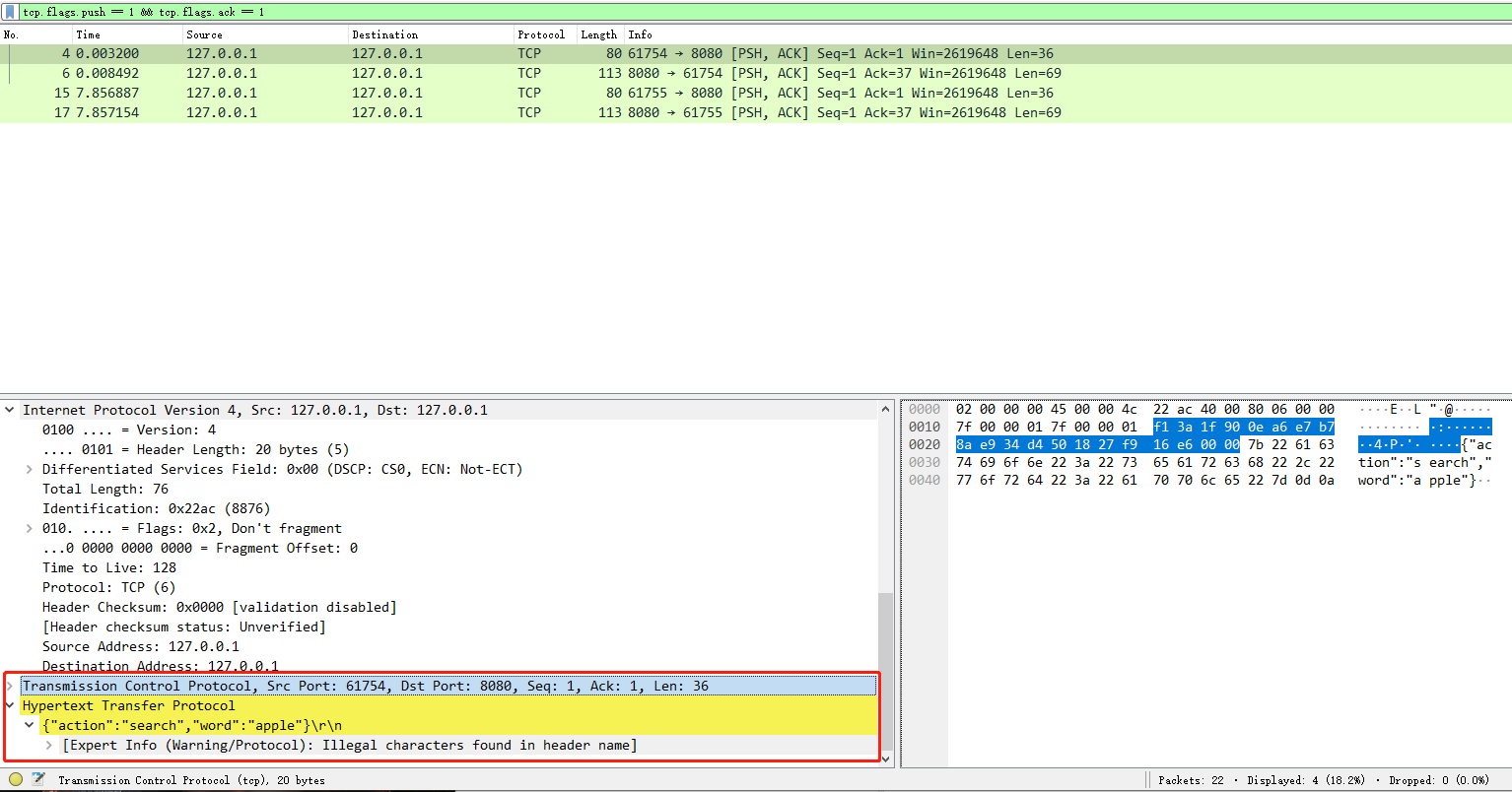
1. Basic Compliance Tests:

1.1. Protocol and Concurrency:

Test the usage of TCP sockets for communication between client and server.



Ensure multiple clients can connect to the server simultaneously.

device 1: windows 1: search word 'apple' on one window, window2: add 'berry' to dictionary on another window device 2: window1: updated word 'apple' on one window, window2 :remove 'grape' from dictionary response: device 1 window 1 get the new meaning of apple, device 1 window 2 add sucessfully add berry to dictionary device 2 window 1 successfully update the new meaning of apple, device 2 window 2 successfully remove 'grape' from dictionary

Verify the server's ability to handle multiple requests concurrently using worker-pool architecture.

1. **400 concurrency, 4 workers**:
   * **Errors**: 2 errors were encountered out of 400 requests. This is a 0.5% error rate.
   * **Average Response Time**: 7ms
   * **Analysis**: With 400 concurrent requests and only 4 workers, it's notable that the system was able to handle the load with a very low error rate and a reasonable response time.
2. **350 concurrency, 4 workers**:
   * **Errors**: 0 errors
   * **Average Response Time**: 6ms
   * **Analysis**: Reducing the concurrency by 50 requests resulted in no errors, with a slightly improved response time.
3. **350 concurrency, 3 workers**:
   * **Errors**: 0 errors
   * **Average Response Time**: 6ms
   * **Analysis**: Despite reducing the number of workers by one, the system was able to handle the same load (350 requests) without any errors and with the same average response time as with 4 workers. This suggests that the system is efficiently distributing tasks among the available workers.
4. **350 concurrency, 2 workers**:
   * **Errors**: 10 errors out of 350 requests, which is approximately a 2.86% error rate.
   * **Average Response Time**: 4ms
   * **Analysis**: Reducing the worker count to half while maintaining the same level of concurrency resulted in a higher error rate. Interestingly, the average response time improved, which might suggest that while the system can process requests faster with fewer workers, it becomes more prone to errors due to overloading.

**Conclusion**:

* The system seems to handle concurrent requests quite well, especially considering the hardware specifications you provided (i3 CPU, 8GB RAM).
* The error rates are relatively low, even under high concurrency, which is a good sign.
* The average response time remains consistent and relatively low across different configurations, which suggests that the system is efficiently processing requests.
* The experiment results show that there's a balance to be struck between the number of workers and the level of concurrency. Too few workers can lead to errors under high concurrency, even if the average response time might improve.

Overall, based on the provided results, the system appears to meet expectations for handling concurrent requests efficiently using a worker-pool architecture. However, it might be beneficial to further explore the cause of the errors, especially when using fewer workers, to optimize the system further.

1.2. Error Handling:

Test the system's behavior when the client sends an invalid or malformed request.

**Send : "invalid{json" , response: {"description":"Invalid JSON format.","status":"error"}**

**Send : {"action": "search"}, response: {"description":"Missing 'word' field in search request.","status":"error"}**

**Send: {"action": "unknownAction", "word": "example"}, response: {"description":"Unknown action: unknownAction","status":"error"}**

**Send: {}, response: {"description":"Invalid JSON format.","status":"error"}**

Test server's response when the database encounters an error.

Delete/rename/read-only while server is running

Send: **search works fine because we have hashmap, add/remove/update will response a database error**

Test client and server behavior on unexpected disconnects or crashes.

Future work: To ensure that both the client and server can handle unexpected disconnections or crashes without causing system-wide issues.

1.3. System and Report Compliance:

Ensure the report is in the specified PDF format.

Validate the directory structure and naming conventions.

Check for timeliness in assignment submission.

2. Functional Tests:

2.1. Query Word:

Query a word that exists in the dictionary and validate the server's response.

Query a word that does not exist and check for the appropriate error message.

Test edge cases: special characters, very long words, or empty strings.

Special characters: **@#$%^&\*()!, success**

**Empty strings: enter a space, response client not just enter space**

2.2. Add a New Word:

Add a new word with its meaning and verify its addition.

Try adding the same word again and check for a duplicate error message.

Add a word without any meaning and verify the error response.

Test adding a new word from one client and querying it from another client.

2.3. Remove an Existing Word:

Remove a word that exists and validate the server's response.

Try removing the word again and check for a "not found" message.

Test removing a word from one client and querying it from another client.

2.4. Update Meaning:

Update the meaning of an existing word and validate the server's response.

Update the meaning of a word that does not exist and check for a "not found" message.

Test updating a word's meaning from one client and querying it from another client.

3. Excellence and Creativity Tests:

3.1. User Interaction:

Validate proper user feedback during errors to understand what went wrong.

Check for meaningful notifications and prompts in the client application.

3.2. Advanced Features:

Ensure the usage of a custom thread pool (not Java’s built-in) for server concurrency.

Check for any additional enhancements, like performance optimizations or security features.

Future work: security features like SQL Injection(like send a ‘DROP TABLE dictionary’)

3.3. GUI and Additional Features:

If a GUI is implemented, test its functionalities and responsiveness.

Test any additional features that have been added beyond the basic requirements.

4. Miscellaneous Tests:

Test the system under load, with multiple clients making simultaneous requests.

High concurrency test done before

Check the system's behavior under prolonged usage.

Future work: due to time limitations

Test the system's behavior in different network conditions, like high latency or packet loss.

Future work: **Test the system's behavior in different network conditions, like high latency or packet loss, due to limitation of knowledge so not implemented**