Embedded system project

RTOS Project

Dr. Khaled fouad

BN	Sec	Student ID	اسم الطالب
21	1	9230178	احمد محمد محمد عبدالحميد موسى
17	1	9230169	احمد محمد حنفى ريحان

1 System Design

This project simulates a network communication system using FreeRTOS to model the behavior of senders, receivers, and a central switch managing a lossy communication link. The simulation aims to evaluate the performance of the Go-Back-N protocol under varying network conditions.

The system consists of two sender nodes (Node 1, Node 2) and two receiver nodes (Node 3, Node 4). Senders generate data packets destined for a randomly chosen receiver. These packets traverse a switch node which introduces propagation delay, transmission delay based on packet size and link capacity, and simulates packet loss.

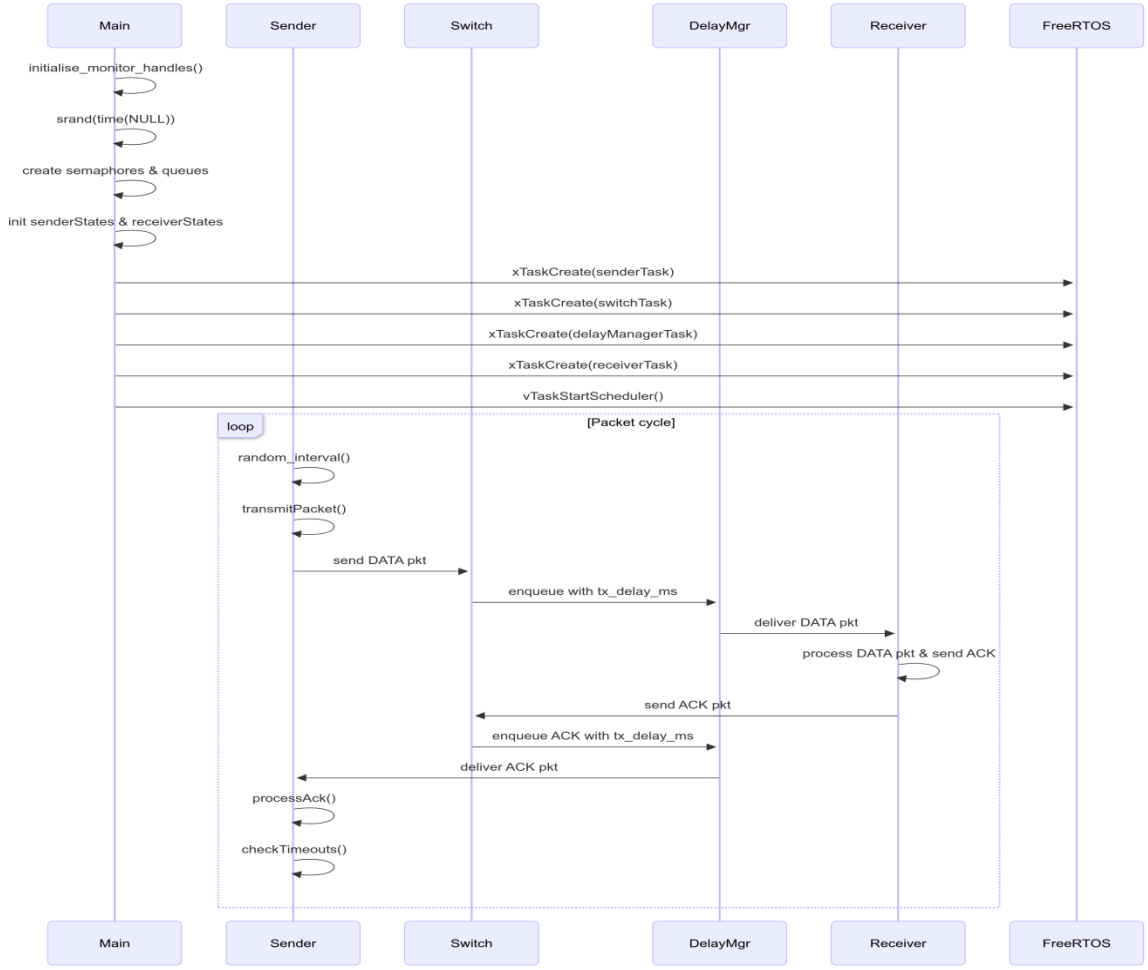

Figure 1 : Message Sequence showing Tasks interaction

1.1 Packet Structure

Packets are dynamically allocated and contain a header and payload. The header includes:

- ✓ Sender ID (1 byte)
- ✓ Destination ID (1 byte)
- ✓ Sequence Number (4 bytes)
- ✓ Packet Length (2 bytes)
- ✓ Packet Type (1 byte: 0 for Data, 1 for ACK)

The payload size L varies randomly between L1 (500 bytes) and L2 (1500 bytes). Acknowledgement (ACK) packets have a fixed size K (40 bytes).

1.2 RTOS Implementation

The system utilizes FreeRTOS components to manage concurrency and communication:

1.2.1 Tasks:- Separate tasks model each sender, each receiver, the switch, and a delay manager.

a. **Sender Task :-** Generates packets at random intervals (T1 to T2 ms), manages a transmission buffer for each destination receiver, handles ACKs, detects timeouts (Tout), and implements the Go-Back-N retransmission logic (window size N , max retransmissions) and S&W Protocol (if N equals 1) .

b. **Receiver Task :-** Waits for packets on its dedicated queue, checks sequence numbers, sends ACKs for in-order packets via the switch, and discards out-of order packets (as per Go-Back-N). It tracks received packets and total bytes.

c. **Switch Task :-** Receives packets from senders and ACKs from receivers via a central switchQueue . It simulates packet drops based on probabilities P_drop (for data) and P_ack (for ACKs). Non-dropped packets are forwarded to the delayManagerTask .

d. **Delay Manager Task :-** Receives packets from the switch via delay Queue . It Holds packets for the calculated total delay (propagation D + transmission (L*8)/C) before forwarding them to the appropriate receiver queue (receiver Queues) or sender ACK queue (ack Queues).

1.2.2 Queues:- Used for inter-task communication:

a. **Switch Queue :-** Central queue for all packets entering the switch.

b. **Delay Queue :-** Holds packets undergoing simulated delay.

c. **Receiver Queues [NUM_RECEIVERS] :-** Dedicated queue for each receiver.

d. **Ack Queues [NUM_SENDERS] :-** Dedicated queue for ACKs destined for each sender.

1.2.3 Semaphores:-

a. **buffer_mutex :-** Protects access to the sender's transmission buffer andstate variables.

b. **stats_mutex :-** Protects access to global statistics.

c. **Termination Semaphore :-** Signals the end of the simulation (when 2000 packets are received for each receiver).

1.3 Protocols

1.3.1 Send and wait protocol (S&W)

The Send-and-Wait (S&W) protocol is a simple Automatic Repeat request (ARQ) scheme in which the sender transmits one packet and then pauses its transmission until it receives an acknowledgment (ACK) for that packet. This approach ensures reliable data delivery over lossy links by retransmitting packets when ACKs are lost or corrupted.(special case of Go-Back-N with N = 1).

1.3.2 Go-Back-N Protocol

The simulation models the Go-Back-N protocol with a selectable window size (1, 2, 4, or 8). The sender can have up to N unacknowledged packets in flight and uses cumulative ACKs—an ACK for packet i confirms all packets up to i. If the base packet's timer expires, the sender retransmits the entire window, and any packet exceeding four retransmissions is dropped. The receiver only accepts in-order packets, discarding any that arrive out of sequence.

1.4 Simulation Flow

1.4.1 Initialization

- ✓ The system sets up tasks, queues, and semaphores, and initializes the state of senders and receivers.

1.4.2 Packet Handling

- ✓ **Generation**: Senders create packets with random intervals, sequence numbers, and destinations.
- ✓ **Transmission**: Senders attempt to send packets if their window isn't full, starting timers and forwarding packets to the switch queue.

1.4.3 Switch & Delay

- ✓ The switch processes packets, applying a drop probability, and forwards non-dropped packets to the delay manager.
- ✓ The delay manager holds packets for a calculated delay before sending them to the receiver's queue or sender's ACK queue.

1.4.4 Receiver & ACK

- ✓ Receivers accept in-order packets, count them, and generate ACKs. Out-of-order packets are discarded.
- ✓ ACKs return to the sender, updating the window and allowing new transmissions.

1.4.5 Timeout & Retransmission

- ✓ Expired timers trigger retransmission of the base packet in the sender's window, with packets dropped after max retries.

1.4.6 Termination

- ✓ The simulation ends once 2000 packets are successfully received by each receiver.

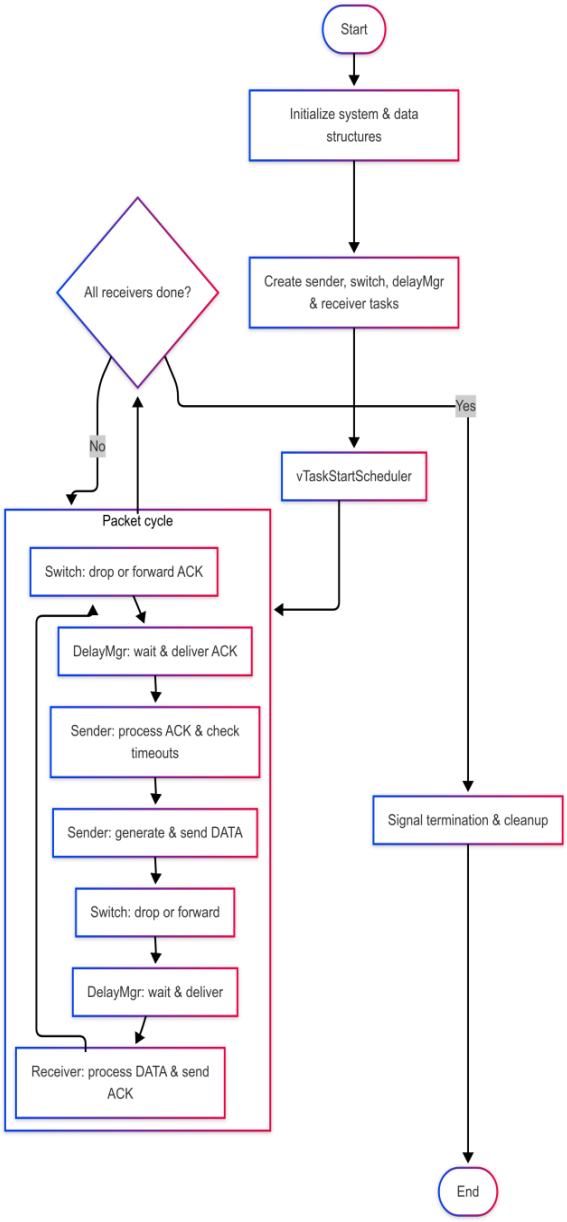

Figure 2 : Simulation Flow

2 Results and Discussion

The simulation evaluated the performance of the S&W and Go-Back-N protocols by measuring the throughput (in KBytes/sec) required to successfully receive 2000 packets for each receiver under various network conditions as shown in Figure 3, specifically varying packet drop probability (P_drop), retransmission timeout (Tout), and Go-Back-N window size (N).

```
================= SIMULATION COMPLETE =================
All receivers have received 2000 packets each
Simulation time: 172408 ms
P_DROP: 0.040
Timeout (TOUT_MS): 175 ms
Window Size (DEFAULT_WINDOW_SIZE): 1
Total packets received: 4011
Sender 1: Generated=2008, Transmitted=2008, Retransmitted=107, ACKs=2006
Sender 2: Generated=2005, Transmitted=2005, Retransmitted=105, ACKs=2003
Receiver 3: Received=2011, Out-of-order=0, Duplicate=25, ACKs sent=2036
  - From Sender 1: 1008 packets
  - From Sender 2: 1003 packets
Receiver 4: Received=2000, Out-of-order=0, Duplicate=22, ACKs sent=2022
  - From Sender 1: 999 packets
  - From Sender 2: 1001 packets
Total data bytes received: 1995542 bytes
Effective throughput: 11574 bytes/sec
Effective throughput: 11 kbytes/sec
=======================================================
[SWITCH] Final Stats - Total: 8283, Data Dropped: 165, ACK Dropped: 47
[button:reset up]Graphic window closed. Quit.
```

Figure 3 : Example of Simulation output for (S&W)

2.1 the average number of transmissions of a packet as function of Pdrop E (Figure 4)

$E = \frac{Total\ num\ of\ tansmissions}{Total\ num\ of\ packets}$ for each value of P_drop

so for P ={0.01 ,0.02 ,0.04 ,0.08} E = {1.01544, 1.02866, 1.0535, 1.09025}

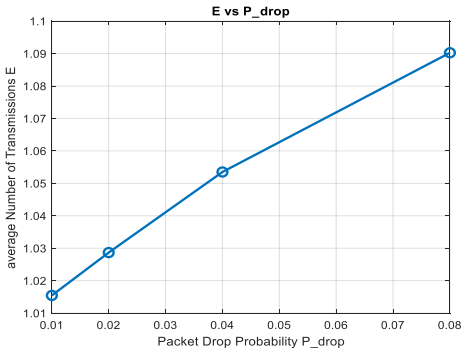

Figure 4 average Number of transmission

2.2 packets dropped due to being transmitted more than 4 times.

After simulations with different values of P_drop packets dropped due to being transmitted more than 4 Times equal zero .

2.3 Throughput Analysis

The primary performance metric measured was throughput, defined as the total bytes of Successfully received unique packets divided by the total simulation time.

2.3.1 Impact of Window Size (N)

The Go-Back-N protocol aims to improve upon the basic Send-and-Wait (N=1) by allowing multiple packets to be outstanding. The table below summarizes the throughput achieved for different window sizes (N), drop probabilities (P_drop), and timeouts (T, equivalent to Tout in ms).

Table 1 : P_drop , T_out , N Index and Corresponding Throughput in Kbyte

window sizes (N)	1 (S&W Protocol)	2	4	8
P_drop=0.02 , T=150	11.863 Kbyte	12.729 Kbyte	12.760 Kbyte	12.827 Kbyte
P_drop=0.02,T=175	11.795 Kbyte	12.678 Kbyte	12.874 Kbyte	12.936 Kbyte
P_drop=0.02,T=200	11.815 Kbyte	12.738 Kbyte	12.878 Kbyte	12.994 Kbyte
P_drop=0.04,T=150	11.579 Kbyte	12.815 Kbyte	12.884 Kbyte	13.026 Kbyte
P_drop=0.04,T=175	11.574 Kbyte	12.656 Kbyte	12.872 Kbyte	12.918 Kbyte
P_drop=0.04,T=200	11.524 Kbyte	12.588 Kbyte	12.854 Kbyte	13.035 Kbyte
P_drop=0.08,T=150	11.099 Kbyte	12.702 Kbyte	12.754 Kbyte	12.970 Kbyte
P_drop=0.08,T=175	10.974 Kbyte	12.515 Kbyte	12.908 Kbyte	13.019 Kbyte
P_drop=0.08,T=200	10.834 Kbyte	12.159 Kbyte	12.754 Kbyte	13.240 Kbyte

2.3.2 Impact of Packet Drop Probability (P_drop)

The probability of packet loss significantly impacts performance. Figure 5 illustrates the relationship between throughput and P_drop for different timeout values (for N = {1, 2, 4, 8}).(N = 1 is the S&W protocol)

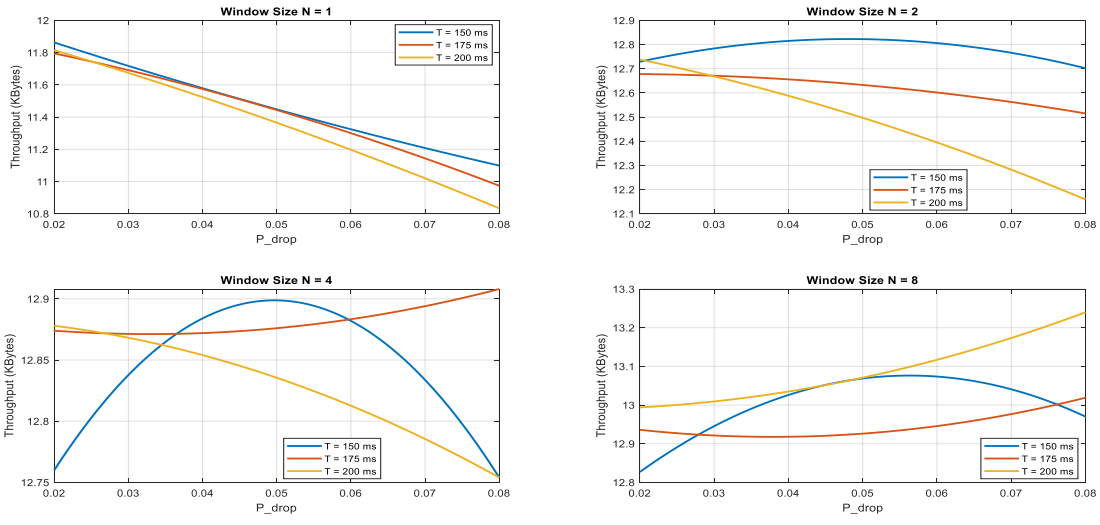

Figure 5 : throughput as a function of Pdrop for different values of timeout period Tout

2.3.3 Impact of Timeout (Tout)

The probability of packet loss significantly impacts performance. Figure 6 illustrates the relationship between throughput as a function of Tout for different values of Pdrop values (for N = {1 ,2 ,4 ,8}).

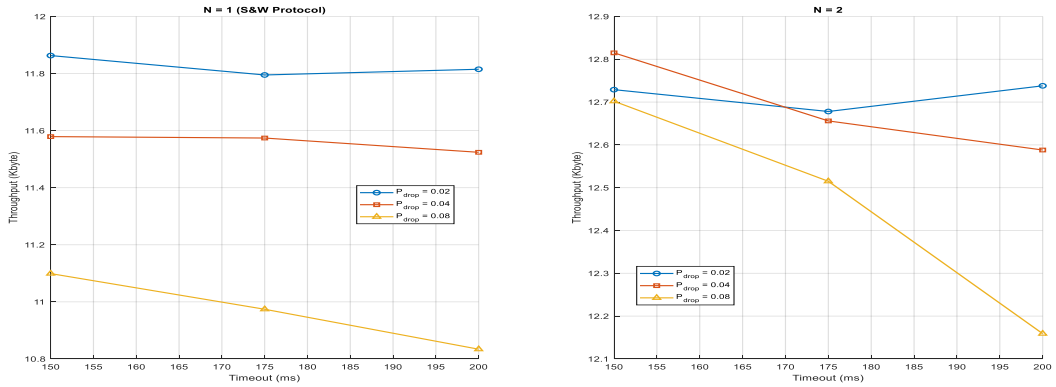

Figure 6 : throughput as a function of Tout for different values of Pdrop

3 References

[1] Mastering-the-FreeRTOS-Real-Time-Kernel.v1.1.0.

4 Code Snippets

```
int main(void) {
    initialise_monitor_handles();
    srand(time(NULL));
    simulation_start_time = xTaskGetTickCount();
    terminationSemaphore = xSemaphoreCreateBinary();
    stats_mutex = xSemaphoreCreateMutex();
    switchQueue = xQueueCreate(SWITCH_QUEUE_SIZE, sizeof(Packet));
    delayQueue = xQueueCreate(DELAY_QUEUE_SIZE, sizeof(DelayQueueItem));
    for (int i = 0; i < NUM_RECEIVERS; i++) {
        receiverQueues[i] = xQueueCreate(RECEIVER_QUEUE_SIZE, sizeof(Packet)); }
    for (int i = 0; i < NUM_SENDERS; i++) {
        ackQueues[i] = xQueueCreate(ACK_QUEUE_SIZE, sizeof(Packet));}
    for (int i = 0; i < NUM_SENDERS; i++) {
        senderStates[i].buffer_mutex = xSemaphoreCreateMutex();
        senderStates[i].window_size = DEFAULT_WINDOW_SIZE;
        for (int j = 0; j < NUM_RECEIVERS; j++) {
            senderStates[i].next_seq_num[j] = 0;
            senderStates[i].base_seq_num[j] = 0;} }
xTaskCreate(senderTask,"Sender1",SENDER_TASK_STACK_SIZE,(void*)1,SENDER_TASK_PRIORITY,NULL);
xTaskCreate(senderTask,"Sender2", SENDER_TASK_STACK_SIZE, (void*)2, SENDER_TASK_PRIORITY,NULL);
xTaskCreate(switchTask,"Switch",SWITCH_TASK_STACK_SIZE,NULL,SWITCH_TASK_PRIORITY, NULL);
xTaskCreate(delayManagerTask"DelayMgr",DELAYMGR_TASK_STACK_SIZE,NULL,DELAY_MGR_TASK_PRIORITY,NULL);
xTaskCreate(receiverTask,"Receiver3",RECEIVER_TASK_STACK_SIZE,void*)0,RECEIVER_TASK_PRIORITY,NULL);
xTaskCreate(receiverTask,"Receiver4",RECEIVER_TASK_STACK_SIZE(void*)1,RECEIVER_TASK_PRIORITY,NULL);
vTaskStartScheduler();
    return 0;}
```