Evaluation of performance of GraphQL Subscriptions compared to WebSockets on single API

EASHIN MATUBBER ENRIC PERPINYÀ PITARCH
matubber | enricpp @kth.se

November 27, 2023

Abstract

This project report aims to describe how have we conducted the performance evaluation of GraphQL Subscriptions compared to WebSockets on a single API endpoint. The key metrics have analyzed are requests per second and latency. The outcomes of this research will help developers and organizations in making informed decisions regarding real-time communications and optimizing the performance of their applications.

Contents

List of Acronyms and Abbreviations

API Application Protocol Interface

REST REpresentational State Transfer

1 Introduction

The need for effective real-time communication solutions is a perennially significant issue in modern software development. This study carefully attempted to assess the relative performance of WebSockets and GraphQL Subscriptions on a single Application Protocol Interface (API) endpoint, focusing on important parameters such as latency and requests per second. Based on the hypothesis indicating possible WebSocket superiority, the study topic seeks to determine handling capacity, measure latency, and identify correlations in GraphQL Subscriptions. The research methodology, findings, and a brief commentary are all covered during the research and its experiments. When navigating the world of real-time communication solutions, developers and organisations may use this study as a resource and may focus on further research on the topic.

1.1 Theoretical framework/literature study

As it was expressed by Lee Byron, Dan Schafer and Nick Schrock (co-creators of GraphQL) in the GraphQL documentary, it changed the way of doing calls to a service; or in this case to multiple services [1]. Many companies are using microservices architecture. If the client have to communicate with the multiple services one by one, as before, then it the client side would go slower and more difficult to maintain as you need to map all the calls. However, with GraphQL as it is only one endpoint that collects all the responses and handles them to the client, it makes it easier use-cases of an API for the client.

Several searches in databases like IEEE, Aalto Primo and KTH Primo or search engines like Google Scholar have been made in order to find out if there is any paper related to GraphQL Subscriptions and its performance. Unluckily, under the terms ”GraphQL Subscriptions”, ”GraphQL subscriptions performance”, and ”GraphQL Websockets” there are no papers found which talk about the specific performance of GraphQL Subscriptions. It may be possible that those papers exist or are currently being written but those are not known by the authors of this proposal.

Nevertheless, there are a few papers and articles which talk about the performance of GraphQL Query compared to REST. Some of those papers, express different results, some arguing that GraphQL outperforms REST and other that REST outperforms GraphQL. This is as well one of the motivations of this research, but with subscriptions and websockets.

Several papers talking about GraphQL and comparisons between REST and GraphQL have been read. In the following lines there are a brief description and results and why are they interesting in the project.

In the *APIs with GraphQL* paper by Jeff Doolittle [2], it discusses about what is GraphQL and its ecosystem. It explains how Subscriptions work under the hood, using WebSockets, and different techniques on how to optimize for client queries. He states that GraphQL can be use for anything ”that is actually a back-end store of data”, even though in [3] it is argued it is not a good idea for databases for example. This article is relevant to the research as it gives a general overview of key points to build a GraphQL API.

In *An Overview of GraphQL: Core Features and Architecture* by Vlatko Spasev, Ivica Dimitrovski, and Ivan Kitanovski [4], it presents the core features of GraphQL and how some big companies like Facebook, Coursera and Twitter have moved to GraphQL in order to reply to client’s queries. This article explains few interesting facts about how GraphQL works inside with the the semantic trees or a comparison of how it works compared to REST. This paper is relevant as it explains key nuances of the internals of both protocols, which it may impact significantly in real-time response.

In *comparative analysis of web application performance in case of using rest versus graphQL* by Milena Vesić and Nenad Kojić [5], it presents a comparison of performance between GraphQL and REST. In their paper, they compare 2 applications one made with Node.js for GraphQL and another made using PHP for

REST. It is arguable how they implement the REST client and server as the results are surprising compared to many other research paper. They found out that GraphQL can be up to 2000 times faster in some cases like the ”Execution time of requests for different flow rates with 1000 records per table” in which at 36.59 Mbps they found out that REST replies in 1.1 min and GraphQL in 352ms. Even though they do not talk about the concurrency, this results can be due to REST replying one by one to the request and GraphQL replying all at once.

In *An evaluation of Falcor and Relay+GraphQL* by Mattias Cederlund, it compares the Falcor [6], which it is JSON based query language made by Netflix, with GraphQL+Relay, which it was the way to develop applications with GraphQL in 2016. This paper is being read at the moment and in the following drafts it is expected to be finished the reading.

1.2 Research questions, hypotheses

The primary objective of this project is to create several plots that effectively illustrate the performance differences in various scenarios. The intention is for readers to gain a clear understanding of the implications of using WebSockets and Subscriptions by simply glancing at these graphs.
The research questions we are investigating are as follows:

- How do GraphQL Subscriptions and WebSockets differ in performance, encompassing metrics like requests per second, latency, and the relationship between latency and request volume, in order to identify the most effective real-time communication solution for applications?

Our research hypothesis is that WebSockets will outperform GraphQL Subscriptions, as most of the implementations rely internally on WebSockets for communication. However, the performance impact per request is low enough that it doesn't justify changing the stack or having a mixed stack if you have already implemented GraphQL instead of REpresentational State Transfers (RESTs). In the case of really large applications, it is justified to use WebSockets to reduce infrastructure costs.

2 Method(s)

The study approach used for this project is empirical in nature, primarily concentrating on providing substantial experimental data to identify and compare the performance characteristics of WebSockets [7] and GraphQL[8] Subscriptions. Important sources of information, such Doolittle's investigation of ”APIs with GraphQL” [9], provide insightful viewpoints on the ecology and Subscriptions of GraphQL. This research shows how to shape the experimental technique, making it relevant and useful for examining how well GraphQL Subscriptions perform compared to WebSockets which helps to find the metrics to answer our research question.

The creation of two separate client-server architectures, one using GraphQL Subscriptions and the other WebSockets, forms a crucial part of the study design. The tools for gathering data include the framework that has been established, which takes an easy-to-understand approach to recording important metrics like latency and request frequencies.

Figure 1 shows the framework for easy and effective data transmission in real-time communication using the Client-Server-Receiver architecture. Three main parts make up this architecture: the Client, the Server, and the Receiver. They coordinate the flow of communication between them. With the Client creating and transmitting data, the Server overseeing communication, and the Receiver receiving and acting upon the provided data, this architectural paradigm guarantees a clear separation of responsibilities.

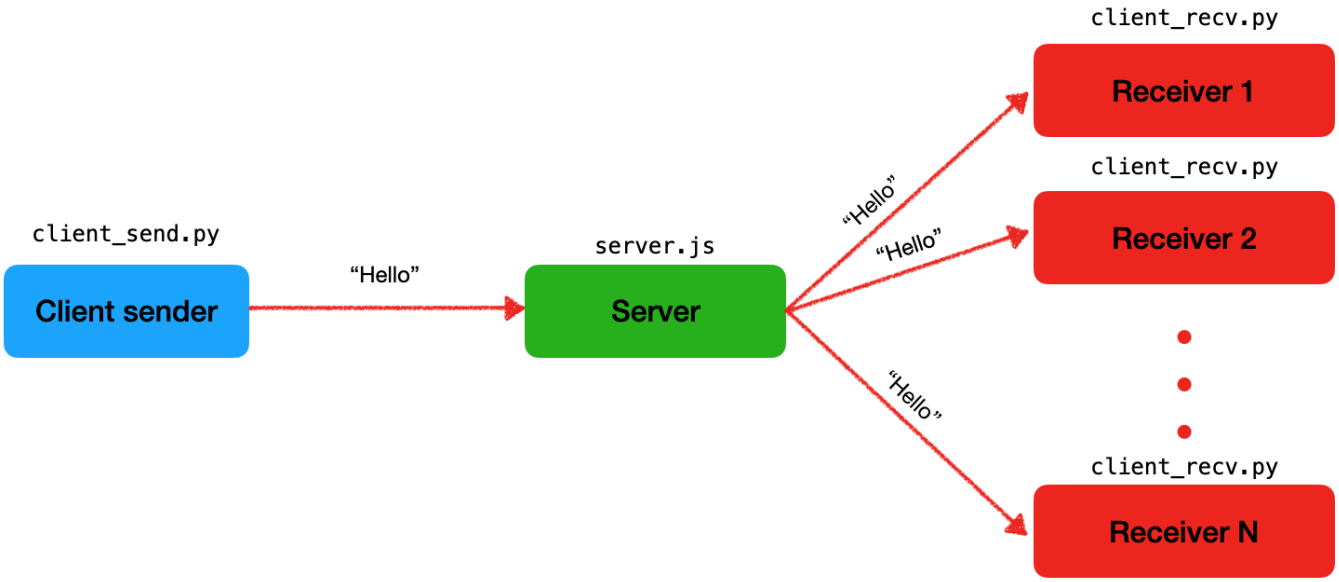

Figure 1: Client-Server-Receiver Architecture of real-time commmunication

2.1 Client Sender to Server

A methodological framework developed for the purpose of conducting experiments and extracting data in a methodical manner in the field of WebSocket performance assessment. a WebSocket connection to a local server with specified event handlers to control various aspects of the connection lifetime, such as error handling, message receiving, and closure protocols. In this scenario, random hash values are generated and sent to the WebSocket server while exact timestamps are meticulously recorded in tandem. This data transfer simulation mimics the actions of a WebSocket client running under heavy demand. The carefully recorded hash values and their corresponding timestamps.

In order to run experiments and collect data for GraphQL performance testing, random hash values are generated and transmitted as GraphQL mutations to a local GraphQL server via HTTP requests. This simulates data transfer. captures the temporal dimension of the communication by recording timestamps for every transmission. Hash values and timestamps make up the created synthetic data. By modifying the command-line options, the script may simulate numerous concurrent GraphQL clients thanks to the methodology's support for scalability. This methodology's main objective is to evaluate the GraphQL application's performance, with an emphasis on metrics like message transmission time. The generated data seeks to provide light on the effectiveness and scalability of the GraphQL implementation in different scenarios.

2.2 Server to Receiver Client(s)

Another technique for creating WebSocket connections to local servers, simulating several concurrent clients, and doing server-to-receiver client(s) tests and data extraction. Every client instance is linked to the server, waiting to receive messages from the WebSocket server. This allows each process to continually receive messages from the server. The timings list keeps track of every message that is received, along with the timestamp, until it receives a 'close' message, at which point the WebSocket connection is terminated. This concentrated on parameters like message reception time and possible relationships between total performance and the quantity of client instances. The experiment's scalability is improved by the use of multiprocessing, which enables the simulation of several WebSocket clients at once.

Server-to-receiver client(s) experiments provided another approach for carrying out tests and gathering data related to GraphQL subscription performance testing. Every client registers for a subscription query in GraphQL and watches for messages from the server in real time. The experiment is more scalable since the script uses multiprocessing to run numerous client instances at once. Captures the temporal dimension of

the communication by logging each message's content and time. Until a message stating "close" is received, the script keeps running and the experiment is not ended. With an emphasis on criteria like message receipt time and possible relationships between the number of client instances and performance, this technique attempts to evaluate the real-time communication performance of a GraphQL subscription system.

3 Results and Analysis

3.1 Server to Receiver Client(s)

Figure 2 "Average time: Server to Receiver Client(s)" describes the average amount of time needed for data to be transmitted from the server to many receiver clients in real-time applications. GraphQL is represented by the blue line, and Websocket by the yellow line. The metrics pertaining to both technologies are highly significant and highlight the need for more investigation, since the GraphQL application's serverside to all connected receiver clients communication is not as efficient as that of the WebSocket application.

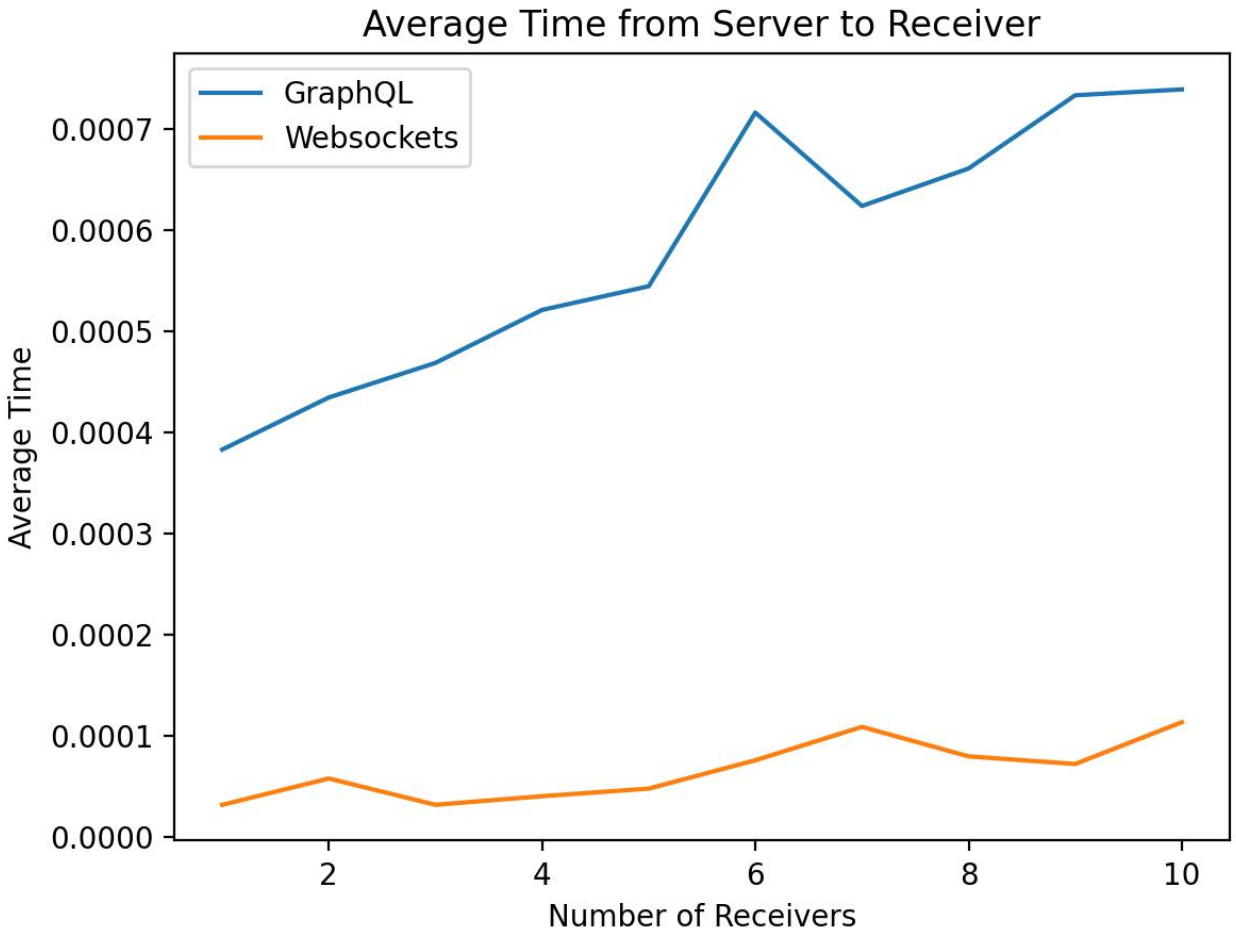

Figure 2: Average time: Server to Receiver Client(s)

Comparably, Figure 3's metrics "Standard Deviation time: Server to Receiver Client(s)" demonstrate how inconsistent the time it takes to send data from the server side to several receiver clients varies for the same connection. A higher standard deviation suggests higher variability and might be an indication of fluctuations or differences in the transmission times to different receiver consumers. Moreover, WebSocket performs noticeably better than GraphQL subscription and remains stable as the number of receiver clients increases.

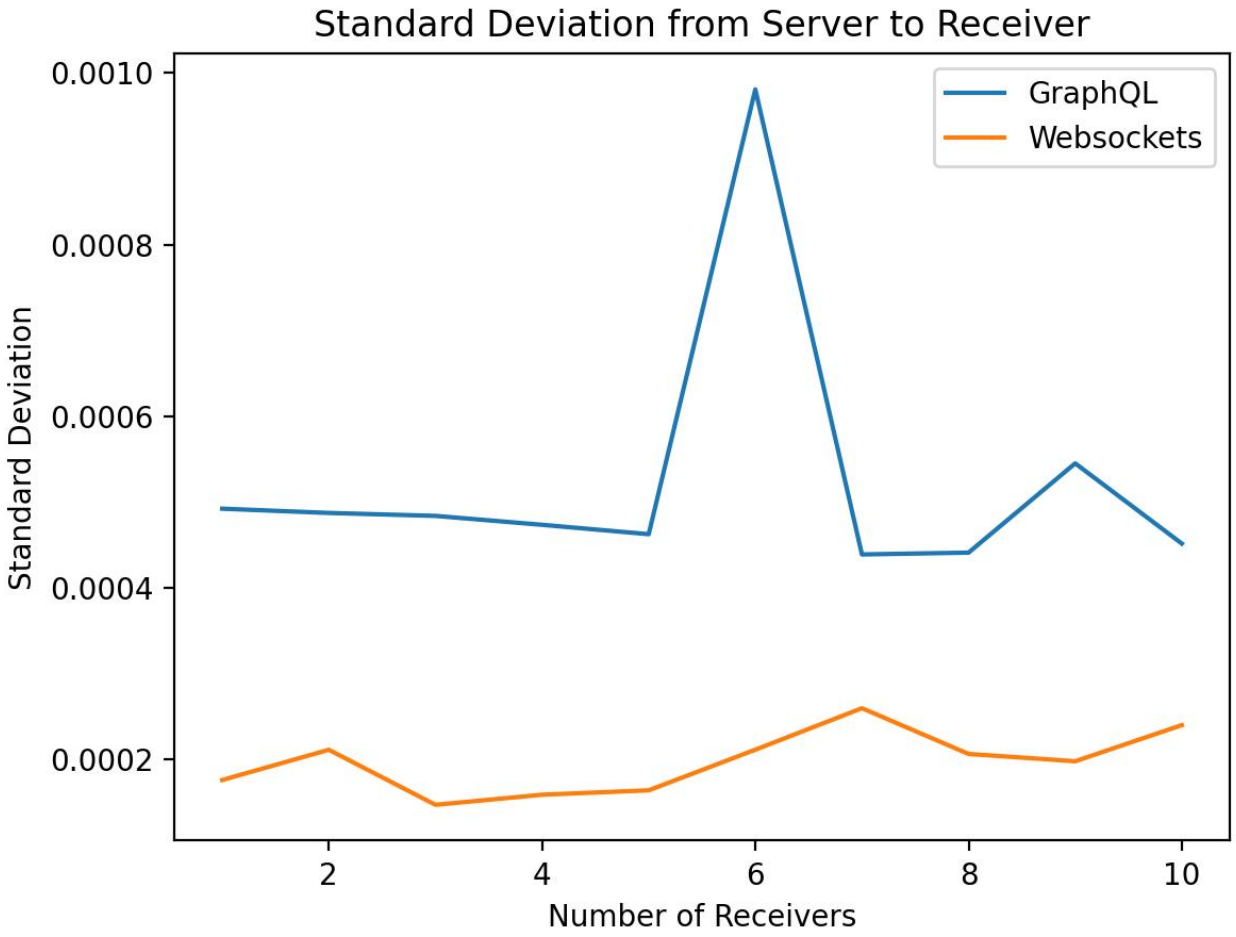

Figure 3: Standard Deviation time: Server to Receiver Client(s)

3.2 Client Sender to Server

The ”Average time: Client sender to Receiver Client(s)” in Figure 4, represents the mean duration taken for data transmission from the client sender to multiple receiver clients in real-time. This statistic evaluates the effectiveness of Websockets and GraphQL in sending data from the sender to numerous receivers in a timely manner, and it offers useful insight into the overall responsiveness and efficiency of these technologies. GraphQL certainly performs better than Websocket in this experiment.

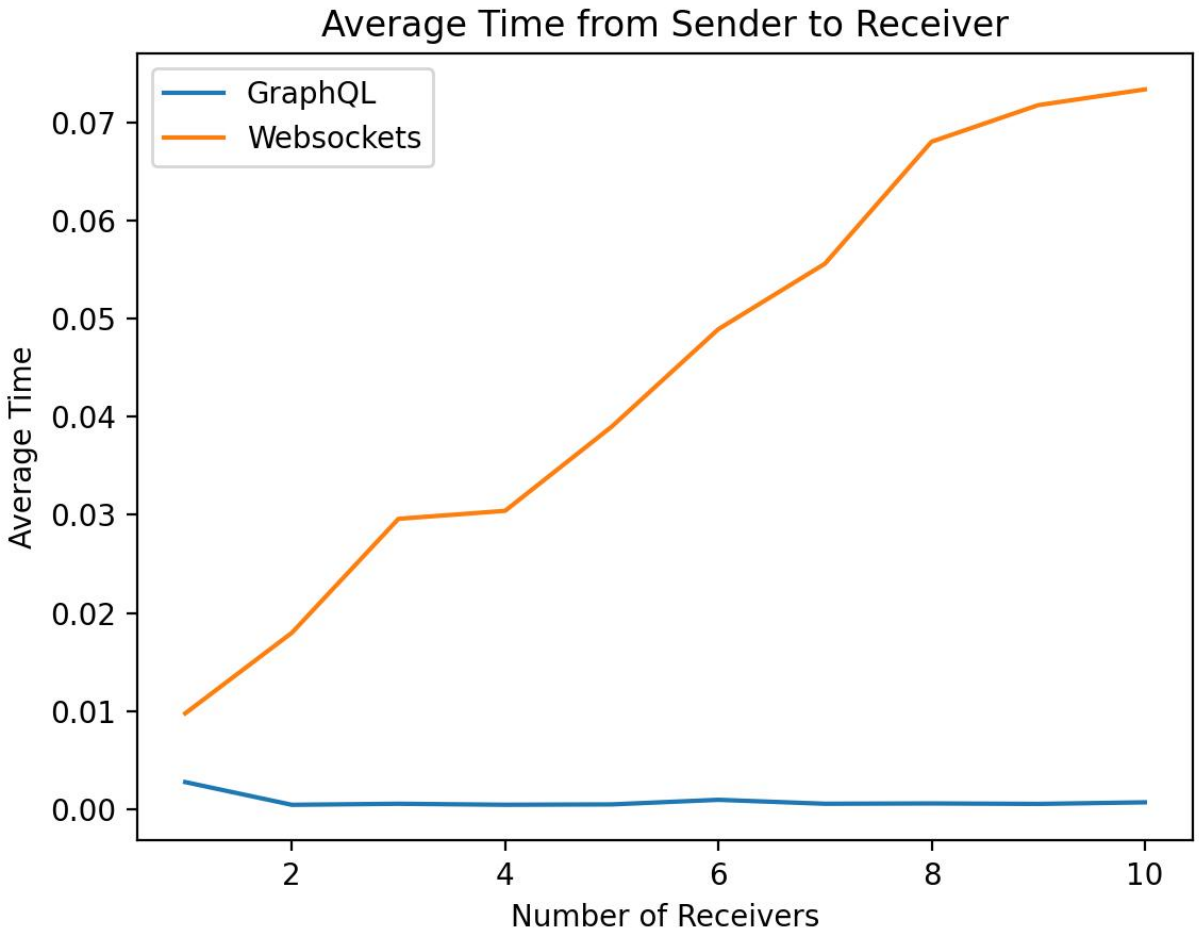

Figure 4: Average time: Client sender to Receiver Client(s)

The ”Standard Deviation time: Client sender to Receiver Client(s)” in Figure 5 describes a metric that quantifies the degree of variation or spread in a set of values and is used to measure the variability or dispersion in the time it takes for messages to travel from the sender client to the receiver client(s) in a real-time communication in both Websockets and GraphQL technologies. GraphQL has a distinctly efficient and consistent performance in sender-to-receiver client(s) communication, while WebSockets performance is relatively poor and inefficient.

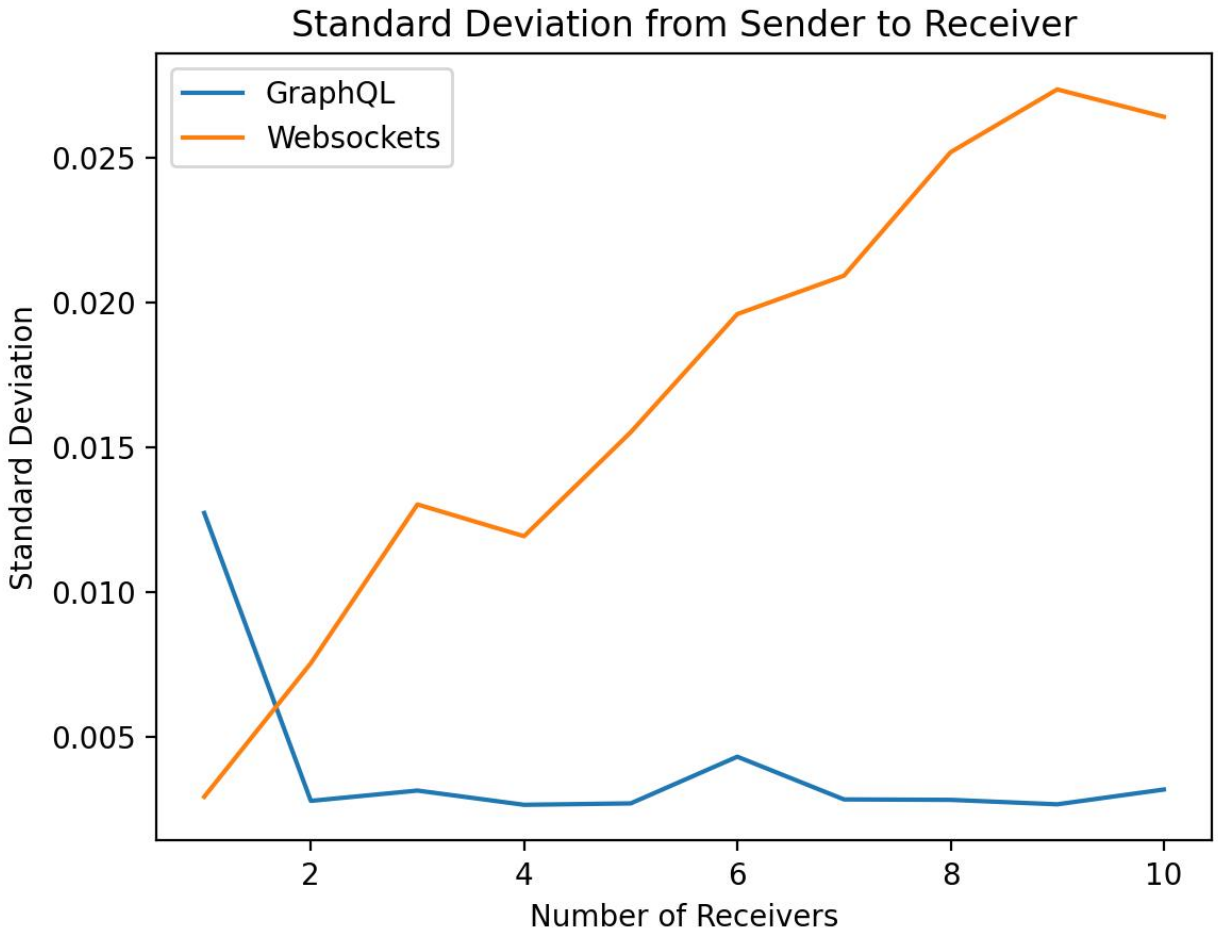

Figure 5: Standard Deviation time: Client sender to Receiver Client(s)

4 Discussion

In order to shed light on the differences in performance between WebSockets and GraphQL Subscriptions inside a single API context, the data must be interpreted and analysed. Notably, WebSockets perform better than GraphQL Subscriptions when it comes to the average time it takes for data to go from the server to recipient clients. In contrast to WebSockets, GraphQL performs more consistently and effectively in sender-to-receiver communication, according to the standard deviation study. This implies that GraphQL Subscriptions show dependability and stability in real-time communication contexts, even if WebSockets could offer some advantages.

In addition, it recognises the theoretical foundation while emphasising GraphQL's special qualities and possible influence on real-time communication. A thorough comprehension of the observed performance measures is provided by revisiting the study questions and hypotheses in light of the findings. Overall, the experimental results showed that without further thorough investigation and testing, it is difficult to draw a conclusion on the study subject.

References

[1] Honeypot. (Published on 2019-06-24) Graphql: The documentary. Accessed on 5 October, 2023. [Online]. Available: https://www.youtube.com/watch?v=783ccP__No8

[2] J. Doolittle, "Apis with graphql," *IEEE Software*, vol. 40, no. 2, pp. 118–120, 2023. doi: 10.1109/MS.2022.3227254

[3] S. Adams, J. Piotrowski, and N. Burk, "How we sped up serverless cold starts with prisma by 9x," *Prisma Blog*, April 2023, accessed on 8 October, 2023. [Online]. Available: https://www.prisma.io/blog/prisma-and-serverless-73hbgKnZ6t

[4] V. Spasev, I. Dimitrovski, and I. Kitanovski, "An overview of graphql: Core features and architecture," in *Proceedings of the International Conference on Information and Communication Technology*. Faculty of Computer Science and Engineering, Skopje, 2020, accessed on 7 October, 2023. [Online]. Available: https://repository.ukim.mk/bitstream/20.500.12188/19669/1/ICT_2020_submission_40.pdf

[5] M. Vesić and N. Kojić, "Comparative analysis of web application performance in case of using rest versus graphql," in *Proceedings of the ITEMa 2020 Conference*, 2020, p. 17, accessed on 6 October, 2023. [Online]. Available: https://www.itema-conference.com/wp-content/uploads/2021/03/0_Itema-2020-Conference-Proceedings_Draft.pdf#page=23

[6] M. Cederlund, "Performance of frameworks for declarative data fetching: An evaluation of falcor and relay+graphql," Master's thesis, Stockholm, Sweden, 2016, accessed on 4 July, 2016. [Online]. Available: http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1045900&dswid=-189

[7] "WebSockets Standard," 9 2023. [Online]. Available: https://websockets.spec.whatwg.org/

[8] "Introduction to GraphQL — GraphQL." [Online]. Available: https://graphql.org/learn/

[9] J. Doolittle, "APIs with GraphQL," *IEEE Software*, vol. 40, no. 2, pp. 118–120, 3 2023. doi: 10.1109/ms.2022.3227254. [Online]. Available: https://doi.org/10.1109/ms.2022.3227254