|  |  |  |  |
| --- | --- | --- | --- |
| **BN** | **Section** | **Student ID** | **اسم الطالب** |
| 46 | 1 | 9210273 | اية اشرف رشاد شاهين |
| 39 | 1 | 9210248 | اميرة مدحت ابراهيم حسن |

**RTOS PROJECT**

1. **System Design**
   1. **Overview**

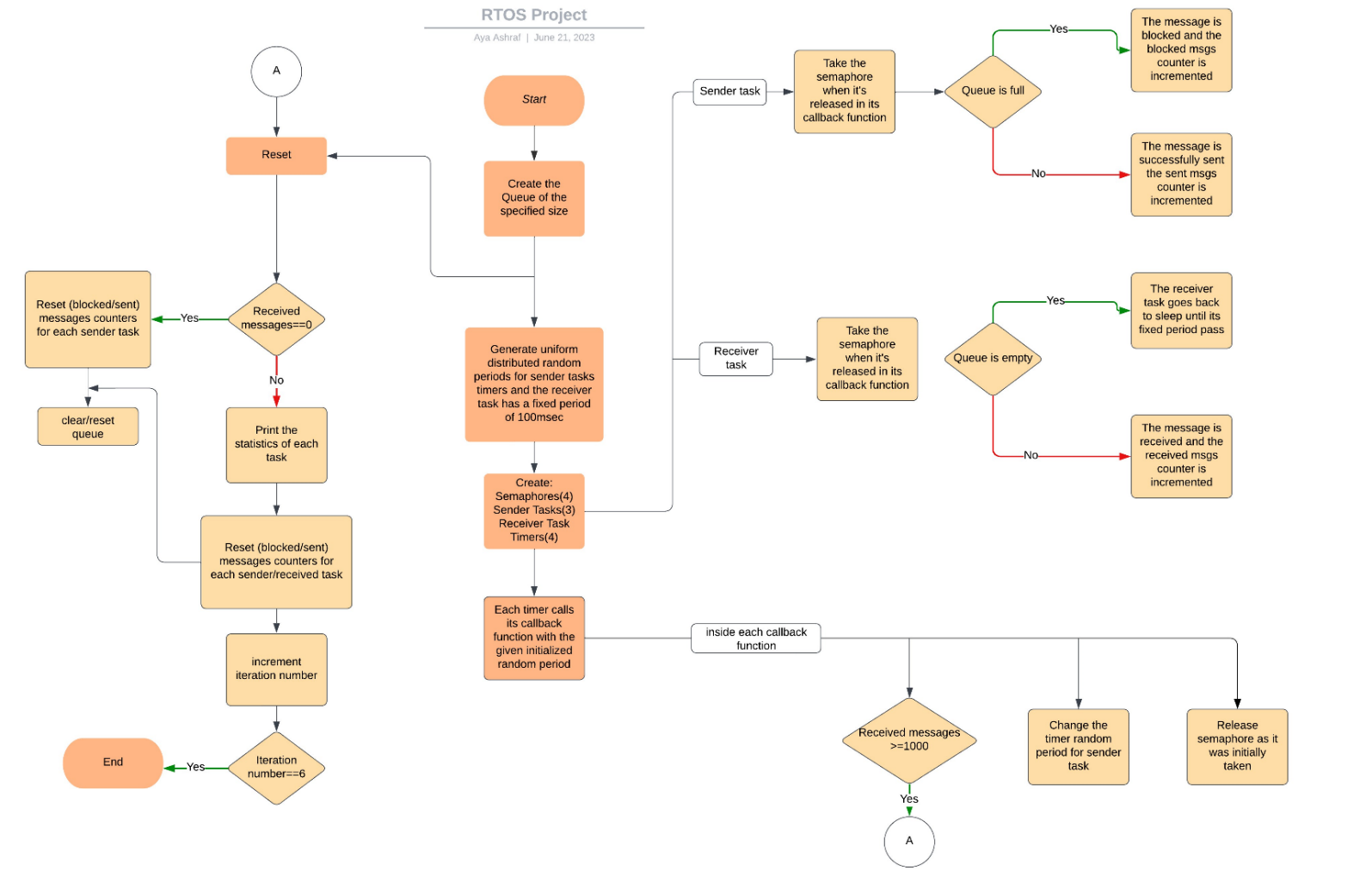
This system contains 4 tasks, 3 sender tasks and 1 receiver. Two of the sender tasks have the same priority (low) and the third one has the higher priority. Each of the tasks has its own timer (controlled by a uniform distributed random period) and its own callback function. The receiver and senders communicate through a queue only when the semaphores are released.

Figure : system flow

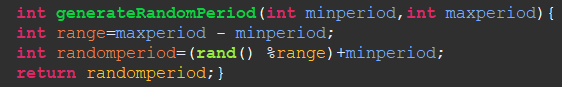
* 1. **Implementation**

****Initially, we create a queue of fixed size. Then we call the reset function for the first time which only resets all the global counters (received messages/transmitted messages per task / blocked messages per task). We create 4 semaphores for each task using the function in **Figure 2**

Figure

as it makes the semaphore initially taken so the tasks stay asleep. After that, we create the timers of sender tasks and as soon as the random period generated for each task passes the timer calls its callback function so the sender task semaphore could be released so it‘ll be available for the sender task to take the semaphore and start the operation of sending the message to queue if it’s not full then increments the counter of the transmitted messages but if it’s full the message is blocked then increments the counter of the blocked messages and the sender task goes back to sleep until another generated random period passes and restarts the whole process again. For the receiver task its timer is controlled by a fixed period (100 msec) so its callback function does the same process as the sender task but with a fixed time then as soon as its semaphore is released the receiver starts to check the queue if it’s not empty it receives the message successfully and increments the counter of the received messages then goes back to sleep, if it’s empty then it sleeps immediately and restart the whole process after another 100msec passes.

We defined two arrays to represent the different values of lower and upper bounds LOWER= {50, 80, 110, 140, 170, 200}, UPPER= {150, 200, 250, 300,350, 400} and used the function in **Figure 4** to generate a uniform distributed random number for each sender task. But we have to define its seed function in main() as in **Figure 3.**

****

Figure

Figure

Each iteration ends when the received messages counter reaches 1000 and we check it at the beginning of each callback function. If so, we call the Reset function, which prints the statistics for each sender and then resets the sent/blocked messages counters again and clears the queue to repeat the process for another iteration. Eventually, at the end of the reset function we check if the system reached the last iteration so if true we terminate the system by exit(0) function.

1. **Results and Discussion**

For queue of size = 3:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sender 1 | | Sender 2 | | Sender 3 | | Total transmitted messages | Total Blocked messages |
| Iteration | Sent messages | Blocked messages | Sent messages | Blocked messages | Sent messages | Blocked messages | \_\_\_\_\_ | \_\_\_\_\_ |
| 1 | 320 | 683 | 329 | 682 | 353 | 656 | 1002 | 2021 |
| 2 | 298 | 404 | 347 | 378 | 357 | 360 | 1002 | 1142 |
| 3 | 322 | 231 | 341 | 222 | 393 | 212 | 1002 | 665 |
| 4 | 329 | 125 | 331 | 124 | 342 | 118 | 1002 | 367 |
| 5 | 343 | 45 | 325 | 59 | 334 | 48 | 1002 | 152 |
| 6 | 336 | 4 | 331 | 3 | 335 | 1 | 1002 | 8 |

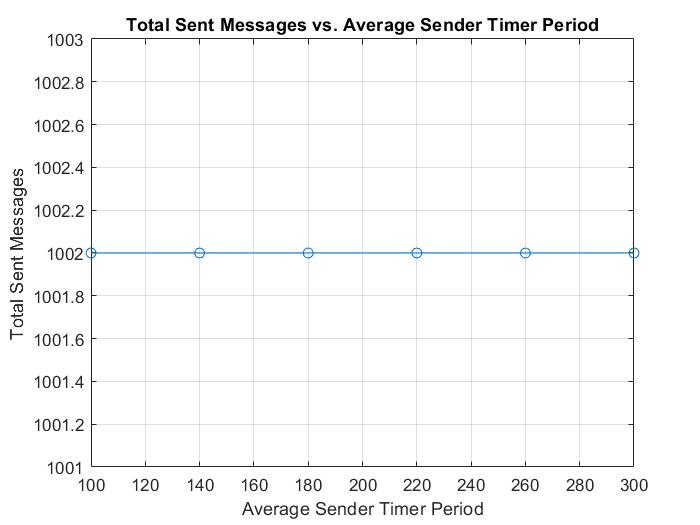
****

Figure : total sent messages VS avg Sender period for queue=3

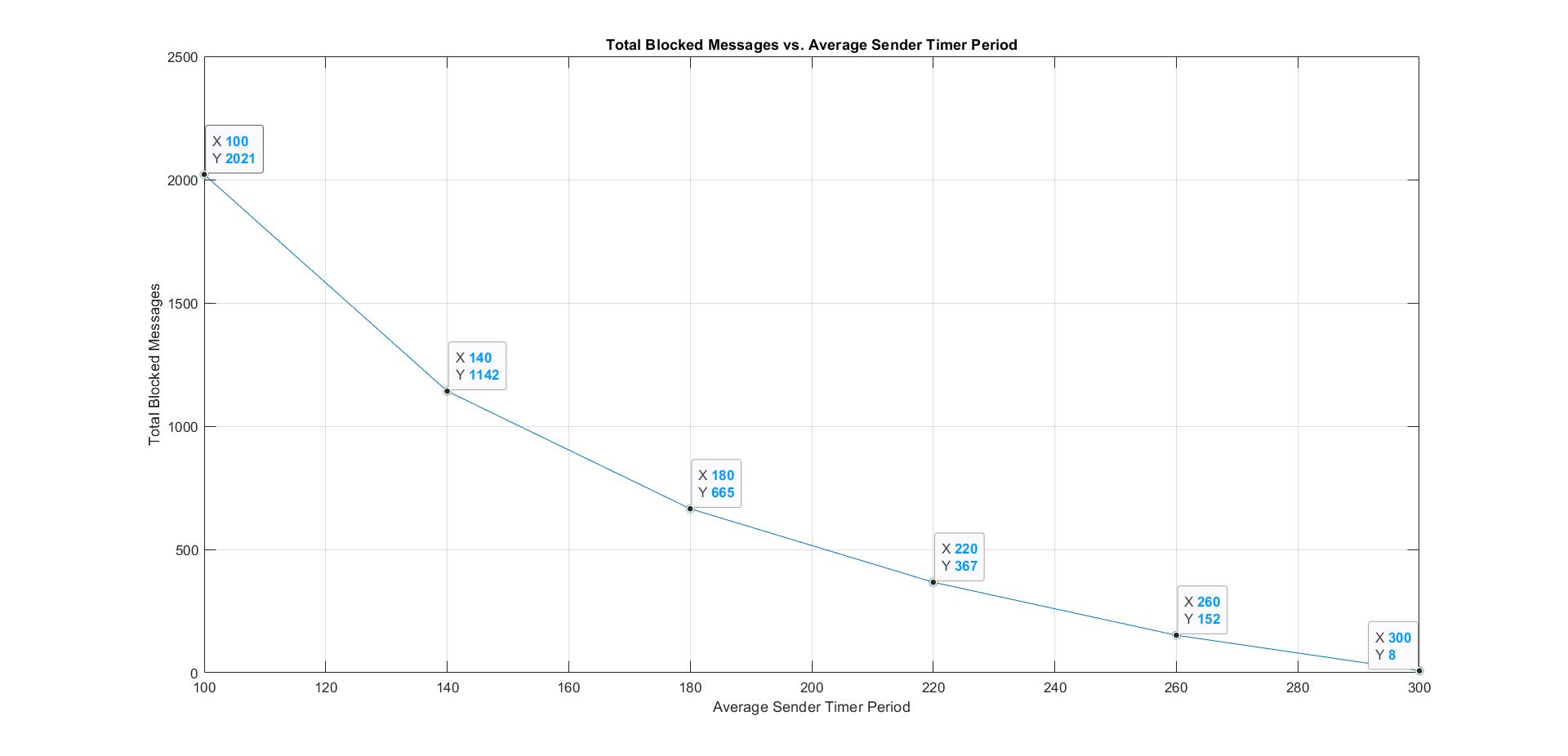
****

Figure : total blocked msgs VS avg Sender period for queue=3

For queue of size = 10:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sender 1 | | Sender 2 | | Sender 3 | | Total transmitted messages | Total blocked messages |
| Iteration | Sent messages | Blocked messages | Sent messages | Blocked messages | Sent messages | Blocked messages | \_\_\_\_\_ | \_\_\_\_\_ |
| 1 | 360 | 648 | 320 | 689 | 329 | 671 | 1009 | 2008 |
| 2 | 342 | 378 | 342 | 379 | 325 | 397 | 1009 | 1154 |
| 3 | 345 | 214 | 336 | 223 | 328 | 225 | 1009 | 662 |
| 4 | 336 | 122 | 328 | 133 | 345 | 115 | 1009 | 370 |
| 5 | 340 | 45 | 336 | 48 | 333 | 43 | 1009 | 136 |
| 6 | 334 | 0 | 329 | 0 | 341 | 0 | 1004 | 0 |

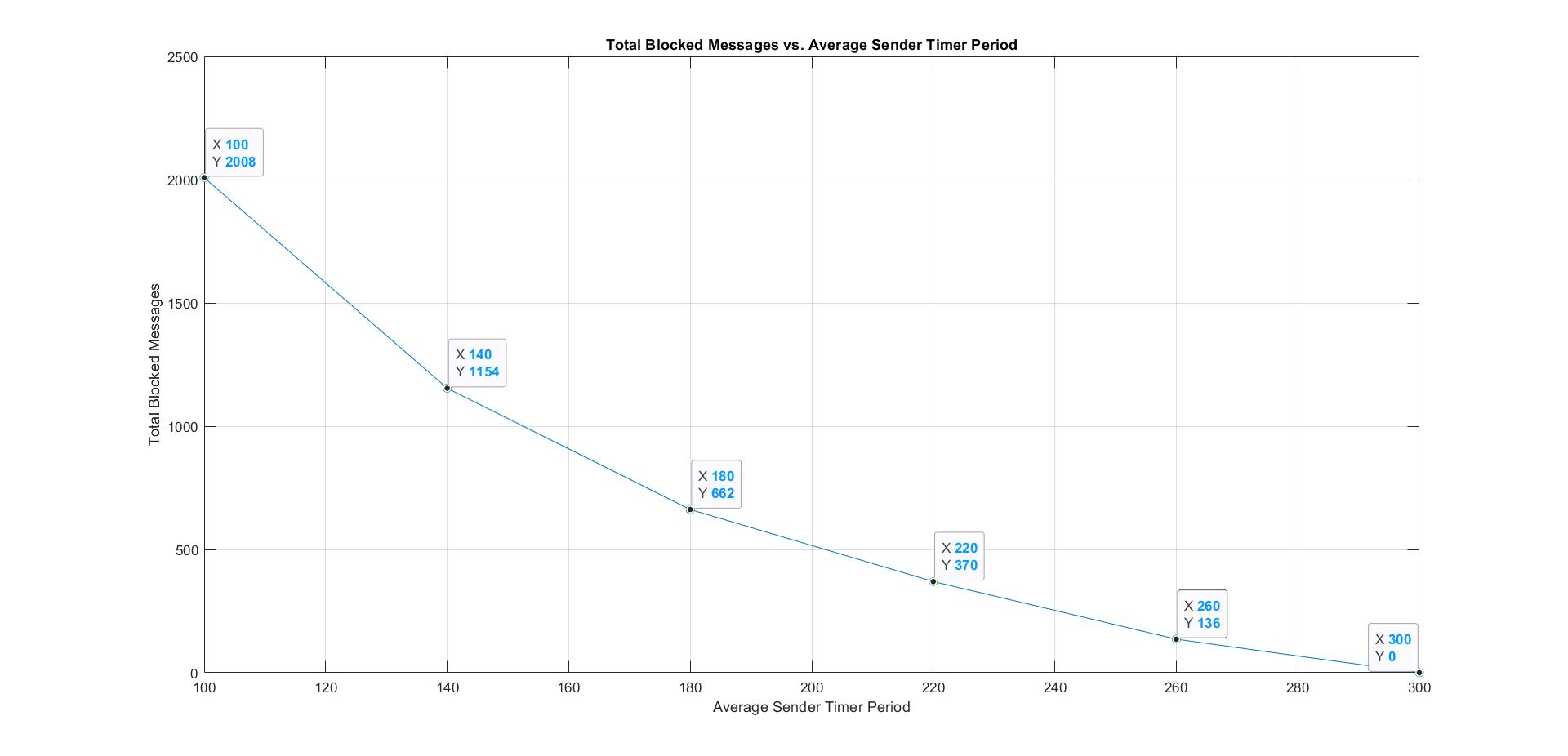
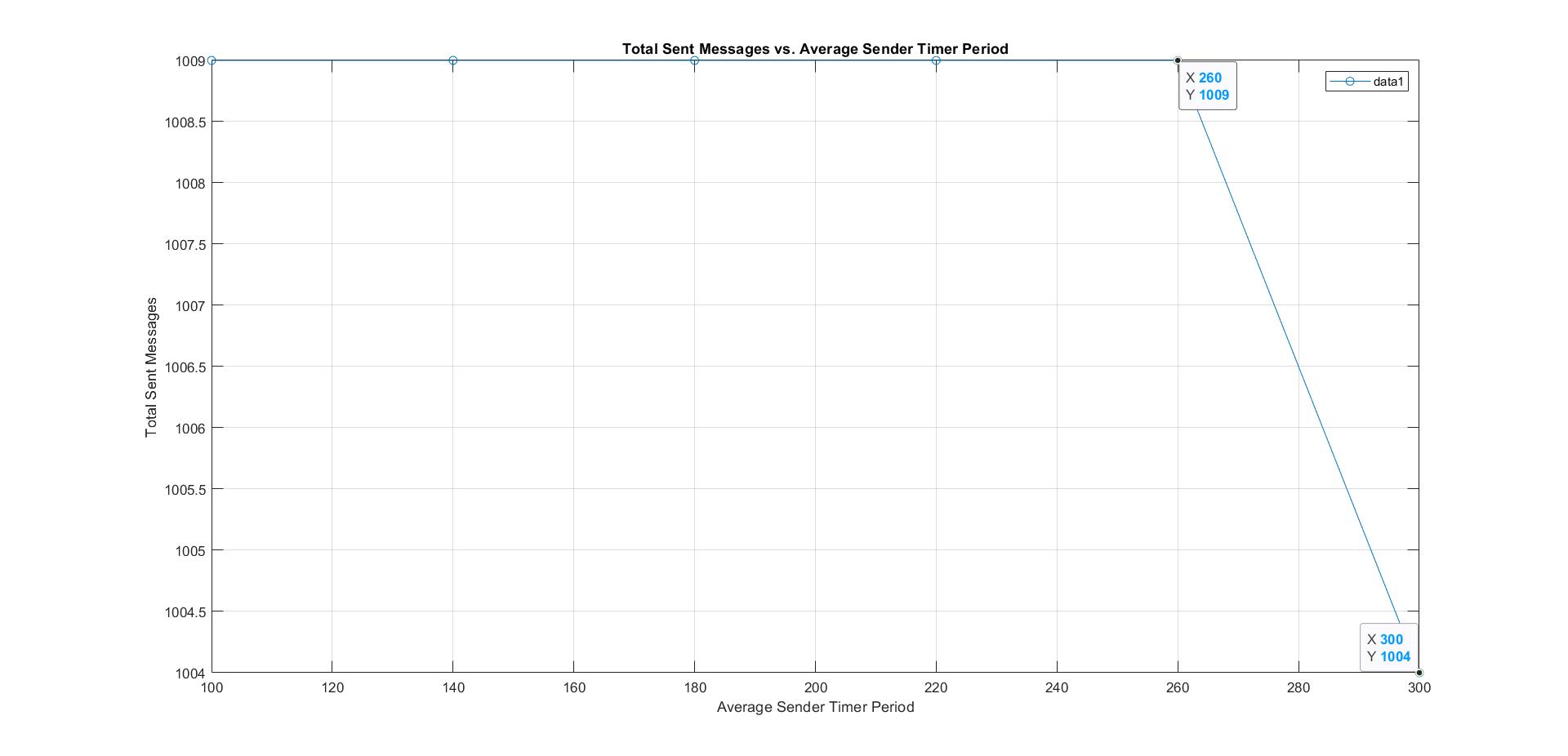
****

Figure : total transmitted messages VS avg sender period for queue=10

Figure : total blocked messages VS avg sender period for queue=10

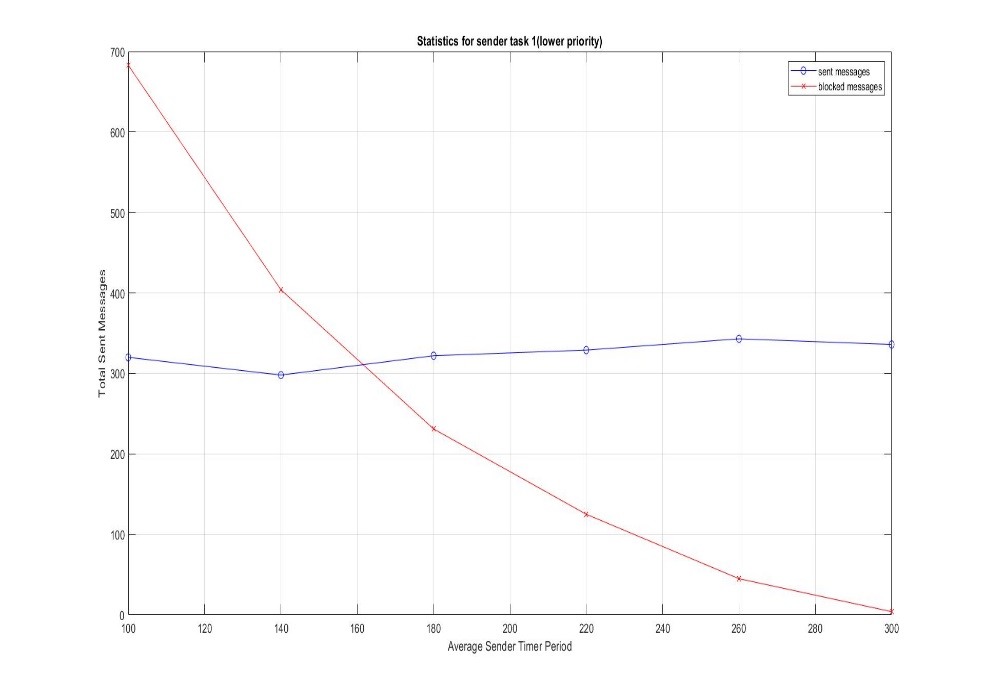
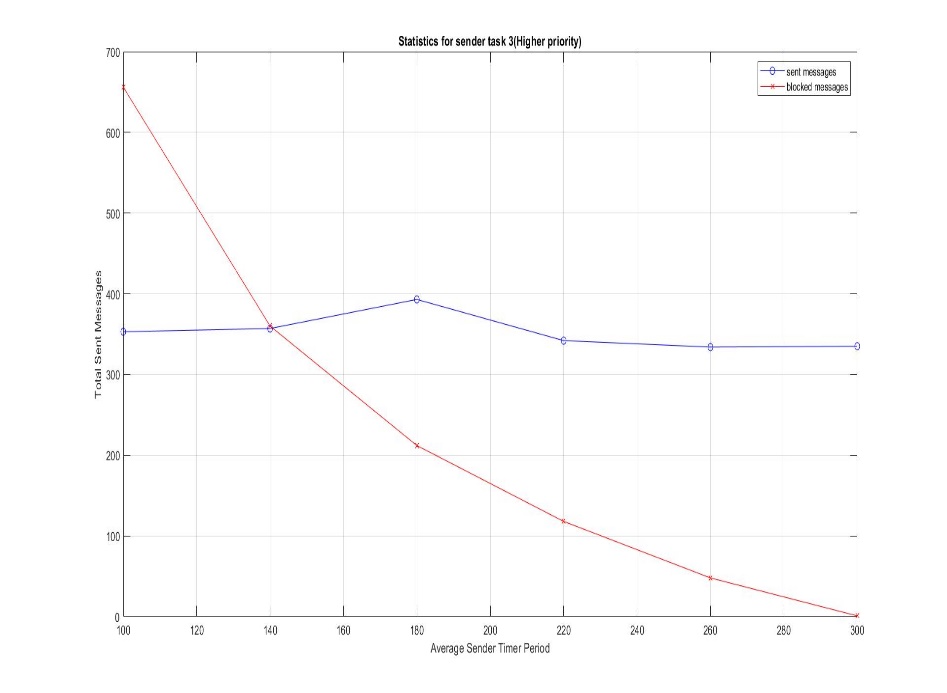


Figure :statistics for low priority task VS average sender timer period

Figure :statistics for low priority task VS average sender timer period

1. **Conclusion**

* We observed when the range between the lower and upper bounds of the random period in Sender Tasks become larger (the rate of the process becomes slower) the total blocked messages remarkably decreased.
* And the reason behind the gap between the total Transmitted messages (like 1002) and the received (1000) is that the sender tasks sent messages successfully before the system figures out (the callback function is called) that it reached its specified 1000 messages for the iteration, so the number of messages in gap <= (the size of the queue). And the other gap between the total sent messages (blocked + transmitted) like why do we need to send like 3023 messages to receive 1000? Because the rate of sending process is faster than the receiver specifically in the early iterations and also the receiver receives just one message per timer period.
* When we repeated our system for a queue of size 10, we observed that the total blocked messages slightly decreased compared to the one of size 3, it actually vanished in the last iteration as it became zero.