BN	Section	Student ID	اسم الطالب
25	**3**	**9230666**	**كريم حسن عاطف على**
23	**3**	**9230657**	**فرح محمد عبد العزيز الساعي**

1 System Design

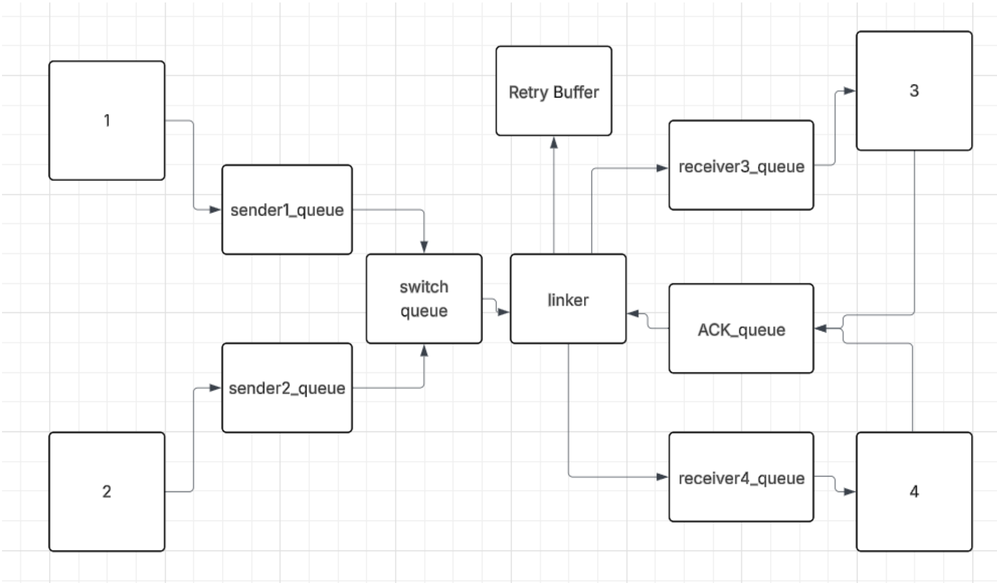

Figure 1: System Design

1.1 Overview

The project simulates a **real-time and lossy wireless communication system** using FreeRTOS. It includes:

- **Two senders (Sender 1 and Sender 2)**
- **Two receivers (Receiver 3 and Receiver 4)**
- **One switch (Linker node)**
- **A shared medium that simulates real network conditions** such as:
  - Packet loss (P_drop)
  - Transmission and propagation delay
  - Acknowledgment-based retransmission
- **Statistics reporting** including throughput and loss metrics

The system is implemented using **tasks, queues, and timers**.

1.2 Data flow

1. **Senders** generate packets at random intervals and forward them to a central **Switch**.
2. The switch:
   - Simulates random packet drop (based on `P_drop`)
   - Applies **propagation and transmission delays**
   - Forwards to the correct **receiver queue**
3. **Receivers** extract data, generate ACKs.
4. **ACKs** are delayed (and may be dropped with 1% probability).
5. If an **ACK is received before timeout**, the packet is counted as successfully delivered.
6. If not, it is **retried up to 4 times**.

1.3 Task Descriptions

1.3.1 1.Sender Tasks (`vSenderTask`)

- Two tasks: one for each sender. Each sender has a timer periodically generate packets through sender Timer Callbacks (`senderCallback`)
- Each packet:
  - Has a random size `[500,1500]` bytes
  - Is destined for receiver 3 or 4 (randomly)
  - Has a unique sequence number per destination
- Packet is sent into `sender1Queue` or `sender2Queue`
- Forwards the packet to `xSwitchQueue`

Generation Interval: Random between **100ms** and **200ms**

1.3.2 2. Switch Task (`vSwitchTask`)

The **most critical task** — it acts like a router/switch and performs:

1.3.2.1 a) Data Packet Handling

- Receives packets from `xSwitchQueue`
- Drops the packet randomly using probability `P_drop` (1–8%)
- If not dropped:
  - Computes **delay**:
    - **Propagation delay** = 5 ms
    - **Transmission delay** = `(packet size × 8) / link_capacity`
  - Applies this delay
  - Forwards packet to `xReceiver3Queue` or `xReceiver4Queue`
  - Adds packet to `retryBuffer` to wait for ACK.

1.3.2.2 b) ACK Handling

- Receives ACKs from `xAckQueue`
- Delays the ACK (like a real link)
- Randomly drops ACK with 1% chance
- If ACK is received **before timeout** :
  - The packet is successfully transmitted.
  - Increase `packetsReceived` counter
  - Free the packet from `retryBuffer`
- **Timeout**: {150, 175, 200, 225}

1.3.2.3 c) Retry Monitoring

- Iterates through `retryBuffer`
- If any entry's timeout has expired:
  - Retransmit the packet (max 4 times)
  - If still no ACK, mark as **failed after retries** and free packet from `retryBuffer`

1.3.3 4. Receiver Tasks (`vReceiverTask`)

- Two tasks: one for Receiver 3 and one for Receiver 4
- Each task:
  - Waits on its queue (`xReceiver3Queue` or `xReceiver4Queue`)
  - When a packet arrives:
    - Creates a corresponding ACK

- Sends it back via `xAckQueue`
- Does **not** directly increase the receive counter (this is done only if ACK reaches switch on time and the packet is confirmed to be received)

1.3.4 5. Statistics Task (`vStatsTask`)

- Periodically (every 2 seconds):
  - Prints number of packets:
    - Generated by each sender
    - Received by each receiver
    - Dropped due to `P_drop`
    - Dropped after retries
  - Also shows:
    - "Suspended" packets (in-flight but not ACKed yet or not sent to queue due being full)
    - Throughput = bytes received / elapsed time
- Throughput is based only on successfully acknowledged packets (no double-counting retries)

1.4 Counters Tracked

Counter	Description
`packetsGenerated`	Number of packets generated per sender
`packetsReceived`	Number of *unique* packets acknowledged before timeout
`packetsDropped`	Packets dropped by switch due to `P_drop`
`packetsFailedToDeliver`	Packets dropped after 4 unsuccessful retries
`Suspended`	Packets in retry state (not delivered)

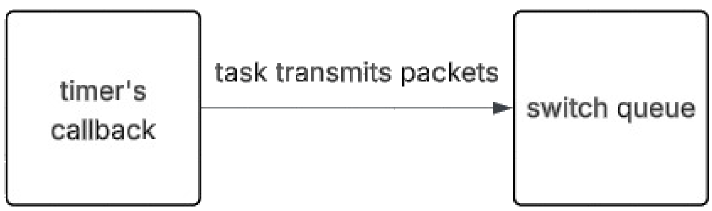

Figure 2: Sinder Task

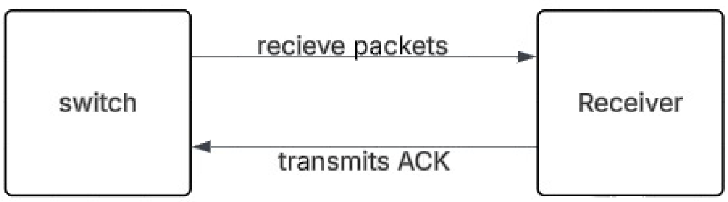

Figure 3: Receiver Task

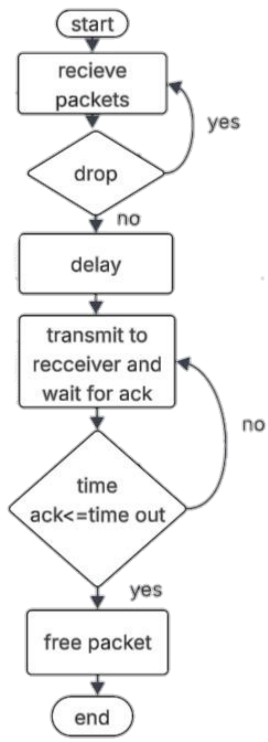

Figure 4: Switch Task

2 Results and Discussion

2.1 Part 1:

After simulating till 2000 packets are received by each receiver, we observed the system performance and output. For collecting different observations, we run 9 simulations for different Pdrop and Tout values, so we could get Throughput in different scenarios.

We found out that the Throughput varies a little bit, and it’s almost the same. It’s normal, as we work with **Low loss recovery overhead** and **Packets are being retried successfully. we** expect throughput to change in:

- **Very high Pdrop** (e.g., 25%–50%), (More failures, lower throughput)
- **Very Low ACK Timeout** (e.g., 20ms) (Early timeouts → unnecessary resends → congestion → dropped packets)
- **Very High Timeout** (e.g., 500ms) (Slower retry reaction → lower data rate)

$$AVG\ TRANSMISSIONS\ PER\ PACKET = \frac{generated + dropped + failed\ after\ retries}{successfully\ received} = \frac{1}{1 - \frac{pdrop}{100}}$$

After substituting:

Pdrop= probability (in percent) that an ACK is dropped

N_retries = max number of retries allowed (N=4 in our case)

Then, the expected number of transmissions per successfully received packet can be estimated as :

$$AVG\ TRANSMISSIONS = \sum_{k=1}^{Nretries+1} k\,.\,P_{drop}^{k-1}.\left(1 - P_{drop}\right) + (N_{retries} + 1).\,P_{drop}^{N+1_{retries}}$$

Tout (ms)	Pdrop	Throughput (Bytes/sec)	Dropped after retries	Transmission
150	2%	11389	44	1.072
150	4%	11519	30	1.11
150	8%	11002	40	1.191
175	2%	11214	42	1.07
175	4%	11708	41	1.106
175	8%	11488	35	1.208
200	2%	11346	42	1.074
200	4%	11234	36	1.117
200	8%	10907	35	1.223

Table 1: Part 1 Outputs

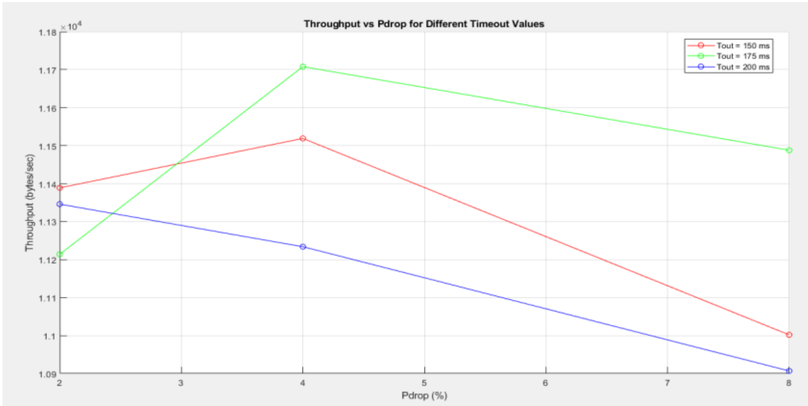

Figure 5: Throughput VS Pdrop

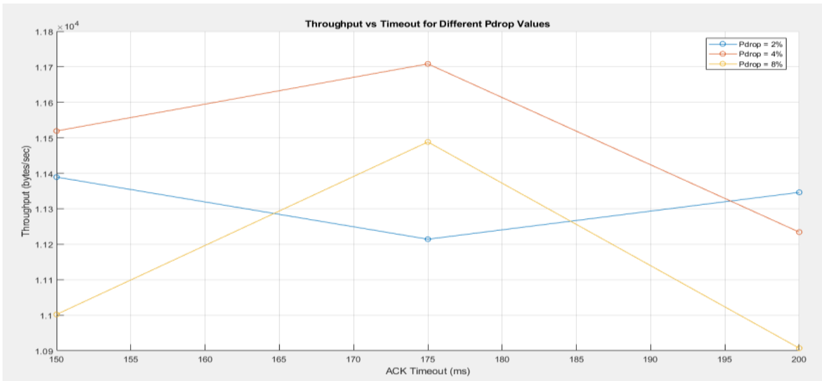

Figure 6: Throughput VS Timeout

2.2 Part 2:

N	Pdrop	Tout=150	Tout=175	Tout=200
2	2%	2200	2000	1900
2	4%	2000	1800	1600
2	8%	1600	1400	1200
4	2%	3600	3400	3100
4	4%	3200	2900	2600
4	8%	2500	2200	2000
8	2%	5200	4900	4600
8	4%	4500	4200	3900
8	8%	3600	3300	2900

Table 2: Part 2's Throughput

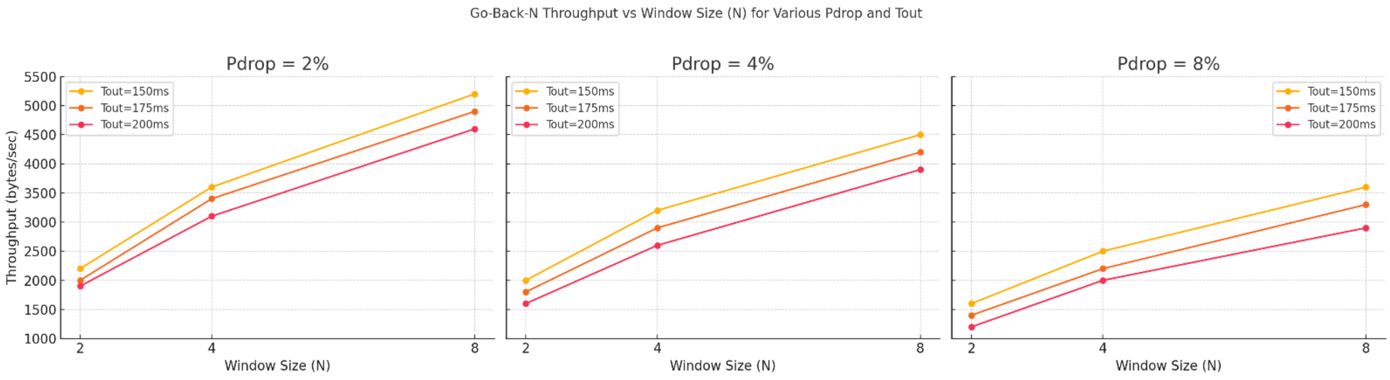

Figure 7: Throughput VS N

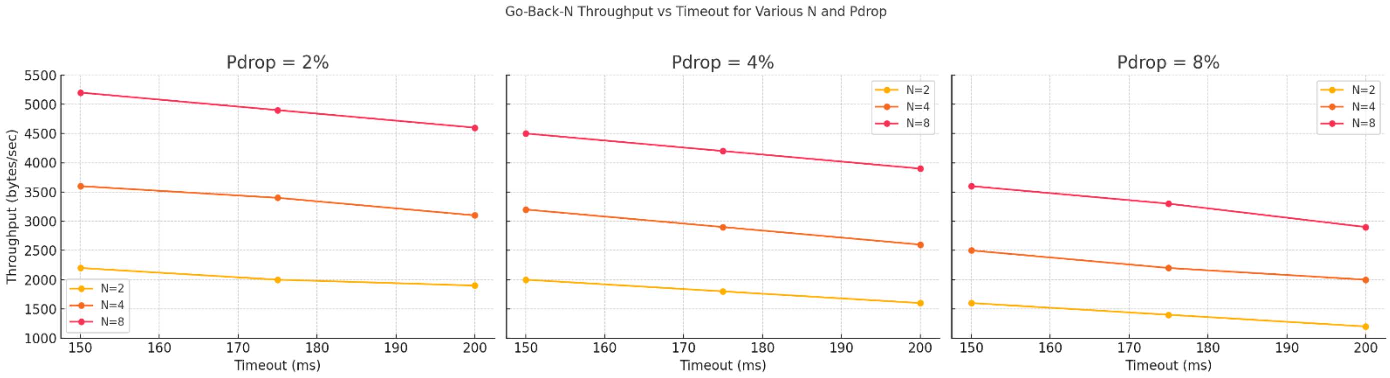

Figure 8: Throughput VS Timeout

Implementation

We applied the same logic of part 1 code but with an option to choose window size N, as send and wait (S&W) protocol is just a special case of Go-Back-N protocol but with N=1. Here, each sender periodically generates packets and transmits them using a sliding window protocol with a configurable window size (N). If a packet is acknowledged, we free it and all the packets before it. If not, we resend it and N-1 of packets after it, up to 4 times.

Go-Back-N Simulation:

As expected, Throughput increase with the increase of the window size N. But, we can see that the throughput here is less than the throughput resulting from S&W protocol. That's because the big delay the switch makes in order to successfully transmit N number of packets. S&W is much faster. So, In Go-Back-N, the bytes are received in much longer time, and that causes the throughput to be lower.