

# Lab 4 Writeup

---

My name: Zuo, Qikun

My SUNet ID: 201830013

This lab took me about 8 hours to do. I did attend the lab session.

I worked with or talked about this assignment with: Nobody

Program Structure and Design of the TCPSender:

First, we add a new class RetransmissionTimer to tcp\_sender.hh serving as the timer for TCPSender. The meaning of each member variable and the role of each member function in the class have been marked in the following code.

```
class RetransmissionTimer {
private:
    size_t _ticks{0}; // total elapsed time

    bool _is_started{false}; // is the timer started or not

public:
    bool is_expired(const size_t ms_since_last_tick, const unsigned
retransmission_timeout) {
        return _is_started && (_ticks += ms_since_last_tick) >=
retransmission_timeout;
    } // decide whether the retransmission timer expires or not

    bool is_started() { return _is_started; } // is the timer started or
not

    void stop() { _is_started = false; } // stop the timer (when all
outstanding data has been acknowledged)

    void start() { // start the timer
        _is_started = true;
        _ticks = 0;
    }
};
```

Second, we add some member variables to the class TCPSender. The meaning of each member variable added to the class has been marked in the following code.

```
size_t _remaining_size{0}; // remaining space for sending new bytes to the
receiver

std::queue<TCPSegment> _outstanding_segments{}; // segments which are not
```

```

acknowledged, stored in a queue

uint64_t _absolute_ackno{0}; // latest (largest) ackno received from the
receiver

size_t _window_size{1}; // latest window size received from the receiver

RetransmissionTimer _retransmission_timer{}; // retransmission timer

unsigned int _retransmission_timeout; // current retransmission timeout
(RTO)

size_t _consecutive_retransmissions{0}; // the number of consecutive
retransmissions

bool _finish_sending{false}; // finish sending or not

```

Third, we focused our major attention on implementing the `fill_window()` and `ack_received()` in `tcp_sender.cc`.

a. In `fill_window()`, the TCP sender should write all of the fields of the `TCPSegment` that were relevant to the `TCPReceiver`, namely, (1) the SYN flag, (2) the sequence number, (3) the payload, (4) the FIN flag. All of these 4 steps, as well as sending the segments (pushing into `_segments_out`), storing the outstanding segments in a queue data structure `_outstanding_segments`, and the update of member variables (`_retransmission_timer`, `_next_seqno`, `_bytes_in_flight`, etc.), are reflected in the code shown below, which is fully commented. And the following figure shows the quantitative relationship between the member variables.

```

void TCPSender::fill_window() {
    if(_finish_sending) {
        return;
    }

    _remaining_size = (_window_size ? _window_size : 1) -
(next_seqno_absolute() - _absolute_ackno); // remaining space for sending
new bytes to the receiver
    // have the sending space and not get to sending ending

    while (_remaining_size > 0 && !_finish_sending) {
        TCPSegment segment = TCPSegment();

        if (next_seqno_absolute() == 0) { // (1) handling SYN
            segment.header().syn = true;
            _remaining_size--; // syn occupies space in the window
        }

        segment.header().seqno = next_seqno(); // (2) handling sequence
number (seqno)

        segment.payload() = stream_in().read(min(_remaining_size,
TCPConfig::MAX_PAYLOAD_SIZE)); // (3) handling payload // use

```

```

min(_remaining_size, MSS)

    _remaining_size -= segment.payload().size();

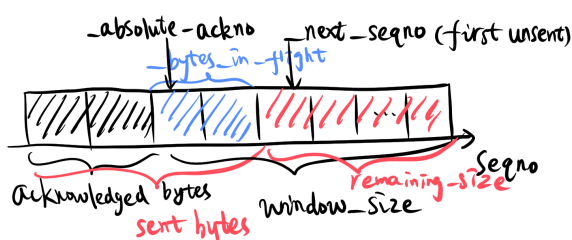
    if (stream_in().eof() && _remaining_size > 0) { // (4) handling FIN
        segment.header().fin = true;
        _finish_sending = true;
        _remaining_size--; // fin occupies space in the window
    }

    if (segment.length_in_sequence_space() == 0) { //
segment.length_in_sequence_space() == length of the segment
        return;
    }

    segments_out().push(segment); // send segment
    _outstanding_segments.push(segment); // store outstanding segment
    if (!_retransmission_timer.is_started()) { // Every time a segment
containing data (nonzero length in sequence space) is sent (whether it's
the first time or a retransmission), if the timer is not running, start it
running so that it will expire after RTO milliseconds (for the current
value of RTO).
        _retransmission_timer.start();
    }

    // update members in class TCPSender
    _next_seqno += segment.length_in_sequence_space(); // bytes sent to
the receiver increases by segment.length_in_sequence_space()
    _bytes_in_flight += segment.length_in_sequence_space(); //
outstanding bytes increase by segment.length_in_sequence_space()
    }
}

```



b. In `ack_received()`, the TCP sender only reads the fields in the segment that are written by the receiver: (1) the `ackno` and (2) the `window size`. After getting the `window size`, the function will use it to update `_window_size`; After getting the `ackno`, the function will use it to decide whether each of the outstanding segments stored in `_outstanding_segments` should be popped out or not. And if the receiver acknowledges the successful receipt of new data, the TCP sender should: (a) Set the `RTO` back to its "initial value." (b) If the sender has any outstanding data, restart the `retransmission timer` so that it will expire after `RTO` milliseconds (for the current value of `RTO`). (c) Reset the count of "consecutive retransmissions" back to zero. The code shown below is fully commented, and the following figure shows the checking process of `_outstanding_segments`.

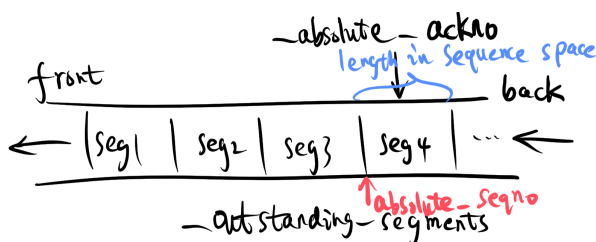
```

void TCPSender::ack_received(const WrappingInt32 ackno, const uint16_t
window_size) {
    _absolute_ackno = unwrap(ackno, _isn, next_seqno_absolute()); // (1)
    read the ackno
    _window_size = window_size; // (2) read the window size (and update
_window_size)

    bool flag = false;
    while (_outstanding_segments.size() > 0) {
        TCPSegment segment = _outstanding_segments.front();
        uint64_t absolute_seqno = unwrap(segment.header().seqno, _isn,
next_seqno_absolute());
        if (absolute_seqno + segment.length_in_sequence_space() >
_absolute_ackno) { // the segment is not fully acknowledged by the receiver
            break; // so do the segments following the segment
        }
        _outstanding_segments.pop(); // fully acknowledged segment
        _bytes_in_flight -= segment.length_in_sequence_space(); //
outstanding bytes decrease by segment.length_in_sequence_space()
        flag = true; // the receiver acknowledges the successful receipt of
new data
    }

    if (flag) { // acknowledges the successful receipt of new data
        _retransmission_timeout = _initial_retransmission_timeout; // Set
the RTO back to its "initial value.
        if (_outstanding_segments.size() > 0) { // If the sender has any
outstanding data, restart the retransmission timer so that it will expire
after RTO milliseconds (for the current value of RTO).
            _retransmission_timer.start();
        } else { // When all outstanding data has been acknowledged, stop
the retransmission timer.
            _retransmission_timer.stop();
        }
        _consecutive_retransmissions = 0; // Reset the count of
"consecutive retransmissions" back to zero.
    }
}

```



Implementation Challenges: In practice, I found that the biggest difficulty in implementation is to sort out the quantitative relationship between the various member variables, as shown in Figure 1. Especially, the update of `_remaining_size` is quite critical. With just a little bit of misconception (for example, not taking into account that the SYN and FIN flags also take up space), some test cases will fail. Also, it is also important to read the tutorial extremely carefully before implementing these functions, as the tutorial

shows some critical steps and details. Therefore, if ignoring these steps and details, it is difficult to pass all test cases.

### Remaining Bugs:

```
cs144@cs144vm:~/lab4-SophisRousseau/sponge/build$ make check_lab4
[100%] Testing the TCP sender...
Test project /home/cs144/lab4-SophisRousseau/sponge/build
Start 1: t_wrapping_ints_cmp ..... Passed 0.00 sec
1/33 Test #1: t_wrapping_ints_cmp ..... Passed 0.00 sec
Start 2: t_wrapping_ints_unwrap ..... Passed 0.00 sec
2/33 Test #2: t_wrapping_ints_unwrap ..... Passed 0.00 sec
Start 3: t_wrapping_ints_wrap ..... Passed 0.00 sec
3/33 Test #3: t_wrapping_ints_wrap ..... Passed 0.00 sec
Start 4: t_wrapping_ints_roundtrip ..... Passed 0.14 sec
4/33 Test #4: t_wrapping_ints_roundtrip ..... Passed 0.14 sec
Start 5: t_rcv_connect ..... Passed 0.00 sec
5/33 Test #5: t_rcv_connect ..... Passed 0.00 sec
Start 6: t_rcv_transmit ..... Passed 0.04 sec
6/33 Test #6: t_rcv_transmit ..... Passed 0.04 sec
Start 7: t_rcv_window ..... Passed 0.00 sec
7/33 Test #7: t_rcv_window ..... Passed 0.00 sec
Start 8: t_rcv_reorder ..... Passed 0.01 sec
8/33 Test #8: t_rcv_reorder ..... Passed 0.01 sec
Start 9: t_rcv_close ..... Passed 0.01 sec
9/33 Test #9: t_rcv_close ..... Passed 0.01 sec
Start 10: t_rcv_special ..... Passed 0.00 sec
10/33 Test #10: t_rcv_special ..... Passed 0.00 sec
Start 11: t_send_connect ..... Passed 0.01 sec
11/33 Test #11: t_send_connect ..... Passed 0.01 sec
Start 12: t_send_transmit ..... Passed 0.04 sec
12/33 Test #12: t_send_transmit ..... Passed 0.04 sec
Start 13: t_send_rettx ..... Passed 0.01 sec
13/33 Test #13: t_send_rettx ..... Passed 0.01 sec
Start 14: t_send_window ..... Passed 0.04 sec
14/33 Test #14: t_send_window ..... Passed 0.04 sec
Start 15: t_send_ack ..... Passed 0.00 sec
15/33 Test #15: t_send_ack ..... Passed 0.00 sec
Start 16: t_send_close ..... Passed 0.00 sec
16/33 Test #16: t_send_close ..... Passed 0.00 sec
Start 17: t_send_extra ..... Passed 0.01 sec
17/33 Test #17: t_send_extra ..... Passed 0.01 sec
Start 18: t_strm_reassem_single ..... Passed 0.00 sec
18/33 Test #18: t_strm_reassem_single ..... Passed 0.00 sec
Start 19: t_strm_reassem_seq ..... Passed 0.01 sec
19/33 Test #19: t_strm_reassem_seq ..... Passed 0.01 sec
Start 20: t_strm_reassem_dup ..... Passed 0.01 sec
20/33 Test #20: t_strm_reassem_dup ..... Passed 0.01 sec
Start 21: t_strm_reassem_holes ..... Passed 0.00 sec
21/33 Test #21: t_strm_reassem_holes ..... Passed 0.00 sec
Start 22: t_strm_reassem_many ..... Passed 0.44 sec
22/33 Test #22: t_strm_reassem_many ..... Passed 0.44 sec
Start 23: t_strm_reassem_overlapping ..... Passed 0.00 sec
23/33 Test #23: t_strm_reassem_overlapping ..... Passed 0.00 sec
Start 24: t_strm_reassem_win ..... Passed 0.42 sec
24/33 Test #24: t_strm_reassem_win ..... Passed 0.42 sec
Start 25: t_strm_reassem_cap ..... Passed 0.09 sec
25/33 Test #25: t_strm_reassem_cap ..... Passed 0.09 sec
Start 26: t_byte_stream_construction ..... Passed 0.00 sec
26/33 Test #26: t_byte_stream_construction ..... Passed 0.00 sec
Start 27: t_byte_stream_one_write ..... Passed 0.00 sec
27/33 Test #27: t_byte_stream_one_write ..... Passed 0.00 sec
Start 28: t_byte_stream_two_writes ..... Passed 0.00 sec
28/33 Test #28: t_byte_stream_two_writes ..... Passed 0.00 sec
Start 29: t_byte_stream_capacity ..... Passed 0.35 sec
29/33 Test #29: t_byte_stream_capacity ..... Passed 0.35 sec
Start 30: t_byte_stream_many_writes ..... Passed 0.01 sec
30/33 Test #30: t_byte_stream_many_writes ..... Passed 0.01 sec
Start 53: t_address_dt ..... Passed 5.06 sec
31/33 Test #53: t_address_dt ..... Passed 5.06 sec
Start 54: t_parser_dt ..... Passed 0.00 sec
32/33 Test #54: t_parser_dt ..... Passed 0.00 sec
Start 55: t_socket_dt ..... Passed 0.01 sec
33/33 Test #55: t_socket_dt ..... Passed 0.01 sec

100% tests passed, 0 tests failed out of 33

Total Test time (real) = 6.85 sec
[100%] Built target check_lab4
```

Until now, no new bugs are found in the code committed.