ELC 2080
RTOS Communicating Tasks Project

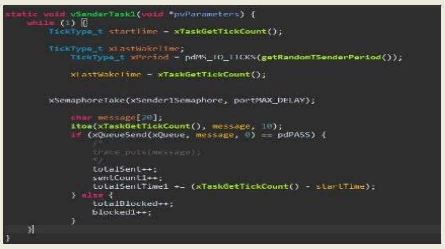
BN	Section	Student ID	اسم الطالب
37	**4**	**9220974**	**يوسف البدرى هارون محمد**
45	**4**	**9221008**	**يوسف محمد سعيد محمد**

1-System Design

- We planned to produce a consistent chance of obtaining any value between two boundaries. For every task, we built a callback function and a timer. The tasks have semaphore take operations, which cause them to block until the timer runs out and the callback function releases the semaphore, whereas the callback function includes a semaphore release operation. Although we considered creating four semaphores for each task. There are three sender tasks; task number three is of higher priority than the other two, which are of equal Priority. The receiver task has a higher priority than the sender tasks and includes a timer and a callback function with a 100 ms period.

- Messages are sent and received from a common queue that has a fixed size of three spots initially, and ten spots later. The task's sleep time is defined as the amount of time the timer takes to fire the callback function. Upon the startup of the callback function, the task proceeds based to its priority and is submitted to the scheduler. We put in place a reset function that resets sent, received, and blocked message counters. Additionally, it determines if the array limit for period boundaries has been reached. The program ends with "Game Over" message and closes the kit if this limit is reached.

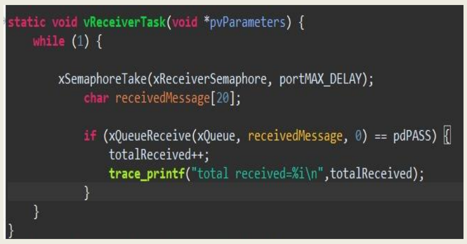
Sender Task



```
static void vSenderTask1(void *pvParameters) {
    while (1) {
        TickType_t startTime = xTaskGetTickCount();

        TickType_t xLastWakeTime;
            TickType_t xPeriod = pdMS_TO_TICKS(getRandomTSenderPeriod());

            xLastWakeTime = xTaskGetTickCount();


        xSemaphoreTake(xSender1Semaphore, portMAX_DELAY);

            char message[20];
            itoa(xTaskGetTickCount(), message, 10);
            if (xQueueSend(xQueue, message, 0) == pdPASS) {
                /*
                trace_puts(message);
                */
                totalSent++;
                sentCount1++;
                totalSentTime1 += (xTaskGetTickCount() - startTime);
            } else {
                totalBlocked++;
                blocked1++;
            }
    }
}
```

- **To guarantee that we add the difference time to calculate the average sent time**
- **Update all Task1 counters**

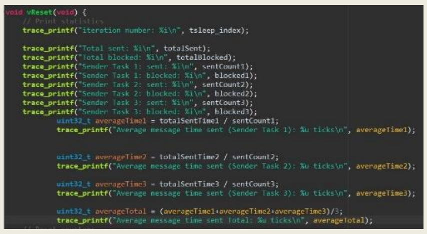
Receiver Task



```
static void vReceiverTask(void *pvParameters) {
    while (1) {

        xSemaphoreTake(xReceiverSemaphore, portMAX_DELAY);
            char receivedMessage[20];

            if (xQueueReceive(xQueue, receivedMessage, 0) == pdPASS) {
                totalReceived++;
                trace_printf("total received=%i\n",totalReceived);
            }
    }
}
```

We print the total received to make sure that the sending and receiving process is in progress without errors

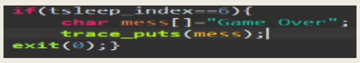
Reset Function



```
void vReset(void) {
    // Print statistics
    trace_printf("iteration number: %i\n", tsleep_index);

    trace_printf("Total sent: %i\n", totalSent);
    trace_printf("Total blocked: %i\n", totalBlocked);
    trace_printf("Sender Task 1: sent: %i\n", sentCount1);
    trace_printf("Sender Task 1: blocked: %i\n", blocked1);
    trace_printf("Sender Task 2: sent: %i\n", sentCount2);
    trace_printf("Sender Task 2: blocked: %i\n", blocked2);
    trace_printf("Sender Task 3: sent: %i\n", sentCount3);
    trace_printf("Sender Task 3: blocked: %i\n", blocked3);
            uint32_t averageTime1 = totalSentTime1 / sentCount1;
            trace_printf("Average message time sent (Sender Task 1): %u ticks\n", averageTime1);


            uint32_t averageTime2 = totalSentTime2 / sentCount2;
            trace_printf("Average message time sent (Sender Task 2): %u ticks\n", averageTime2);

            uint32_t averageTime3 = totalSentTime3 / sentCount3;
            trace_printf("Average message time sent (Sender Task 3): %u ticks\n", averageTime3);

            uint32_t averageTotal = (averageTime1+averageTime2+averageTime3)/3;
            trace_printf("Average message time sent Total: %u ticks\n", averageTotal);
```

To show the statistics and then we reset the all counters to be ready to the next iteration

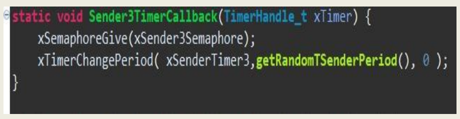
---



```
if(tsleep_index==6){
        char mess[]="Game Over";
        trace_puts(mess);
exit(0);}
```

to stop the program after end all iterations

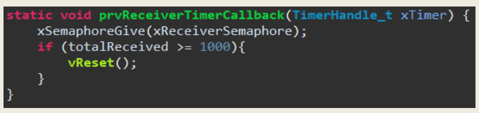
Sender Callback Function



```
static void Sender3TimerCallback(TimerHandle_t xTimer) {
    xSemaphoreGive(xSender3Semaphore);
    xTimerChangePeriod( xSenderTimer3,getRandomTSenderPeriod(), 0 );
}
```

To make sure the timer updates the period to confirm that we use random period after each call

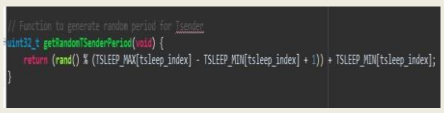
Receiver Callback Function



```
static void prvReceiverTimerCallback(TimerHandle_t xTimer) {
    xSemaphoreGive(xReceiverSemaphore);
    if (totalReceived >= 1000){
        vReset();
    }
}
```

To stop after 1000 messages to move to next iteration

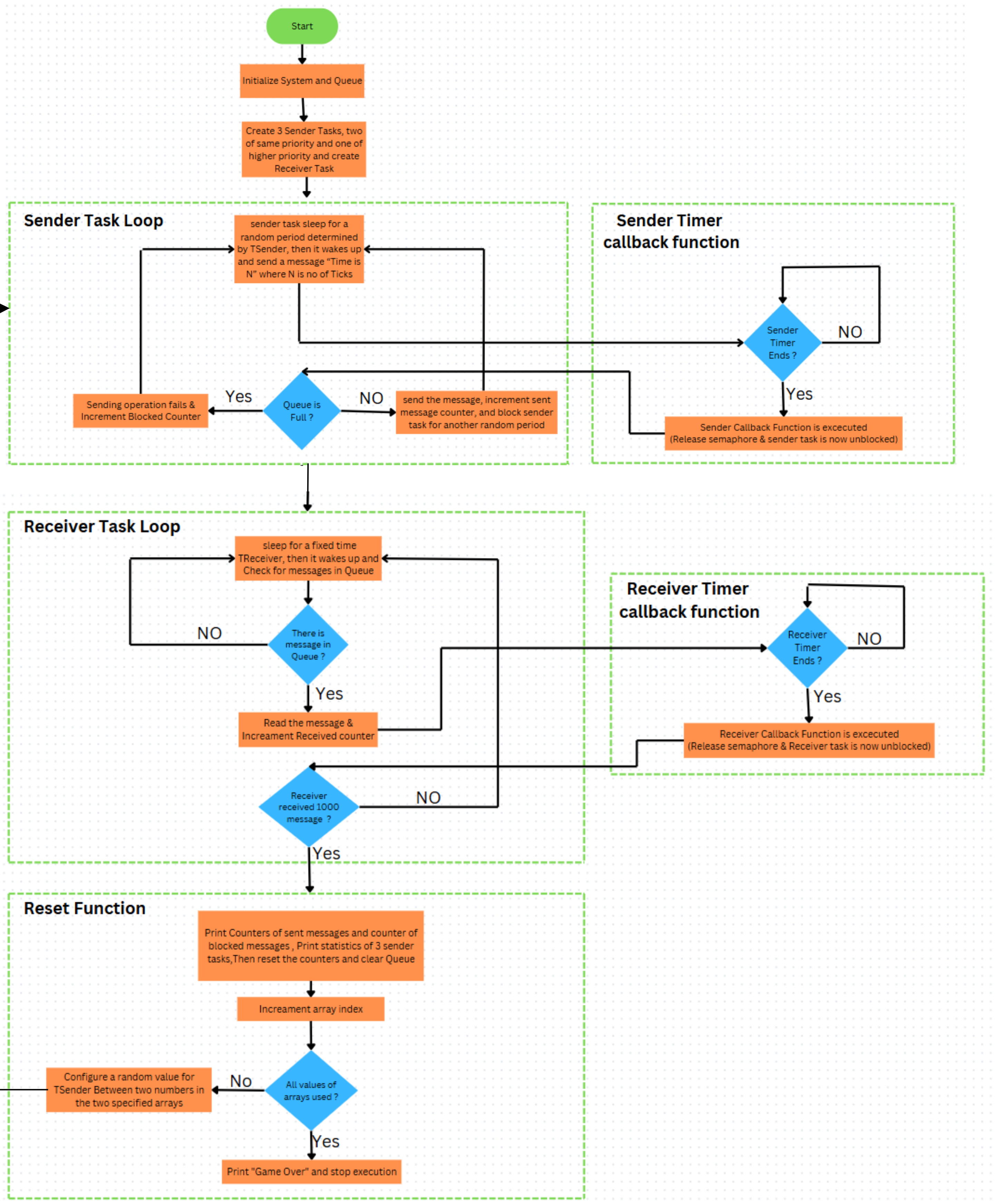
Random Period



```
// Function to generate random period for Tsender
uint32_t getRandomTSenderPeriod(void) {
    return (rand() % (TSLEEP_MAX[tsleep_index] - TSLEEP_MIN[tsleep_index] + 1)) + TSLEEP_MIN[tsleep_index];
}
```

To get a random value between the determined ranges

Design Flow: Link here


Start
Initialize System and Queue
Create 3 Sender Tasks, two of same priority and one of higher priority and create Receiver Task
Sender Task Loop
sender task sleep for a random period determined by TSender, then it wakes up and send a message "Time is N" where N is no of Ticks
Queue is Full ?
Yes
Sending operation fails & Increment Blocked Counter
NO
send the message, increment sent message counter, and block sender task for another random period
Sender Timer callback function
Sender Timer Ends ?
NO
Yes
Sender Callback Function is executed (Release semaphore & sender task is now unblocked)
Receiver Task Loop
sleep for a fixed time TReceiver, then it wakes up and Check for messages in Queue
There is message in Queue ?
NO
Yes
Read the message & Increament Received counter
Receiver received 1000 message ?
NO
Yes
Receiver Timer callback function
Receiver Timer Ends ?
NO
Yes
Receiver Callback Function is excecuted (Release semaphore & Receiver task is now unblocked)
Reset Function
Print Counters of sent messages and counter of blocked messages , Print statistics of 3 sender tasks,Then reset the counters and clear Queue
Increament array index
All values of arrays used ?
No
Configure a random value for TSender Between two numbers in the two specified arrays
Yes
Print "Game Over" and stop execution

2-Results and discussion

Queue of size 3

Iteration	0	1	2	3	4	5
Sender 1	Sent (321) Blocked (660) Avg time (103)	Sent (346) Blocked (359) Avg time (143)	Sent (304) Blocked (251) Avg time (182)	Sent (341) Blocked (110) Avg time (223)	Sent (336) Blocked (54) Avg time (259)	Sent (329) Blocked (5) Avg time (302)
Sender 2	Sent (360) Blocked (644) Avg time (100)	Sent (324) Blocked (392) Avg time (140)	Sent (342) Blocked (223) Avg time (181)	Sent (339) Blocked (116) Avg time (224)	Sent (333) Blocked (53) Avg time (262)	Sent (338) Blocked (6) Avg time (294)
Sender 3	Sent (321) Blocked (660) Avg time (103)	Sent (332) Blocked (383) Avg time (140)	Sent (356) Blocked (202) Avg time (180)	Sent (322) Blocked (125) Avg time (228)	Sent (333) Blocked (52) Avg time (262)	Sent (334) Blocked (5) Avg time (297)

Queue of size 10

Iteration	0	1	2	3	4	5
Sender 1	Sent (329) Blocked (689) Avg time (100)	Sent (339) Blocked (378) Avg time (140)	Sent (319) Blocked (235) Avg time (180)	Sent (334) Blocked (115) Avg time (226)	Sent (323) Blocked (60) Avg time (262)	Sent (338) Blocked (0) Avg time (298)
Sender 2	Sent (333) Blocked (660) Avg time (100)	Sent (312) Blocked (409) Avg time (140)	Sent (343) Blocked (207) Avg time (181)	Sent (333) Blocked (117) Avg time (225)	Sent (353) Blocked (35) Avg time (259)	Sent (335) Blocked (0) Avg time (300)
Sender 3	Sent (347) Blocked (662) Avg time (99)	Sent (358) Blocked (352) Avg time (142)	Sent (347) Blocked (210) Avg time (180)	Sent (342) Blocked (115) Avg time (222)	Sent (333) Blocked (51) Avg time (261)	Sent (333) Blocked (1) Avg time (302)

- From the above tables we can realize that the larger the period we have, the less number of messages is blocked. We have lower bound which is increased by 30 ms and higher bound which is increased by 50 ms so we have difference about 20 ms from each interval and each previous one.

- The maximum number of sent messages in a queue of size three is 1002, because when the receiver receives 1000 messages, there will be two empty seats in the queue, allowing two senders to send messages at that time.

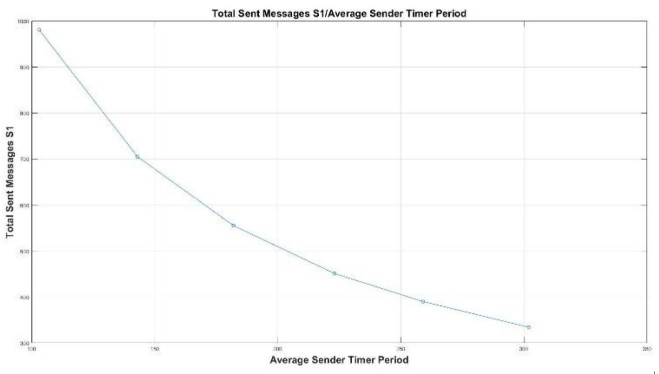
- In a queue of size ten, the maximum number of sent messages is 1009, because when the receiver receives 1000 messages, there will be nine empty seats in the queue, allowing each sender to send three more messages.

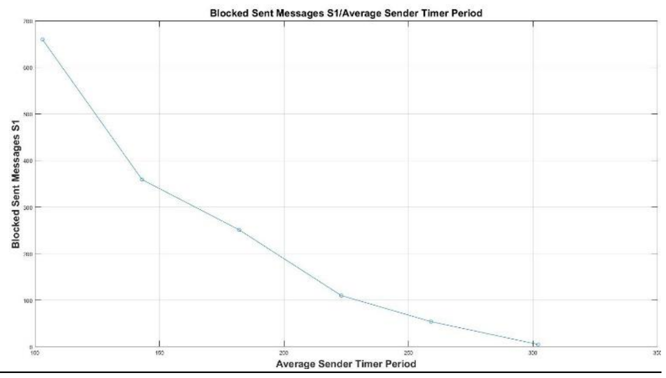
- And as we can see, all senders with queue sizes of 3 and 10 have roughly the same amount of messages sent. This is because each of the three senders has a nearly equal random period, although it varies by roughly 5 ms.

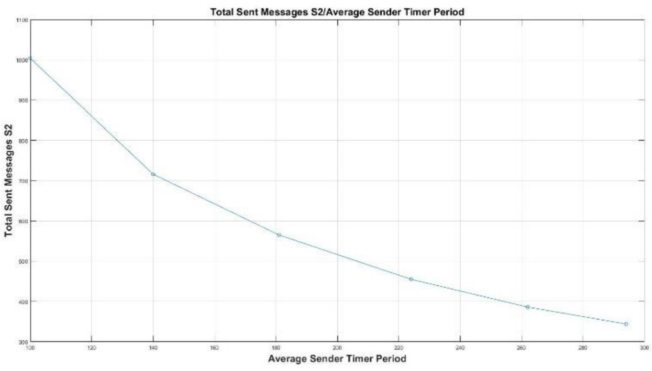
- The same is true for the messages that are blocked.

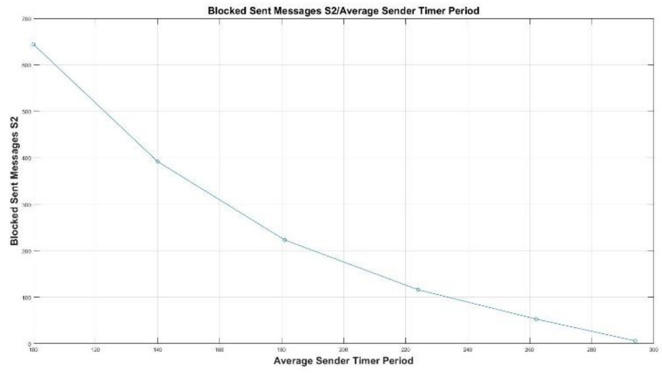
Graphs:

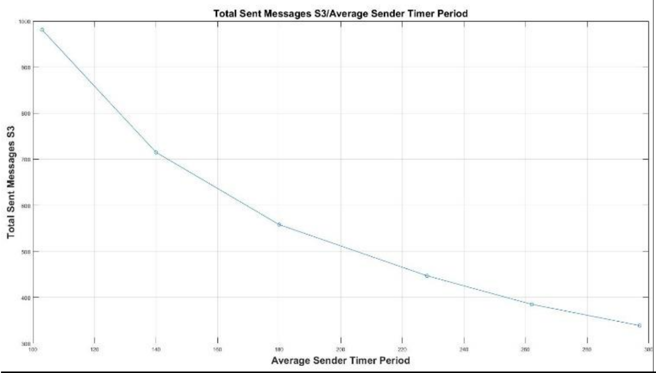
Queue of size 3

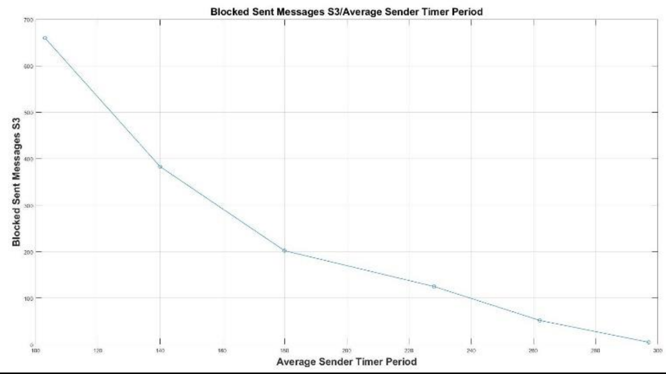












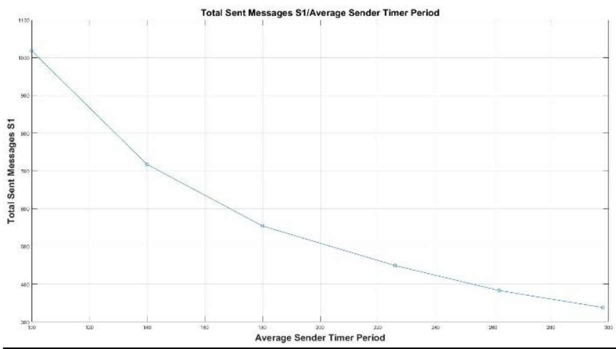
Explain the gap between the number of sent and received messages in the running period.

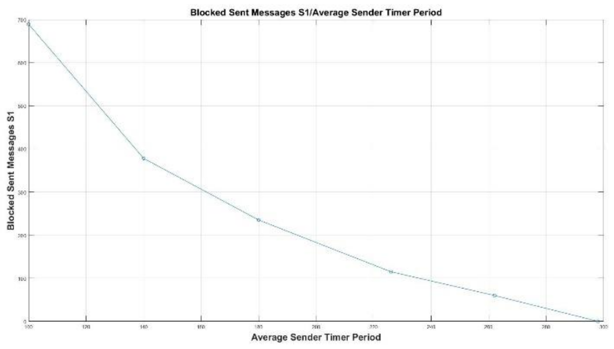
- A small **queue size** causes it to fill up quickly, blocking messages because there isn't enough space to hold them. This results in more messages being sent than being received in a given amount of time.

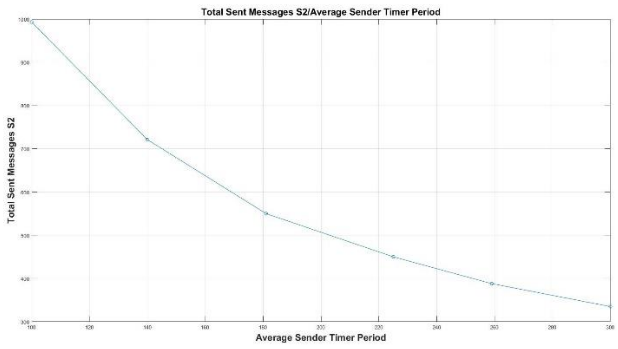
- **Prioritization and timers** used by each task to control the sending and receiving of messages. Sending and receiving messages in perfect timing can be challenging due to the random periods that are generated for each task. There may be a brief pause because higher priority tasks are more likely to send messages first.

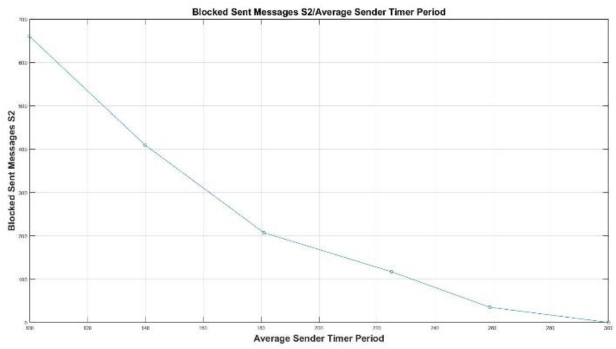
- The gap gets smaller over **multiple iterations** as the queue size grows. Greater queue size results in fewer blocked messages and a shorter time interval between sent and received messages because there is more space for incoming messages.

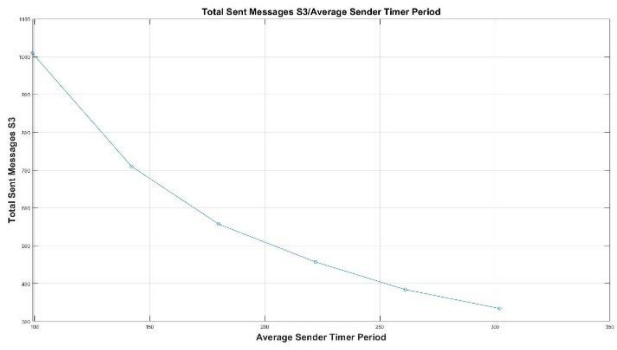
Queue of size 10

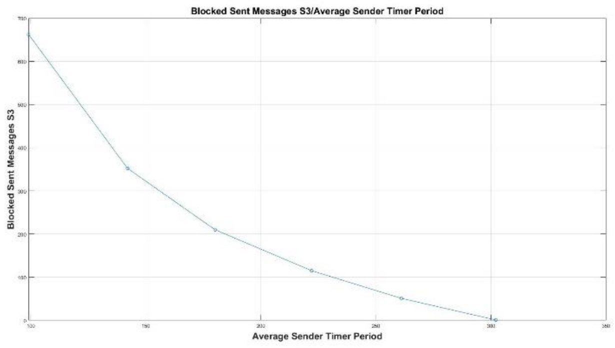




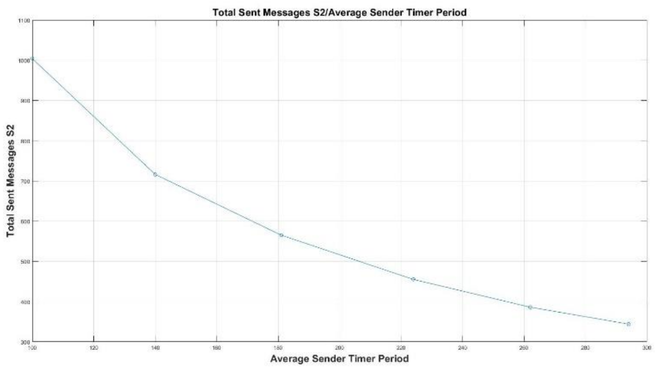


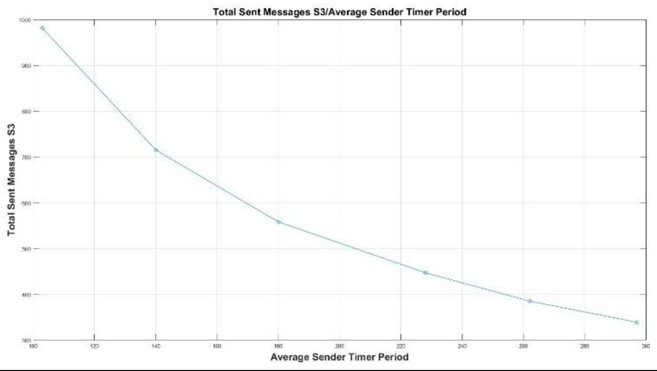






Comparasion between task of higher and lower priority (Queue of size 3)





- As we can see, they don't usually meet because of the random value of the period, but when they do, it's for a task with a higher priority.

3-Refrences

https://www.freertos.org/