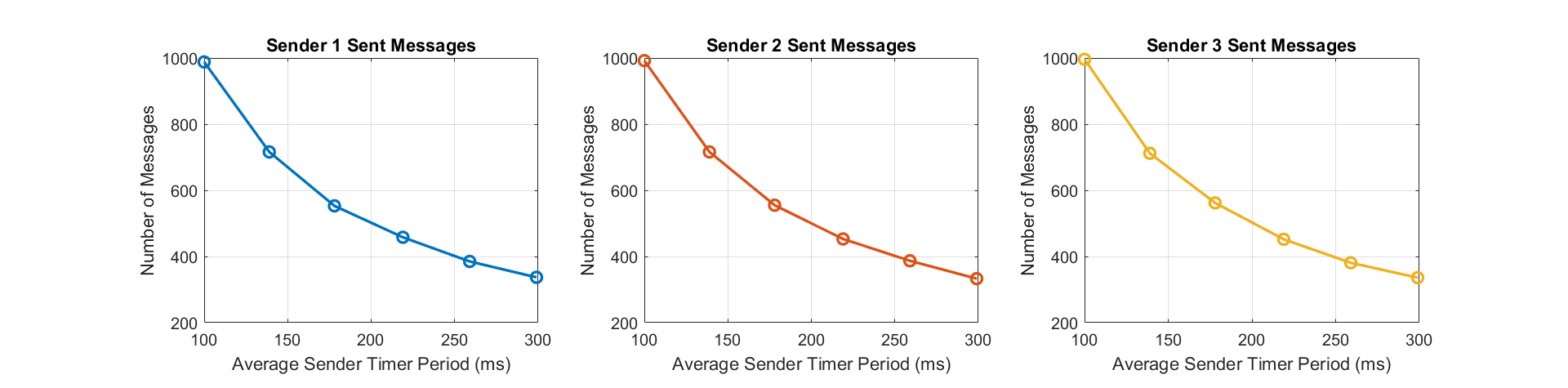


Figure 10 Sent Messages by each Sender task

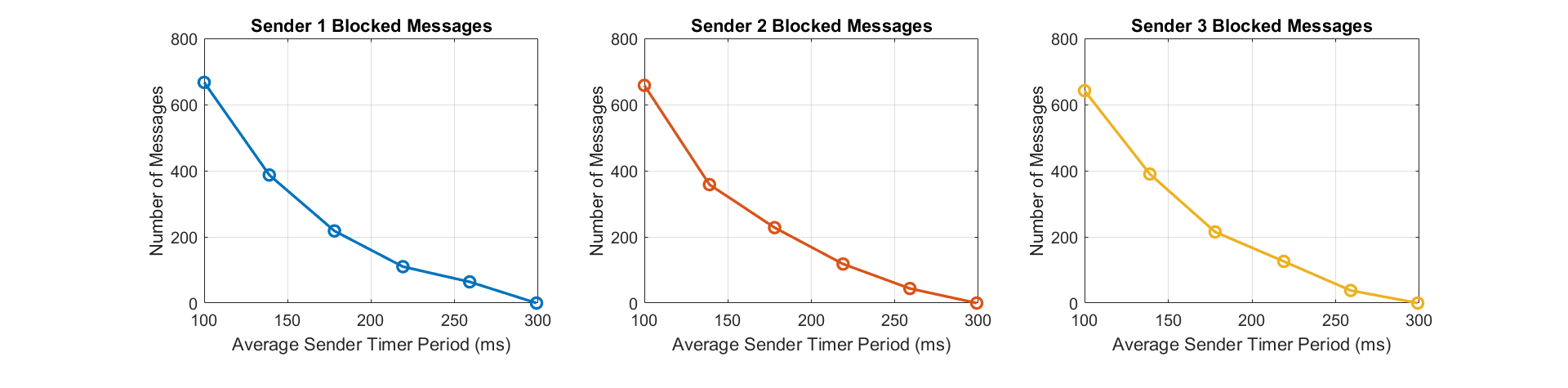


Figure 11: Blocked Messages by each Sender task

As observed from the graphs, when the queue size increases, it impacts the total number of blocked messages compared to the scenario with a smaller queue size of 3. In the larger queue scenario, the total number of blocked messages decreases due to more frequent intervals of an empty queue. However, the total number of sent messages likely remains consistent because the period bounds do not change. Therefore, each iteration still takes approximately the same random period of time, allowing tasks to send messages at a consistent rate as in the first case.

# References

[1] R. Barry, *Mastering the FreertosTM Real Time Kernel: A Hands-on Tutorial Guide*. Sin lugar de publicación: Real Time Engineers Ltd, 2016.