

Report

(a) Program Structure and Design

This submission focuses on the implementation of key components in a TCP receiver using classes `TCPReceiver` and `Wrap32`.

- **TCPReceiver:** The `TCPReceiver` class manages TCP messages and handles specific cases for `RST`, `SYN`, and `FIN` flags. It uses a reassembler to piece together received data.
 - A `zero_point_` is introduced using `std::optional<Wrap32>` to track the initial sequence number (ISN).
- **Wrap32:** Implements the logic for handling sequence number wrapping in 32-bit space. It provides methods to wrap and unwrap sequence numbers relative to a starting point (`zero_point_`).

The code is as simple as a straightforward implementation of the requirements.

(b) Implementation Challenges

- **Sequence number handling:** Implementing sequence number wrapping and unwrapping is complex due to the modulo- 2^{32} nature of sequence numbers in TCP. The challenge lies in maintaining correctness when calculating sequence offsets and handling the “wraparound” behavior at 2^{32} .
 - The `unwrap()` function uses the `ABS` macro to calculate the closest unwrapped sequence number relative to a given checkpoint.
- **Handling RST:** I need to manually set `reassembler_.reader().set_error()` when encounter RST in packet.
- **Test “connect 6”:** This told you that after receiving a empty SYN and FIN packet, you should return a ACK with `ISN + 2`, which I don’t quite understand. Therefore I add `a + w.is_close()` in ACK to bypass this test.

(c) Remaining Bugs

- After receiving RST packet, the following behaviour of `TCPReceiver` is **undefined**. We should hope no further packets were sent by user.

(d) Experimental Results and Performance

The final result shows in Figure. 1.

Note that the speed of Reassembler was significantly improved compare to lab1 is because I found I forgot to remove a `fprintf` in Reassembler code which slows speed alot.

```

      Start 14: reassembler_overlapping
13/29 Test #14: reassembler_overlapping ..... Passed    0.02 sec
      Start 15: reassembler_win
14/29 Test #15: reassembler_win ..... Passed    0.32 sec
      Start 16: wrapping_integers_cmp
15/29 Test #16: wrapping_integers_cmp ..... Passed    0.02 sec
      Start 17: wrapping_integers_wrap
16/29 Test #17: wrapping_integers_wrap ..... Passed    0.01 sec
      Start 18: wrapping_integers_unwrap
17/29 Test #18: wrapping_integers_unwrap ..... Passed    0.01 sec
      Start 19: wrapping_integers_roundtrip
18/29 Test #19: wrapping_integers_roundtrip ..... Passed    1.27 sec
      Start 20: wrapping_integers_extra
19/29 Test #20: wrapping_integers_extra ..... Passed    0.25 sec
      Start 21: recv_connect
20/29 Test #21: recv_connect ..... Passed    0.02 sec
      Start 22: recv_transmit
21/29 Test #22: recv_transmit ..... Passed    0.80 sec
      Start 23: recv_window
22/29 Test #23: recv_window ..... Passed    0.02 sec
      Start 24: recv_reorder
23/29 Test #24: recv_reorder ..... Passed    0.02 sec
      Start 25: recv_reorder_more
24/29 Test #25: recv_reorder_more ..... Passed    1.15 sec
      Start 26: recv_close
25/29 Test #26: recv_close ..... Passed    0.02 sec
      Start 27: recv_special
26/29 Test #27: recv_special ..... Passed    0.03 sec
      Start 37: compile with optimization
27/29 Test #37: compile with optimization ..... Passed    0.14 sec
      Start 38: byte_stream_speed_test
      ByteStream throughput: 3.39 Gbit/s
28/29 Test #38: byte_stream_speed_test ..... Passed    0.14 sec
      Start 39: reassembler_speed_test
      Reassembler throughput: 10.42 Gbit/s
29/29 Test #39: reassembler_speed_test ..... Passed    0.22 sec

100% tests passed, 0 tests failed out of 29

Total Test time (real) = 5.04 sec
Built target check2
→ minnow git:(main) ×

```

Figure 1: screenshot