Charlie Andre
CSCI 551
11/8/19

Lab 3A Write Up

Program Structure and Design - Describe the high-level structure of your code, by insisting on what you actually added to the code.

- Some key decision made early:
  - segment wrapper
    - allowed me to keep track of
      - the number of times I sent a segment
      - the time of the last send of that segment, so I could make sure not to send again before the retransmission timeout
  - included the config structure in my state struct
    - allowed for neat and easy importation of the configurations upon initialization
    - made code more readable
  - linked list of unacknowledged segments as a linked list of wrapped segments
    - this decision was made because:
      - I would need to keep track of how many times I sent these. If any packet is sent more than 5 times, you destroy the connection
      - I needed to track the last time that the packet was sent so that I could wait until retransmission timeout before sending again
  - Linked list of unoutputted segments as plain, unwrapped ctcp_segments
    - I did not need to try multiple times to output these, so there was no need to wrap them.
- Algorithms – these are the two that took the longest.
  - ordering unoutputted segments by sequence number
    - if this linked list is empty, simply add segment to the linked list
    - otherwise, you have to check sequence numbers for sorting.
      - Look at the head of the linked list. If this segment should go before it, make this segment the new head
      - Otherwise, loop through the linked list till the nodes seq number is greater than the segments seq number. Add the segment here.
      - This was a helpful online reference: https://www.geeksforgeeks.org/given-a-linked-list-which-is-sorted-how-will-you-insert-in-sorted-way/
  - Outputting segments
    - Check if there is room, if yes output the segment, update amount of space available in the buffer, update state variables. When this is all done, you can ACK based on the most recent state variables.
      - This allows you to ACK part of the unoutputted linked list without the buffer space to output all of it at once.
  - Ctcp_destroy: I used the following link to guide me through how to free struct containing pointer to structs that has pointers with structs.

        - For ctcp_destory: https://stackoverflow.com/questions/4915369/how-to-free-a-struct-that-contains-only-pointers
- networking principles:
    - Stop and Wait:
        - for this lab, I implemented a stop and wait protocol. The sender does not send until the previously sent segment has been ACK'd. This is tracked using the bool state->recv_ack
    - in order delivery
        - this is guaranteed by the algorithm that I use for adding segments to the unoutputted segments linked list in order
    - reliable delivery
        - this is ensured using ACK's. If delivery is not happening reliably, the sender will not receive ACKs. After 5 tries on the same segment, the sender will give up. Connections where delivery is unreliable can't exist in TCP.

Implementation Challenges - Describe the parts of code that you found most troublesome and explain why. Reflect on how you overcame those challenges and what helped you finally understand the concept that was giving you trouble.

- Outputting segments in order—
    - I implemented something (sort of) similar in EE 542, which allowed me to more easily see how I would deal with sorting segments that arrived out of order, so that I could output them in the correct order. However, actually doing this again was quite hard. I eventually had to draw out the linked list to develop the final algorithm the allowed me to add unoutputted segments to the respective linked list in the correct order.
    - Another change is that in EE542, I was allowed to make the buffer size as big as I wanted, which made the assignment much easier.
    - This is described in the algorithms section
- Getting ACK_num and Seq_num and seq_num_next all sorted out.
    - This was really tough. At first, I started writing code without really thinking about what I was doing—classic mistake. My state variables were off. I fixed this by reading some tutorials. This one was particularly helpful: https://madpackets.com/2018/04/25/tcp-sequence-and-acknowledgement-numbers-explained/
    - I also had to use a temporary variable send_num to help keep track of sending. This was closely related to an outputted segments ackno. This helped track the ackno of the last ACK packet that I received.
- Removing an unoutputted segment from a linked list was a bit of a challenge because I had to ensure that I managed the link list just the right way.
    - I was greatly helped by this tutorial: https://www.geeksforgeeks.org/delete-a-given-node-in-linked-list-under-given-constraints/

Testing - Describe the tests you performed on your code to evaluate functionality and robustness. Enumerate possible edge cases and explain how you tested for them. Additionally, talk about general testing strategies you used to test systems code.

I basically tried to pass the first few tests in order:

1. ctcp.c on client side and reference on server side.
    a. Pass if: my implementation can send to the server
2. Ref on client, ctcp on server
    a. Pass if ctcp outputs correct info
3. For checksum, I relied on the test in the script.
4. Then I made sure that I could pass these same tests with my binary on both sides.
5. The other algorithms that I implemented, I tested by printing out the key values
6. Then I would test these by forcing the condition. For instance, to verify that I could handle data larger than the window size, I would send a large file.
7. Each linked list algorithm took tuning and printing segments and variables to get it right. Also would test with methods similar to 6, forcing their inconsistencies to be exposed so I could see what they are and fix them
8. After debugging the individual linked list algorithms, with help from the listed online sources, I ran the testing script, and passed a good amount of the tests.
9. I figured I was done testing as the score I got (130) was enough for me, especially since this lab is now graded out of 140. Sometimes I even get 140, for full credit. I was even able to pass 15 tests, but was not able to replicate this.

To summarize my main strategies for testing:
Printing out key variables, referencing lots of tutorials online (links given throughout the report), drawing the linked lists when I would get really stuck.
Testing using basic server client connection and sending to do each test individually with lots of prints, then once those all worked, running the testing script, to see that they worked there.

```
Starting tests...

Results
-------
  1. Client sends data ..................................... PASS
  2. Client receives data .................................. PASS
  3. Correct checksum ...................................... PASS
  4. Correct header fields ................................. FAIL
  5. Bidirectionally transfer data ......................... FAIL
  6. Handles data larger than window size .................. FAIL
  7. Handles segment corruption ............................ PASS
  8. Handles segment drops ................................. PASS
  9. Handles segment delay ................................. PASS
  10. Handles duplicate segments .......................... PASS
  11. Handles truncated segments .......................... PASS
  12. Sends FIN when reading in EOF ....................... PASS
  13. Tears down connection ............................... PASS
  14. Handles sliding window .............................. PASS
  15. Talks to Google ..................................... FAIL
  16. Talks to Bing ....................................... FAIL
  17. Supports different send/receive windows ............. PASS
  18. Window size field set in header ..................... FAIL
  19. Supports multiple clients ........................... PASS

PASSED: 13/19

SCORE: 130/190
root@mininet-vm:~/candre97-cs551/lab3#
```

```
Results
-------
  1. Client sends data ..................................... PASS
  2. Client receives data .................................. PASS
  3. Correct checksum ...................................... PASS
  4. Correct header fields ................................. FAIL
  5. Bidirectionally transfer data ......................... FAIL
  6. Handles data larger than window size .................. PASS
  7. Handles segment corruption ............................ PASS
  8. Handles segment drops ................................. PASS
  9. Handles segment delay ................................. PASS
  10. Handles duplicate segments .......................... PASS
  11. Handles truncated segments .......................... PASS
  12. Sends FIN when reading in EOF ....................... PASS
  13. Tears down connection ............................... PASS
  14. Handles sliding window .............................. PASS
  15. Talks to Google ..................................... FAIL
  16. Talks to Bing ....................................... FAIL
  17. Supports different send/receive windows ............. PASS
  18. Window size field set in header ..................... FAIL
  19. Supports multiple clients ........................... PASS

PASSED: 14/19

SCORE: 140/190
root@mininet-vm:~/candre97-cs551/lab3# git commit -am "passing 14/19 tests"
```

Remaining Bugs - Point out and explain as best you can any bugs that remain in the code.

Failed tests (excluding #15 & #16)

- Correct header fields – most likely an htons / ntohs error because there are a lot of other test cases that work. I played around with all the fields that needed this doing a bunch of different variations, and could not see where the error was. Upon printing segments, I

couldn’t tell what was wrong – should be a quick fix after I haven’t looked at this code for a while.

- Bidirectional transfer data – most likely due to the header fields being incorrect. Not totally sure why this fails, besides the headers being incorrect because I think I handle this in my code.

- Window size field set in header – could be an issue with how config is set from memory, or could be an issue with how its adjusted

- Overall, I think that I could fix these bug if I were given one more full day. However, I had to stop coding around 9pm in order to write up this report. I think that I can squash these bugs if lab 3B builds on lab 3A.