# Benchmarking a Minecraft-like game: Terraria

Anmol Bhatia
2739180
VU Amsterdam
a.bhatia2@student.vu.nl

Abhilash Balaji
2735067
VU Amsterdam
balaji@student.vu.nl

Sven Lankester
2668125
Vu Amsterdam
s.lankester@student.vu.nl

Leroy Velzel
2569971
VU Amsterdam
l.q.velzel@student.vu.nl

Aditya Srivastava
2775769
VU Amsterdam
a.n.srivastava@student.vu.nl

## ABSTRACT

Terraria is the most positively reviewed video game of all time on Steam. Its main characteristic is that a player can interact with the in-game environment to modify it as desired. Another critical aspect of the game is that it can be played online in real-time with other players. Combining these two traits leads to a problem: the large amount of information that results from player interactions with the world needs to be spread to all players, causing these types of games to have notoriously poor scaling. In this paper, we adapt the core ideas of Yardstick by using an existing bot framework to develop a tool that allows us to benchmark Terraria servers. The tool will enable us to deploy custom workloads on an unmodified Terraria server and gather system- and application-level metrics.

**ACM Reference Format:**
Anmol Bhatia, Abhilash Balaji, Sven Lankester, Leroy Velzel, and Aditya Srivastava. 2022. Benchmarking a Minecraft-like game: Terraria . In *Distributed Systems 2022/2023 - VU Amsterdam, 2022, Amsterdam (The Netherlands).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The video game industry is one of the most popular forms of entertainment shown by its estimated 208 billion U.S. dollars in revenue for the year 2022 [1]. Looking at the number of concurrent players as reported by the most popular digital video game distribution service Steam, multiplayer online video games consistently have the highest number of concurrent players [2]. The large demand for multiplayer video games shows the importance of scalability. Several studies have been performed to analyze and improve the scalability of multiplayer games [3–5].

A canonical example of a popular multiplayer game would be Minecraft, the highest-selling online video game of all time [6]. Minecraft-like games are fundamentally different from most online games, as it allows the players to modify the in-game environment freely. These *modifiable virtual environments (MVEs)* face severe scaling challenges due to every modification to the environment having to be propagated to other players [7]. The lack of scaling capabilities can be seen in Microsoft's official server hosting platform, Minecraft Realms, which only allows up to 11 users to play concurrently on a single server.

Minecraft is not an outlier for the genre of MVEs either. We can see that this subset of games has significant interest by looking at another example: Terraria. It is the first game on Steam to reach 1 million positive reviews, whilst also having the highest number of total reviews of which 97.1% are positive [8]. Terraria is a game in which the player can, similarly to Minecraft, modify the terrain at will. A significant difference compared to Minecraft comes from Terraria's in-game environments being 2-dimensional and being limited in size, compared to Minecraft's 3-dimensional worlds which span millions of in-game units. Its multiplayer functionality relies on real-time interactions with the environment, thus server performance and scalability are of high importance if we wish to support a large number of people concurrently playing in the same environment.

Research on the scaling of MVEs is limited and often has Minecraft as its focus. However, a tool named Yardstick has been proposed and implemented to benchmark Minecraft-like games [9]. The existing implementation works with the Minecraft protocol and evaluates the performance of Minecraft. This study aims to create a Yardstick-like benchmark for the game Terraria, in which we generate and deploy realistic workloads on a Terraria server so we can extract and analyze performance data. The main reason we wish to evaluate Terraria is that despite its similarities in functionality to Minecraft, we expect its 2-dimensional limited-world nature to have significant implications on the scalability of a Terraria server compared to a Minecraft server.

## 2 BACKGROUND

### 2.1 Minecraft-like games

Minecraft is developed by Mojang Studios in Java. It was first made public in May 2009 and has since created different versions. In Minecraft, users may locate and extract raw materials, make tools and products, and construct buildings, earthworks, and primitive machinery in a pixellated randomly generated 3D world with a nearly unlimited area. The players live in an environment that is modifiable and visible to all other players on the server. One of the implications of having this MVE is that every player has to perceive


*Distributed Systems Project, 2022/2023, Amsterdam, The Netherlands*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

these modifications made by other users. This is one of the bottlenecks of MVEs in terms of scalability and performance. Minecraft is one of the biggest and most popular MVE, rating as the third most popular game on PC[10]. Minecraft is not just popular within the game community. A Google Scholar search reveals around 37.000 results for the search query "Minecraft". In comparison with other games in the same genre like Fortnite(12.200), Terraria(8.800), and, Roblox(3.880), Minecraft far out is the most prominent one.

## 2.2 Terraria

Terraria is a game similar to Minecraft. It also contains an MVE which is playable in both single-player and multiplayer modes. A huge difference between Terraria and Minecraft is that Terraria is 2D instead of 3D. Also, Terraria is more focused on exploring and battling and Minecraft is more focused on building and crafting. Although similar to Minecraft, the 2D limited-world nature of Terraria has significant implications on the performance and scalability of the Terraria server, compared to a Minecraft server.

# 3 SYSTEM MODEL

## 3.1 Overview

To define and model the operation of Terraria we use a system model similar to Yardstick's model of Minecraft-like services.[9] with a few modifications to validate the model specifically to Terraria. As Terraria is built on the XNA Framework, which was created as an offshoot of the .NET framework for game development. Thus we used the Mono Framework on Linux which itself is a software framework that Terraria runs on. Terraria does come with an officially released (vanilla) server binary bundled which we do use for our experiments. The first challenge we faced was choosing a version of the game, as Terraria had 3 major updates to the network protocol and we went ahead with v1.3.5.3 as it was the build with most network documentation and had existing community-made network protocol implementations for us to use as both reference and foundation.

*3.1.1 Network Protocol.* Network communication in Terraria is done via a series of Messages. It is not a request/response system. The client sends messages when it wants to notify the server of changes (the player moves, for example), and the server sends messages when it wants to notify the client of changes (another player talks, for example). Generally, when a client sends a message say for example a player joins, the server will take note of that information, and then echo that same information back to all connected clients.

*3.1.2 Terraria Server.* The Terraria server is where the main game loop is processed and client synchronization takes place. The rate at which the game loop is processed refers to the tick rate of the game and Terraria has a tick rate of 60hz, ie - the game logic is processed 60 times a second. Although the server tries to maintain 60 ticks a second, this cannot be guaranteed as the time to process the game loop is dependent on the number of clients connected to the server.

*3.1.3 Client.* The clients emulate a player's actions by opening a TCP client to handle client communication with the server using terraria's network protocol. This is where we have complete control over the exact actions to be executed and record latency.

# 4 DESIGN

In this section, we present the design of our bench-marking experiment for Terraria. We start by defining the requirements of the experiment, the high-level design of the setup, and the important configurations of the experiments.

## 4.1 Requirements

We begin by defining the requirements that our experiment will try to fulfill. These include the requirements applicable for all types of benchmarks and application-specific requirements.

*4.1.1 Requirements Applicable to Many Types of Benchmarks.* We focus on the most common ones [11]-

(1) R1 Fairness - The benchmark should provide a fair performance and scalability assessment for similar distributed systems.
(2) R2 Ease of use - The setup should be easy to set up, configure and use so that it is easy to adapt to the new and applicable systems.
(3) R3 Clarity - The bench should present results in a logically understandable manner.
(4) R4 Reproducibility - The bench-marking experiment should be programmatically reproducible and should give similar/-comparable results under similar conditions.

*4.1.2 Requirements specific to Terraria.*

(1) R5 Realistic Workloads - The workloads chosen for the experiment should be similar to that of real player behavior.
(2) R6 Relevant Metrics - The metrics chosen for the experiment should be able to describe the performance of the system as perceived by the players in reality.
(3) R7 Low overhead - The setup should be such that the application and system-level metrics can be collected without adding any recognizable overhead.

## 4.2 Design overview

Among different Terraria versions to choose from, the game build v1.3.5.3 has the most network documentation and an already existing community-made network protocol implementation[12]. To reduce the overhead of creating our own network protocol we used this version (R2, R4). The metric that we used in the bench-marking test is player latency. This latency is calculated by timing between a performed and perceived player action, this is the player-perceived latency and therefore relevant in our experiment (R3, R6). We picked one of the least intensive actions to time the latency (R7). We can simulate different workloads by simulating different actions by the bots deployed on the server. These different actions will generate different messages with different intervals. These accounts are valid accounts that can be executed by real players (R6).

## 4.3 Design: existing tools

To address the knowledge gap in the scalability and performance of MVEs, Van der Sar, et al., designed Yardstick [9] as a benchmarking


**Figure 1: In-game representation of a completed experiment-setup**

tool to analyze the performance of MVEs. Yardstick subjects a Minecraft server to workloads determined by the virtual world and a set of emulated players. Both are parameterizable, leading to infinitely many possible workloads. Yardstick monitors both the machine running the Minecraft server application and the machines that run the emulated players. Metrics such as CPU, RAM, network usage, tick duration, and the number of transmitted messages are monitored. After an experiment is complete, both raw and processed data are made available to the Yardstick user to analyze server performance.

# 5 EXPERIMENT

## 5.1 Design

During the benchmarking process, we subject a Terraria server to different workloads such as bots teleporting or bots walking within the game. The workloads and number of bots are both defined by users. Moreover, we have two types of experiments: local and distributed, depending on the fact that the server and bot runner services are in a single machine or separate machines over different availability zones. The purpose of different setups is two find two types of latency: processing and networking, which separately affect the game-play experience.

*5.1.1 Setup.* At the setup phase, we initialise our benchmark server, which will host the game and measure and store the data. The first two bots will be spawned in the game, these bots are the sending and receiving bots. The sender starts a timer, moves to a know location A if at position B and vice-versa then it waits for the receiver. This process is repeated until the end of the experiment. The receiver waits for the sender to teleport and then stops the timer. If the timer gets stopped the latency is logged.

After both sending and receiving bots have been spawned the workload bot will be created and spawn in the game. See Figure1. We have defined connect these workload-bots in batches to avoid a spike in the server load. After spawning the workload bot, it will start executing the workload defined by the configuration.

*5.1.2 Workloads*
. We have created three different workloads, we used these workloads because we thought that this simulates the player's behaviour around the map. These workloads do have an impact on the server because these changes have to be sent to all players.

- The teleport workload: teleports the bot to a new location every x millisecond(s). This x is chosen randomly between 0 and 1000. After this interval, a new random interval is chosen. The new location of the bot only differs in the x-coordinate. A random number between -200 and 200 is chosen to update the current x-coordinate.
- The walking workload moves the bot to the left or right for a random interval between 2000-4000 milliseconds. If the interval has ended the bot will walk in the opposite direction for a new randomly chosen interval between 2000-4000 milliseconds.
- The modification workload selects a random tile in the range of the bot then it tries to place an item on this tile or tries to remove the selected item on this tile. This will be repeated at an interval randomly chosen between 2000-4000 milliseconds.

*5.1.3 Experiments*
. To get a global view of the measured benchmarks we have tested our tool using two different experiments:

- Experiment 1: A local experiment where the setup and the workload are run to the same benchmark server.
- Experiment 2: A distributed experiment where the setup runs on the benchmark server and the workload is connected to the benchmark server from a separate server.

The reason why we test our experiment locally and distributed is to measure the processing latency with and without the network latency. In the local experiment, the network latency can be neglected, but for the distributed experiment we expect to measure network latency.

*Experiment 1: The local experiment.* This experiment is run on a single machine we host the game and run all the (measuring and workload) bots from the same server.

*Experiment 1: Hypothesis.* The latency of Terraria may be affected by different workloads types and sizes. An increase in the number of connected clients will lead to more communication between the server and the end user. This will therefore increase the latency of the game and will be shown in the measured latency. For different workloads we expect different latency increases, the reason for these differences, we assume that for the walking workload, the server and the client do communicate more intensely than for the teleport workload.

| ID | Bots | Workload | B. size | B. delay | Time | Metric |
|---|---|---|---|---|---|---|
| 1LA | 50 | teleport | 3 | 1500ms | 8min | Latency |
| 1LB | 50 | walking | 3 | 1500ms | 8min | Latency |
| 2L | 105 | teleport | 10 | 20000ms | 4min | Latency |

**Table 1: Different local workloads**

*Experiment 2: The distributed experiment.* This experiment is run on multiple machines, however, we made a distinction between measuring and workload bots. The game is hosted on the benchmark server, this server also runs the measuring bots. As for the workload bots, we evenly separated the amount across 2 different servers.

*Experiment 2: Hypothesis.* We expect the same hypothesis for experiment 2, however, we expect that the latency will higher because of the overhead in communication between the workload bots. The benchmark server will be less occupied because the workload bots will be run on a different server, this could have a positive effect on the latency. But we expect the network communication overhead will be the leading factor in determining the overall latency. Therefore making the latency of experiment 2 higher than experiment 1.

| ID | Bots | Workload | B. size | B. delay | Time | Metric |
|---|---|---|---|---|---|---|
| 1D | 24 | teleport | 3 | 60000ms | 8min | Latency |

**Table 2: Different distributed workloads**

## 5.2 Experiment setup

For every experiment, we made sure we had a Terraria server of version v1.3.5.3 running which was only accessible by the IP addresses of the workload bots. Incoming traffic from other IP addresses would therefore be ignored. We killed any other non-trivial processes to make sure these processes do not interfere with our results.

*5.2.1 Local experiment.* In this experiment, we measured the performance of Terraria by taking the latency over a period of 10 seconds, with different configurations mentioned in Table 1. We used a local machine with the following specifications:

- CPU: AMD Ryzen 7 3700X
- RAM: 16 GB DDR4
- GPU: NVIDIA GeForce RTX 2070 Super Gaming X
- OS: Windows 10

*5.2.2 Distributed experiment.* The different configurations used in this experiment can be found in Table 2. Every experiment is run on an AWS EC2 instance using a t2.large instance [13]. Every instance has two virtual CPUs, 8 GB of ram, and is running a Linux Operating system. The sending and receiving-bot are connected to the server using our local machine and the bots are connected to the server using one t2 instance. A visual representation of the setup can be found in Figure 2.
The EC2 server instance has - terraria server, Prometheus, and Cadvisor, the latter two for measuring the system level metrics like - memory usage, CPU cores usage, and network receive and transmit bytes.

# 6 RESULTS

## 6.1 Results analysis

As we see from our results we find that we notice differences even at 1 million nanoseconds = 1 millisecond. We found this surprising because the Terraria process updates at a ratio of 60 Hz, which is around 16,7 milliseconds.

*6.1.1 Local experiment.* Figure 3 has large outliers, from our data we determine this to occur when our bot client is connecting the new batch of bots. This leads us to believe the outliers are caused by the initialization of the new TCP clients and thus a result of client-side processing delays. For that reason, we remove the aforementioned outliers in Figure4 to better visualize the server-side latency.

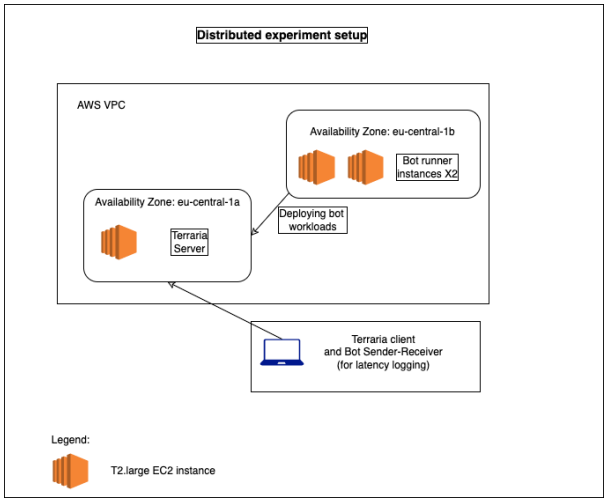


**Figure 2: Setup for the distributed experiment.**

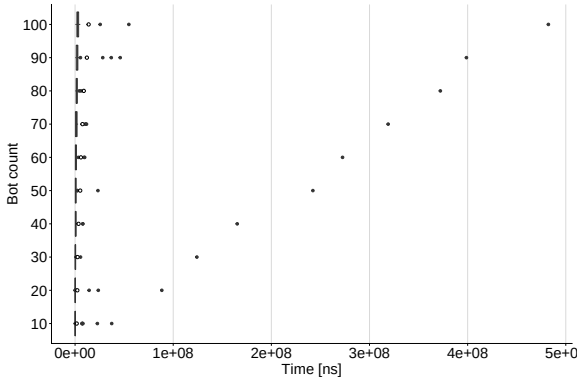


**Figure 3: Results of experiment *2L* (105 bots, local, teleport).**

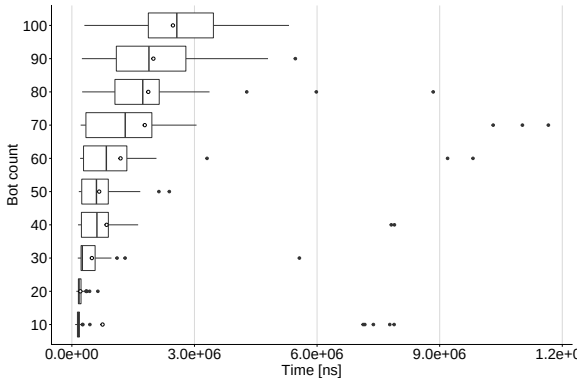


**Figure 4: Results of experiment *2L* (105 bots, local, teleport) with outliers greater than 13ms removed.**

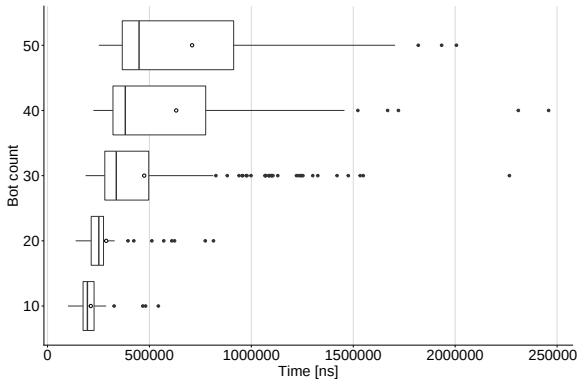


**Figure 5: Results of experiment *1LA* (50 bots, local, walking).**

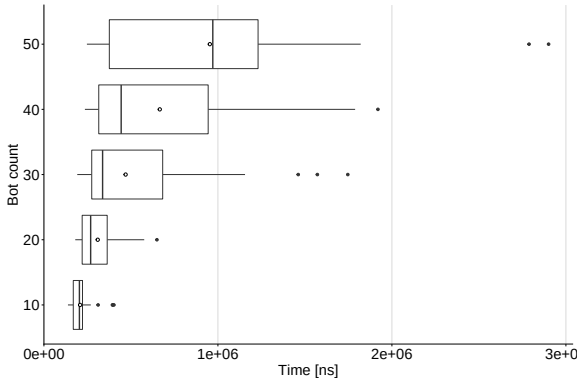


**Figure 6: Results of experiment *1LB* (50 bots, local, teleport).**

*6.1.2 Finding 1.* We noticed a latency increase when increasing the number of bots deployed on the Terraria server. This shows that every workload does have an impact on the server. We also noticed that the standard deviation increased when increasing the number of bots working on the workload. This means that the server performance becomes less stable, which will impact player experiences when playing on a server that has a lot of connected players.

*6.1.3 Finding 2.* Whilst running different workloads we expected to see a difference between the measured latencies. Looking at Figure 5 and Figure 6 we do see a difference between measured latency. We expected that for the teleport workload, the latency would be less, however, the opposite is true. We do find that the walking workload of the measured latency is smaller than the teleport workload.

*6.1.4 Distributed experiment.* For the distributed experiment the latency is a summation of two different latencies, the performance latency and network latency. The performance latency is the latency that is caused by the servers computing. The network latency is caused by the network communication overhead.

Figure 11 contains a large set of outliers, in Figure 12 we have filtered these outliers at 150 ms, to focus on the latency of the server. When looking at the latency we find that from 0 to 26 bots

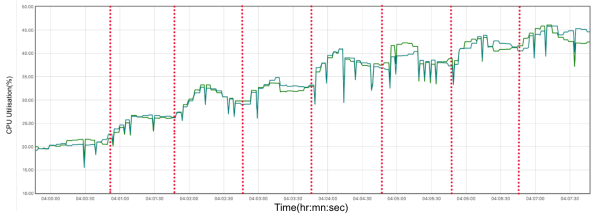


**Figure 7: CPU usage of experiment 1D (each dotted line indicates when new bots join)**

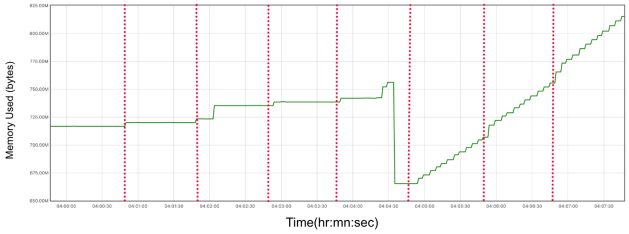


**Figure 8: Memory usage of experiment 1D**

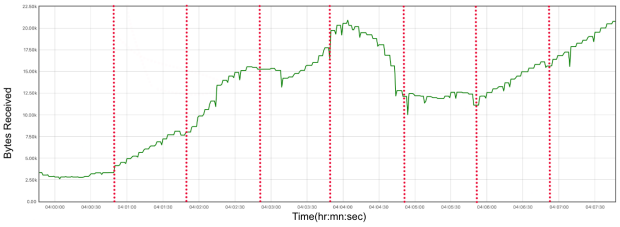


**Figure 9: Received network packets of experiment 1D**

the latency does not change consistently for our workload. This indicates that for our workload, up to 26 bots, no real latency is experienced and that the biggest cause for the measured latency is network latency. We found this by combining results from the local experiment and found that the local experiment latency is in a range of 10ms. The distributed experiment latency is in a range of 100ms. Therefore we know that the network latency is an order of magnitude larger than the processing latency for our workload. This means that the processing latency is insignificant until around 100 players because the network latency will take the overhead in the overall latency.

## 7 DISCUSSION

From Fig 7 the CPU usage of the server has a periodic upward trend which correlates to the number of bots joining the server and is equally distributed over both cores.
The memory usage of the server as shown in Fig 8 shows that the game state is stored in memory till 10 bots join and then seem to offload the game state to disk.
The packet throughput as seen in Fig 9 and Fig 10 shows a much higher number of updates up until 10 bots join the server. We believe

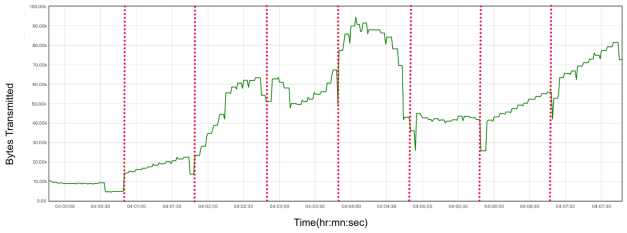


**Figure 10: Transmitted network packets of experiment 1D**

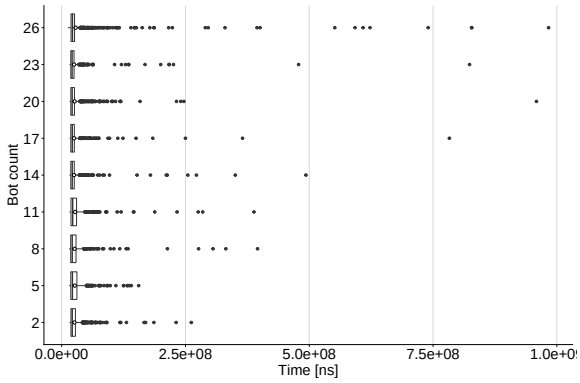


**Figure 11: Results of experiment 1D (24 bots, distributed, teleport)**

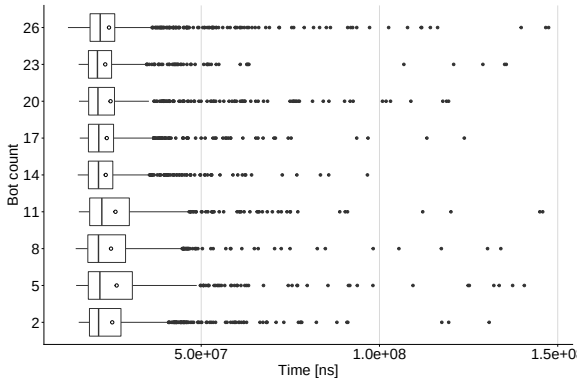


**Figure 12: Results of experiment 1D (24 bots, distributed, teleport) with outliers greater than 150 ms removed.**

this could be due to an unoptimized net code and the developers have been in contact with the community regarding this issue and are continuously optimizing it[14].

We believe that the gradual increase and then sudden drop in all metrics between 4:04:00 and 4:04:30 could be caused by the backup routine taking place after 10 bots joined the server but it is open to further investigation.

## 7.1 Complications

Writing a benchmarking tool for a closed-source game was never going to be straightforward. Due to the lack of documentation of Terraria's inner workings, a large amount of time was used up in troubleshooting Server and client versions. The lack of consistent online documentation made our only option to reverse engineer the network protocol from an older network library[12]. Added to these inconsistencies, the original bot implementation that we derived our bot from[15] had its own inconsistencies in working making debugging the bot and protocol a lot more time-consuming. In our distributed setup we ran into an issue where each node could hold a maximum of 24 bots (plus one observer). This could be due to a limit in the number of open TCP connections in the EC2 instance. We also could not connect more than two workload runners at a time and would like to further investigate this issue for future work.

## 7.2 Future work

The workload that we have used only uses a teleportation workload due to the complex nature of Terraria's network protocol. Although the walking workload ran in a local environment we faced challenges in running it in a distributed environment. We would like to run the experiments on a workload using other messages like interaction, item building, and modifying terrain. We would also like to see the experiments running with higher bot counts on different nodes.

# 8 CONCLUSION

To conclude, the challenges of performance and scalability in multiplayer games of the MVEs category like Minecraft and Terraria are eminent and ever-growing. The rapidly growing number of players of these games and the real-time interaction data these players generate means problems with server design, latency, and poor scalability will always be an issue. But this also implies there is always room for innovation and improvement. With the help of our design and experiment, we aimed to understand and measure the impact of these challenges, with a relatable and realistic workload and benchmarking. The results and the graphs obtained from our benchmarking tools and testing experiments indicate that our goal - To test the server performance and scalability of the game Terraria, seems to be achievable. We tend to observe the impact that a player's workload as basic as movement can have on the game and server performance. We also got to investigate what makes the server have performance issues and which basic and complex parts of the game have the most impact when it comes to the game's latency and scalability issues. In the future, more and more research and findings in this domain will lead us to more cutting-edge ideas to solve scalability problems, and organizing a well-suited distributed system customized only for a particular game/application to run on will be a thing of the past.

# REFERENCES

[1] "Gaming - statistics & facts | statista," https://www.statista.com/topics/1680/gaming/, (Accessed on 12/09/2022).
[2] "Steam charts - tracking what's played," https://steamcharts.com/, (Accessed on 12/09/2022).
[3] J. Gascon-Samson, J. Kienzle, and B. Kemme, "Dynfilter: Limiting bandwidth of online games using adaptive pub/sub message filtering," in *2015 International Workshop on Network and Systems Support for Games (NetGames)*, 2015.
[4] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, August 2008.

[5] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A distributed architecture for online multiplayer games." January 2006.
[6] J. Sirani, "Top 10 best-selling video games of all time - ign," https://www.ign.com/articles/2019/04/19/top-10-best-selling-video-games-of-all-time, (Accessed on 8 February 2022).
[7] R. Diaconu, J. Keller, and M. Valero, "Manycraft: Scaling minecraft to millions," in *2013 12th Annual Workshop on Network and Systems Support for Games (NetGames)*, 2013.
[8] "Top rated games on steam · steamdb," https://steamdb.info/stats/gameratings/, (Accessed on 12/09/2022).
[9] J. van der Sar, J. Donkervliet, and A. Iosup, "Yardstick: A benchmark for minecraft-like services," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 243–253. [Online]. Available: https://doi.org/10.1145/3297663.3310307
[10] "Most popular pc games - global," https://newzoo.com/insights/rankings/top-20-pc-games, Sep 2022, (Accessed on 12/15/2022).
[11] R. Weicker, "Benchmarking," p. 179–207, 2002.
[12] "Terrariabot: A library to make terraria bots in c#," https://github.com/Xwilarg/TerrariaBot, Dec 2021, (Accessed on 1/11/2022).
[13]
[14] "Known/reported journey's end issues," https://forums.terraria.org/index.php?threads/known-reported-journeys-end-issues.88826, (Accessed on 12/09/2022).
[15] "Meina: A bot that can play at terraria," https://github.com/Xwilarg/Meina, Dec 2021, (Accessed on 1/11/2022).

| Offset | Type | Description |
|---|---|---|
| 0 | Int32 | Message Length |
| 1 | Byte | Message Type |
| 2 | * | Message Payload |

**Figure 13: Network packet structure of the Terraria network protocol**

# A APPENDIX

## A.1 Packet Structure

The Terraria packet structure is defined in fig.13 message Length is the length of Type + Payload in bytes. The smallest message is one without a payload, so the Message Length would be 1.

# B APPENDIX

## B.1 Time Log

The below table shows the number of hours each team member has committed to the project in total. The fine-grained version can be found in the time log on Google Drive [1].

| Name | Sven | Abhilash | Anmol | Leroy | Aditya |
|---|---|---|---|---|---|
| Hours | 92 | 91 | 91 | 89.5 | 67 |

**Table 3: Time log.**

[1] bit.ly/DistSysTerrariaTimeLog