CS3103 Assignment 4 Technical Report

1. Design Choice

- **gameNetAPI:** reliable UDP with Selective Repeat; unreliable UDP
- **packet design:** see below
- **ack:** SACK (Selective Acknowledgment)
- **synchronization:** concurrent access is synchronized to prevent race conditions
- **skipping rule:** default set to 200 ms; configurable

2. GameNetAPI

Method: `send`

```
send(data: bytes, reliable: bool = True) -> None
```

Description

Sends a message to the remote peer.

- When `reliable=True` (default): uses **reliable channel** with acknowledgments, retransmissions, and ordered delivery.
- When `reliable=False`: uses **unreliable channel**, which sends the packet once without guarantees.

Parameters

Name	Type	Default	Description
`data`	`bytes`	—	Payload to send.
`reliable`	`bool`	`True`	Whether to send via reliable channel.

Raises

- `RuntimeError`: if called on a receiver-only endpoint (no remote address).

Method: `receive`

```
receive() -> tuple[int | None, int, bytes] | None
```

Description

Retrieves the next available message from the internal delivery queue.
This method is **non-blocking** — it returns immediately if no data is available.

Returns

Type	Description
`(seq, ts_ms, payload)`	For reliable packets — includes sequence number and timestamp.
`(None, ts_ms, payload)`	For unreliable packets — sequence number is `None`.
`None`	If no packets are available.

Method: `close`

```
close() -> None
```

3. Test Setup

The testing environment is automated using a Bash script (`test.sh`) that simulates real network behavior and verifies both reliable and unreliable transmission over UDP.

Overview

The script launches:

1. A **receiver** process running `ReliableUDP_API` in background.
2. Optionally applies **`tc netem`** on the loopback interface to emulate network conditions such as loss, delay, jitter, and packet reordering.
3. A **sender** process that transmits both reliable and unreliable packets interleaved at a specified rate.

Test Cases

1 Functional Correctness

Objective: Verify correct operation of reliable and unreliable channels under normal conditions.
Setup:

- No packet loss or delay (`USE_NETEM=0`).
- Sender transmits 20 reliable and 10 unreliable packets.
  Expected Outcome:
- All reliable packets are received in order.

2 Packet Loss Simulation

Objective: Test retransmission and Selective Repeat (SR) reliability.
Setup:

- Enable `LOSS=10`.
- 50 reliable and 20 unreliable packets.
  Expected Outcome:
- Lost reliable packets are retransmitted until acknowledged.
- Receiver eventually delivers all reliable packets in order.
- Some unreliable packets are permanently lost.

3 Delay and Jitter and Reordering

Objective: Assess timing tolerance and in-order buffering under variable latency.
Setup:

- Apply `DELAY_MS=200`, `JITTER_MS=50`, `LOSS=0`, `REORDER=20`.
- 40 reliable and 20 unreliable packets at 10 pps.
  Expected Outcome:
- Receiver handles delayed packets with reordering buffer.
- Delivery latency increases, but sequence integrity is preserved.
- Unreliable packets arrive with non-deterministic delay.

Test result

- Functional Correctness: passed
- Packet Loss Simulation: passed
- Delay and Jitter: passed

4. Packet Header Format

```
+-----------+---------------------+----------------------------------+
| 1 byte    | 2 bytes             | 4 bytes                          |
+-----------+---------------------+----------------------------------+
| chan      | seq                 | ts_ms                            |
+-----------+---------------------+----------------------------------+
```

5. Protocol Flowchart

Sender Flow (Reliable vs Unreliable)

Step	Reliable Channel	Unreliable Channel
Build Packet	Build packet: `DATA_CHANNEL`, `seq = rsn`, `ts = now`	Build packet: `UNREL_CHANNEL`, `seq = useq`, `ts = now`
Send	Send packet	Send packet
Timers	Start retransmission timer `T_rsn`	*(No retransmission timer)*
Buffering	Store packet in `send_buffer[rsn]`	*(No buffering)*
Sequence Update	`rsn++`, increment inflight counter	`useq++`

ACK Handling (Sender)

Event	Sender Action
Receive `ACK(ack_num)`	Cancel timer `T_ack_num`
Buffer Update	Remove entry from `send_buffer`
Window Update	Free window slot → send pending packets if any
Timer Expiry	Retransmit packet and restart timer

Receiver Flow (Reliable vs Unreliable)

Step	Reliable Channel	Unreliable Channel
Acknowledgment	Send `ACK(seq)`	*(No ACK sent)*
Duplicate Check	If `seq < next_expected`: ignore duplicate	*(No duplicate handling)*
Buffering	Store packet in `receive_buffer[seq]`	*(No buffering)*
In-Order Delivery	If continuous sequence available: Deliver to application, `next_expected++`, clear `skip_deadline`	Deliver `(None, ts, payload)` directly to application
Handling Gaps	If gap exists and no deadline: `skip_deadline = now + SKIP_TIMEOUT`	*(No gap handling)*

6. Latency/Jitter/Throughput

run the test in windows 1.37.0.0, with receiver and sender attached to local host, port 6000 6001

Latency

Metric	Reliable Channel	Unreliable Channel
Average Latency	0.20 ms	0.19 ms
Minimum Latency	0.00 ms	0.00 ms
Maximum Latency	1.00 ms	1.00 ms

Jitter

Metric	Reliable Channel	Unreliable Channel
Jitter (StdDev)	0.40 ms	0.39 ms

Throughput

Total Bytes Received	Test Duration	Average Throughput
3,895,119 bytes	35.01 s	890.00 kbps