

# Report

## (a) Program Structure and Design

The `TCPSender` class is responsible for TCP message transmission, managing sequence numbers, acknowledgments, retransmissions, and flow control. Key private fields track bytes sent (`sent_bytes_`), bytes acknowledged (`acked_bytes_`), and retransmission count (`retry_count_`). This setup simplifies the tracking of transmission states, enhancing flow control and retransmission reliability.

1. **Flow Control:** `receiver_window_` manages the allowable bytes in-flight, helping control payload size and respecting the receiver's window.
2. **Retransmission:** An exponential backoff mechanism using `current_RTO_ms_` controls retransmission timing.
3. **Zero Window Handling:** When `receiver_window_` is zero, retransmission timers avoid backoff, ensuring reliable retries in low-window situations.
4. **Message Queue:** A `std::queue` maintains messages sent but unacknowledged. This queue structure allows for reliable retransmissions, tracks each packet's sequence number (`seqno`), and simplifies removing acknowledged packets.

This design emphasizes modularity and simplicity, which reduces the risk of bugs. By limiting the fields to essential state-tracking ones (`sent_bytes_`, `acked_bytes_`, etc.), the code remains easier to debug and understand.

## (b) Implementation Challenges

- **Understanding Lab Requirements:** Interpreting the lab specifications took time but ultimately clarified the design approach.
- **Window Size Calculation:** Calculating the effective window size was tricky, as it needed to include SYN packets but sometimes exclude FIN packets.
- **C++ Compilation Challenges:** While challenging, previous labs made navigating compiler nuances manageable.

## (c) Remaining Bugs

The code currently passes all tests for `check3`. Thus, no known bugs remain.

## (d) Experimental Results and Performance

The results, as shown in Figure 1, confirm that the `TCPSender` implementation meets the performance requirements and effectively handles retransmission and flow control.

```

    Start 21: recv_connect
20/36 Test #21: recv_connect ..... Passed    0.02 sec
    Start 22: recv_transmit
21/36 Test #22: recv_transmit ..... Passed    0.78 sec
    Start 23: recv_window
22/36 Test #23: recv_window ..... Passed    0.02 sec
    Start 24: recv_reorder
23/36 Test #24: recv_reorder ..... Passed    0.03 sec
    Start 25: recv_reorder_more
24/36 Test #25: recv_reorder_more ..... Passed    1.15 sec
    Start 26: recv_close
25/36 Test #26: recv_close ..... Passed    0.02 sec
    Start 27: recv_special
26/36 Test #27: recv_special ..... Passed    0.02 sec
    Start 28: send_connect
27/36 Test #28: send_connect ..... Passed    0.02 sec
    Start 29: send_transmit
28/36 Test #29: send_transmit ..... Passed    0.53 sec
    Start 30: send_retx
29/36 Test #30: send_retx ..... Passed    0.02 sec
    Start 31: send_window
30/36 Test #31: send_window ..... Passed    0.11 sec
    Start 32: send_ack
31/36 Test #32: send_ack ..... Passed    0.02 sec
    Start 33: send_close
32/36 Test #33: send_close ..... Passed    0.02 sec
    Start 34: send_extra
33/36 Test #34: send_extra ..... Passed    0.05 sec
    Start 37: compile with optimization
34/36 Test #37: compile with optimization ..... Passed    0.20 sec
    Start 38: byte_stream_speed_test
    ByteStream throughput: 3.38 Gbit/s
35/36 Test #38: byte_stream_speed_test ..... Passed    0.14 sec
    Start 39: reassembler_speed_test
    Reassembler throughput: 10.52 Gbit/s
36/36 Test #39: reassembler_speed_test ..... Passed    0.22 sec

100% tests passed, 0 tests failed out of 36

Total Test time (real) =    5.92 sec
Built target check3
➔ minnow git:(main) ×

```

Figure 1: Results and Performance