Mini Project-1 :Tri-code Communication

Team: Code Heist

Sri Charan Reddy Mallu 017419779
Mahek Virani 017446936
Naga Varun Bathina 017434261

Introduction:

As part of this project, we've implemented comprehensive strategies for code hardening, error handling, and edge case management. We meticulously recorded performance and memory metrics for client-server interactions, both intra- and inter-language. Additionally, we utilized a suite of linters, analysers, and sanitizers for Java, C++, and Python to ensure code quality and security.

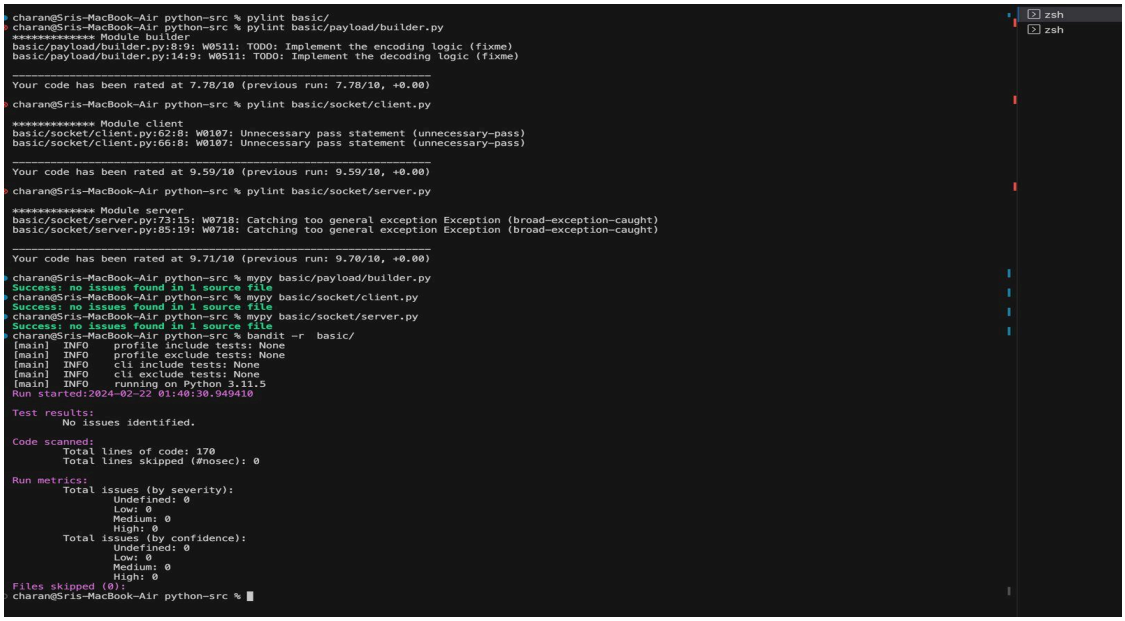
Language and Tools:

The Socket - Tri-Code Interoperability project necessitates proficiency in multiple programming languages and the adept utilization of various tools and libraries. We set up our code in C/C++, Java, and Python, along with familiarity with essential development tools such as `cmake`, linters for C/C++ ,Java and Python and text editor VSCode.

Intra-language communication pathways:

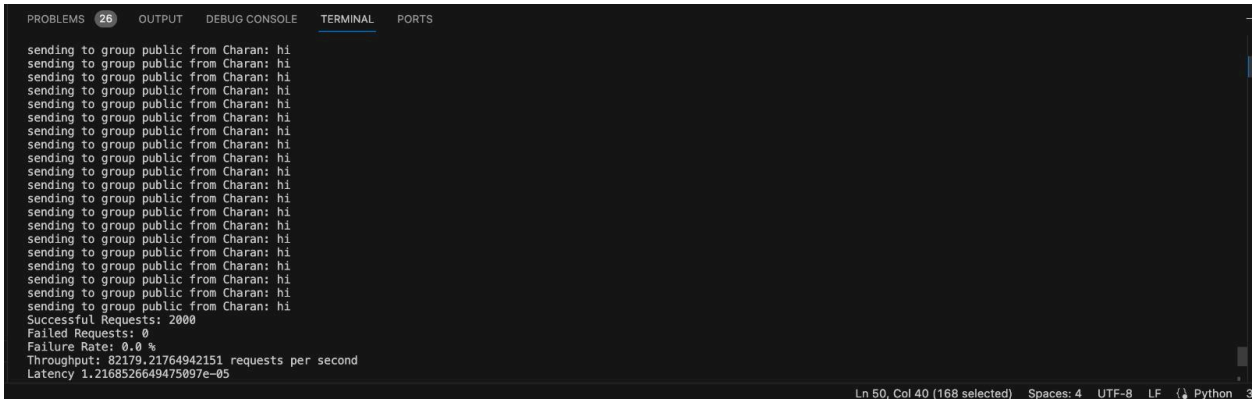
Python contribution: Mahek Virani

Initially, we established a communication channel between clients and servers within the same language. Mahek has commenced cleaning up the Python code, utilizing Python Linters. These linters are tools designed to scrutinize Python code to pinpoint possible errors, breaches of coding standards, and various other concerns. They operate by interpreting Python code and enforcing a predetermined set of rules or standards to detect parts that might need improvement.



The communication between Client and Server of Python intra-communication and the **latency** measured for a single message is as below:

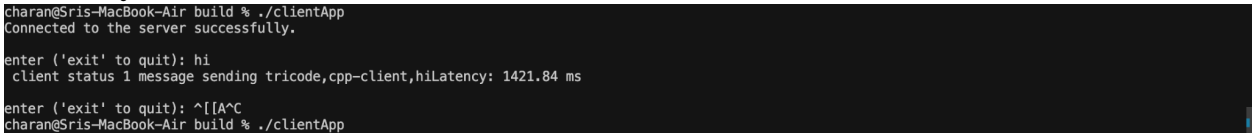
To measure the performance, we calculated **Throughput** by calculating number of messages communicated in an interval of time i.e. Number of messages divided by difference of start timestamp and end timestamp in Milliseconds.



C++ contribution: Sri Charan Reddy Mallu

In C++, Charan has undertaken the task of intra-process communication, addressing and rectifying code anomalies, and enhancing code quality through the use of Linters. The installation of cmake in C++ was executed as described:

Latency is calculated within intra-communication as below:



Linters have been utilized in order to clean-up the code and organize as below:

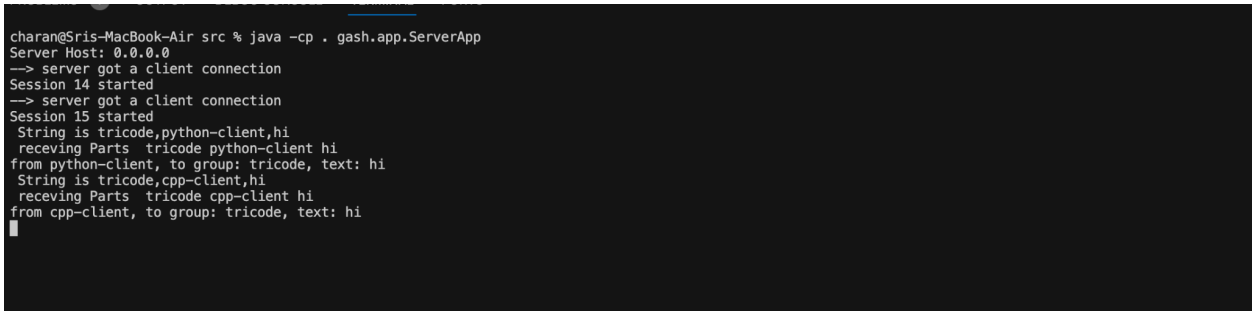
Java contribution: Naga Varun Bathina

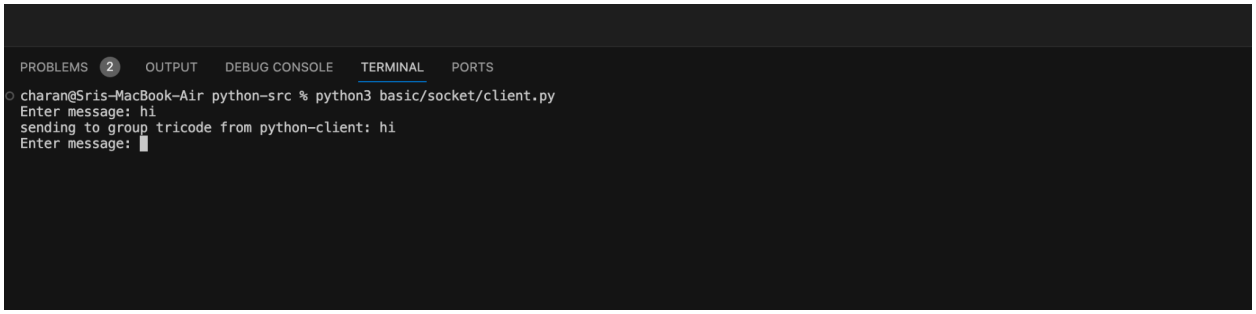
Varun began working with Java. He found that a significant portion of the necessary code was missing in Java, so he started developing it. He also made modifications to specific snippets that were marked as TODO, focusing on encoding and decoding aspects. The communication between different sections of the Java code is outlined below:

Interlanguage communication pathways:

We took up the task of hardening the connectivity matrix, ensuring robust communication between client and server implementations across all language combinations. Emphasis is placed on writing high-quality code, identifying edge cases, and verifying interlanguage communication pathways.

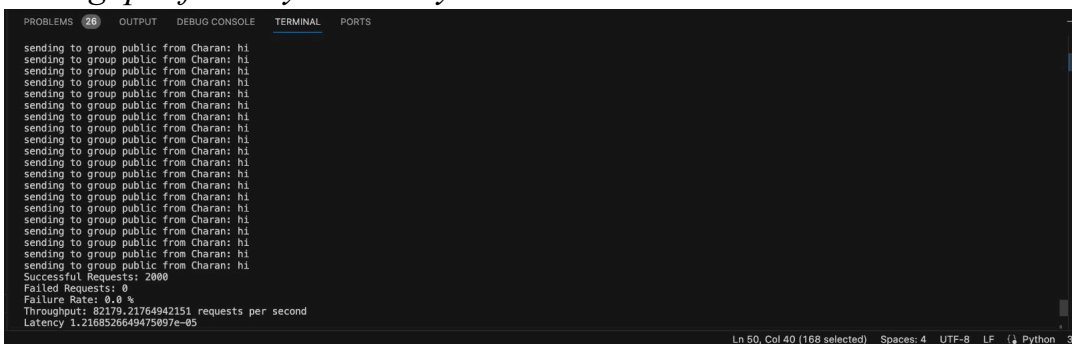
All the 6 directions of permutations of inter communications are achieved like Cpp to Java, Python to Cpp, java to Python:





Measurement of Performance and Throughput:
Performance metrics and throughput measurements are essential aspects of the project, providing insights into the efficiency and scalability of the implemented solutions. The main performance metric we utilized is the Throughput which is the measure of number of messages transferred within an interval of time.

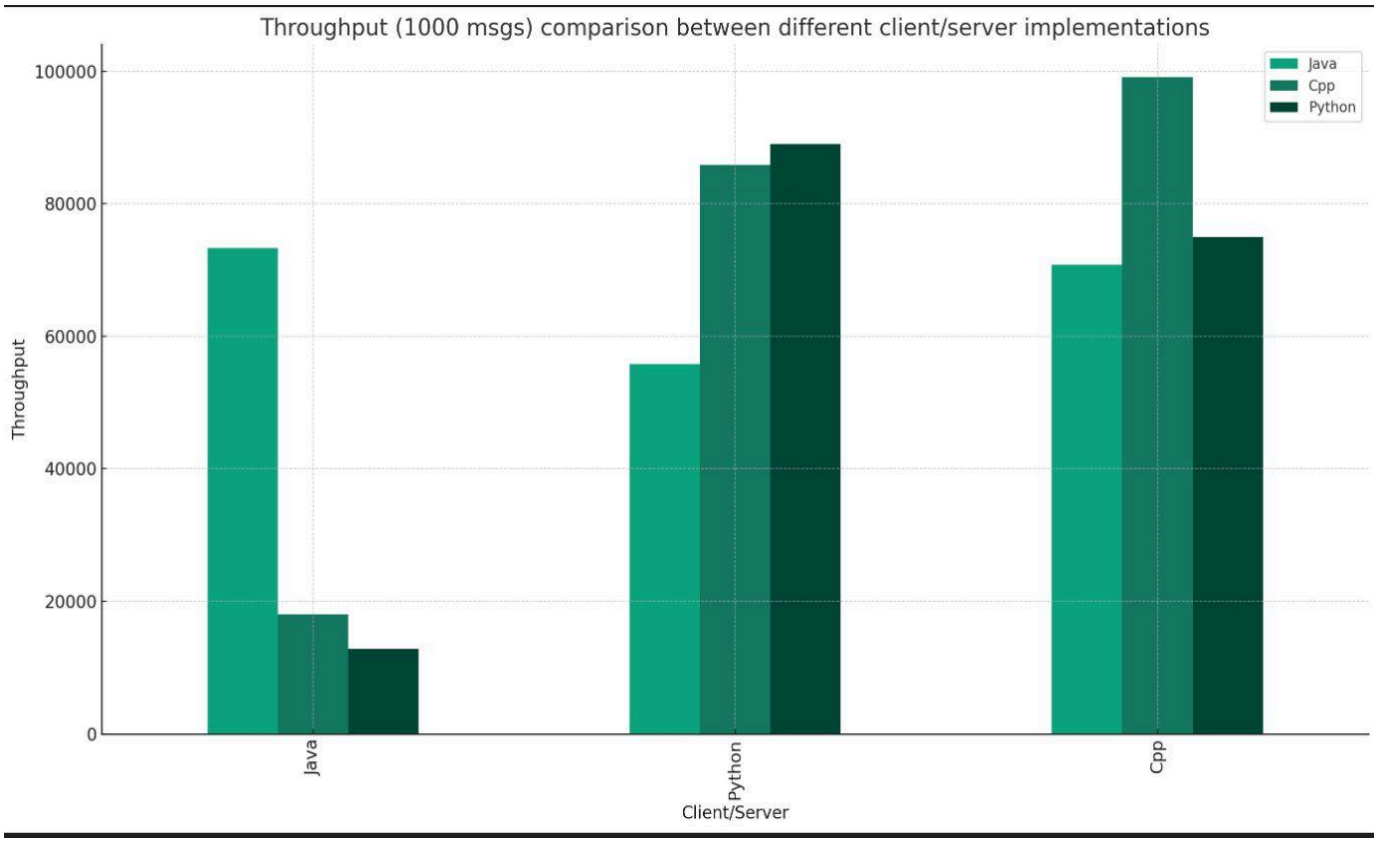
Throughput from Python to Python intra-communication:



Throughput from Java to Java intra-communication:

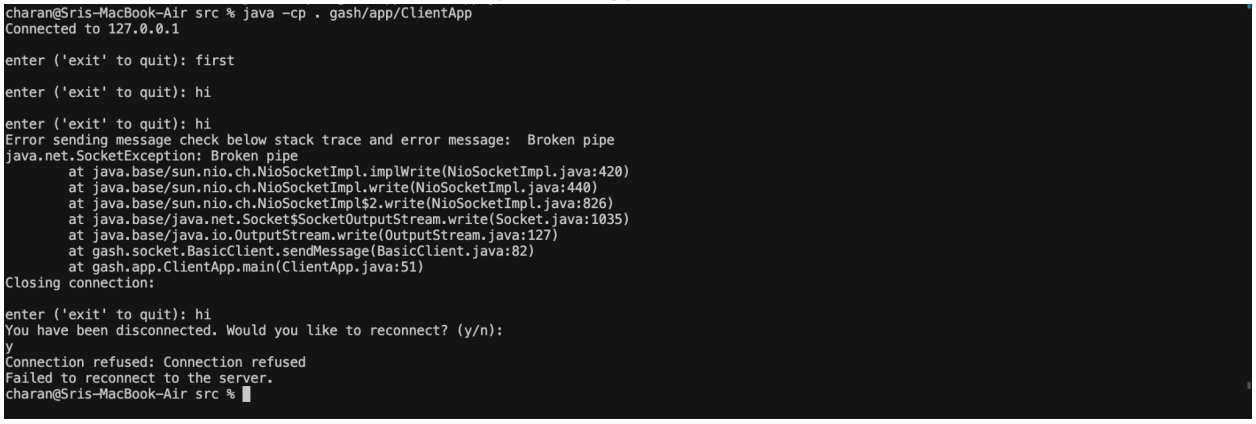
```
PROBLEMS 12  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

mahek@Maheks-Laptop basic % java -cp src gash.app.Test
Connected to 127.0.0.1
Average Latency: 33.0 ms per message
Throughput: 303030.30303030304 messages/second
mahek@Maheks-Laptop basic %
```

The bar graph for comparison within Java,Python and Cpp servers as X-axis and Throughput as Y-Axis. The 3 consecutive bars are 3 different clients Java, Cpp, Python respectively.



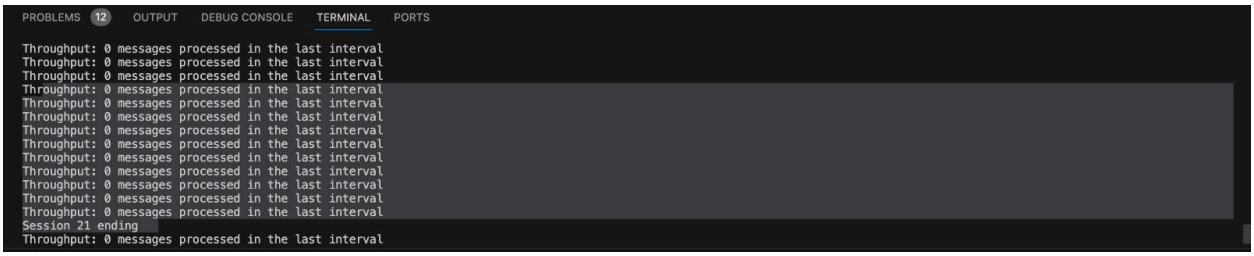
Exploration of Error Handling,Edge case Handling:
Beyond the core objectives, we explored additional challenges and considerations which led us to observe following issues such as:

- Implemented error handling for server shutdown scenarios to ensure proper client termination with retry strategy.



- Resolved the close_wait state issue in clients to maintain system stability.(Bytes read -1)

- Added sessions maintained and closed by server:



Limitations:

Limitation on Third-Party Libraries: The use of third-party libraries or services beyond those explicitly listed is prohibited.
Direct Use of Sockets: Participants are required to utilize the underlying socket mechanism directly, without resorting to higher-level abstractions or frameworks.
Adherence to Project Scope: The focus of the project should remain on achieving interoperability and optimizing performance, with minimal deviation towards feature enrichment or extraneous functionality.

Conclusion

In conclusion, the Socket - Tri-Code Interoperability project has been a rewarding journey, culminating in a deeper understanding of distributed computing concepts, effective communication protocols, and the challenges and intricacies of interoperability across diverse programming languages. Through collaboration, experimentation, and rigorous testing, we have laid a solid foundation for future explorations and innovations in the realm of distributed systems. The main takeaways for our team include the Error handling and handling various edge cases like Server crash and Client Terminate Ack issue, handling several exceptions. We tested with different payload lengths across the client and servers and handled accordingly. Utilized the tools such as Linters, Sanitizers for achieving well formatted code. The main aspect of Performance Measurement included noting down the values of Latencies of Intra and inter language communication and Throughputs of all permutable combinations among these 3 languages which gave certain useful insights such as the Python client to Cpp server gave highest throughput that clearly explains the Cpp fastness.