A decidir

Tiago da Silva Guerreiro and Lucas Costa dos Prazeres

Abstract

Keywords

I. Introduction

II. Technical Background

This article's multiplayer game is designed in Javascript, using Node.js in backend, and Socket.io to establish Websocket communication.

A. *Websocket*

Websocket provides bidirectional full-duplex communication over a single TCP connection. Unlike HTTP, the websocket connection remains open until terminated by the server or client, which reduces the number of handshakes. Since the connection is already open, data transmission is faster.

The websocket technology is crucial to applications that require low latency, such as multiplayer browser-based games and streaming services.

B. *Node.js*

Node.js is a server environment that enables Javascript on the server side, without a browser. It uses an event-driven, asynchronous I/O, which makes Node.js capable of processing various requests simultaneously, and consequently, highly scalable.

Node.js provides a low cost, highly scalable environment for server-side applications. This technology is very useful on streaming web applications, real-time collaboration tools, online chats, microsservices and multiplayer games, since the server has to handle multiple requests from clients.

Special thanks to professor Eduardo Cerqueira, ITEC, UFPA, Belém-PA, e-mail: cerqueira@ufpa.br

C. *Socket.io*

Socket.io is a Javascript library that enables Websocket capabilites in Javascript. With a few lines of code, an application can start listening to Websocket connections on the desired port. It is a very powerful tool to applications that depend on fast exchange of data, like multiplayer games.

III. Game changes

IV. Method

One essential network requirement of a multiplayer game is the ability to handle many connections (from many players) as gracefully as possible, while providing a good real-time game experience. This is why it is important to observe how time-related parameters behave on such applications, like delay or latency. This article manages to study how this temporal characteristic responds to linear variations on the number of active connections in a small experiment.

The experiment consists of a series of matches played with different numbers of clients and a single server hosting the game. The server is a common laptop running *Ubuntu Linux* with the backend game instance listening on localhost on port 3000. Since localhost applications usually are not acessible to hosts on different LANs, we used a port forwarding tool called *ngrok*, which generated a tunnel using a USA located server to generate a public url to access the game server. Then, a few regular matches were played by clients located in different LANs - unlike the previous study [], in which the matches were played in a single LAN - while a native node.js function called *process.hrtime()* collected temporal benchmarks in background showing the average time it took for the server to respond to every player move. The linear relation between both parameters was plotted using the Python library *matplotlib* and are presented in the next section.

V. Results

The delay (or latency) is the time between an action and a system's reaction to it. In this particular context, it means the time it takes for the game server to notify all players when something changes on one of its clients, which could be a player moving, a fruit being removed or any other event triggered by an user interaction, from the moment an input is inserted on the keyboard. Since this response time depends on the whole network conditions from one host to another, it suffers variations even in similiar or identical game state scenarios, which brings the necessity of using statistical formulas like average or standard deviation to normalize the measured points. These techniques were applied to the experiment presented here and the results are depicted above.

Moreover, is important to point that this study faced a few challenges in terms of resources. For instance, all connections needed to be active and generate a sufficient data flow during the match, which means someone actually playing the game, in opposition to an idle client, and the number of available volunters has limited how many samples could be collected on the experiment, making it difficult to analyze parameters other than the game latency. Nevertheless, despite of this obstacle it was possible to observe an increasing pattern on the latency measured with a linear variance on the number of players, as shown in Fig.1.

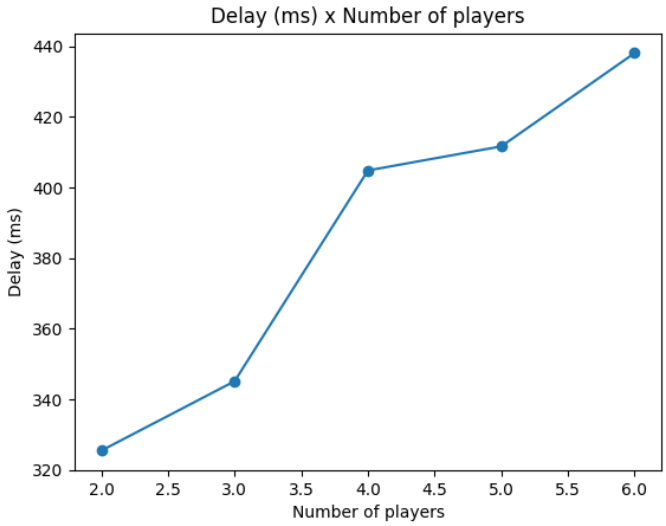

Fig. 1: Game latency with linear increasing on number of players

The results depict, as mentioned, a small incresing on the game latency, everytime a new connection is stablished with the server. This was the expected behavior, after all each new client means a new packet being sent though the network and due to the queue schema adopted by routers to handle priority of packets, in which the number of elements grows linearly, this also represents a similar raising pattern on latency. However, the fast data flow provided by the websocket protocol (as mentioned on section 2) attenuates the incresing on the response time, which could be significantly higher if each packet was followed up by a handshake operation as it happens with regular HTTP transmissions.

VI. Conclusion

The higher demand for real-time web applications for business purpouses, like fast collaboration tools, and also for the entertainment of users, as it happens with streaming services and multiplayer games was followed by an emergence of platforms and programming tools which support these needs and the usage of the nodejs ecossytem along with the agile websocket protocol has proved to be a strong alternative in such scenarios.

[1]

REFERENCES

[1] B. Chen and Z. Xu, "A framework for browser-based multiplayer online games using webgl and websocket," in *2011 International Conference on Multimedia Technology*. IEEE, 2011, pp. 471–474.