TP 3: Tool for monitoring QoS on the Internet

Carlos Gomes [pg47083], Miguel Gomes [pg54153], and Rui Armada [pg50737]

Universidade do Minho, Braga 4710 - 057, Portugal
Rua da Universidade, Braga, Portugal
https://www.uminho.pt/PT

Abstract. This report outlines the development and implementation of a sophisticated tool designed to enhance monitoring and task management within a gaming context, utilizing Discord bots as a central element. The project began with a decision-making process that led us away from replicating a prior study on latency in gaming footage, instead opting to develop a tool that integrates with Discord for real-time interaction and task management.

Keywords: Quality of Service · Performance · Latency · Discord · DiscordAPI · Discord Bots

1 Introduction

For this third practical assignment, the aim is to create a performance and quality of service monitoring tool. Quality of Service is related with monitoring, focusing specifically on the performance levels of network services and ensures that network resources are allocated efficiently, prioritizing critical tasks and managing bandwidth to reduce latency and packet loss. This is crucial in environments where real-time applications and services, such as VoIP and streaming media, are in use.

In the realm of online communities and services, Discord bots are a vivid illustration of these concepts at work. These bots, automated tools within the Discord platform, rely on robust monitoring and QoS to interact with users and manage tasks efficiently. They can be programmed to respond to network conditions actively or passively, enhancing engagement and functionality within servers.

Throughout this report, we will provide an in-depth examination of the steps taken during our project, offering detailed explanations for each decision we made. We aim to clearly articulate the rationale behind our strategies and the challenges we faced, shedding light on the careful considerations involved in our decision-making process.

2 Development

2.1 Why Discord?

The initial step in our practical work involved selecting a theme that would guide our project. Our objective was straightforward: to develop a tool capable of both monitoring and task creation, with the freedom to choose any theme. Building on the momentum of our first practical work, which involved analyzing latency in gaming footage, we considered continuing this line of research. However, due to potential risks and time constraints, we opted against replicating the study. Instead, we sought a theme that remained connected to gaming, leading us to the innovative idea of utilizing Discord bots. By implementing these bots, we could effectively monitor activities and manage tasks, thereby aligning our project with both gaming-related themes and our original goals.

2.2 Defining the architecture

Main bot The central component of our system, the main bot, is tasked with several critical operations and user interaction. It is primarily responsible for creating the dedicated communication channel within the Discord platform. This channel serves as the primary medium for task management and real-time updates. Additionally, the main bot is designed to manage and store all configuration settings reliably.

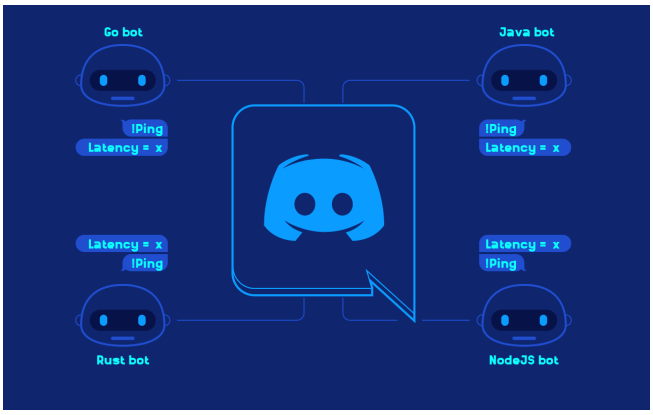

Fig. 1. Architecture Overview

Different bots In our project, we utilized a diverse range of programming languages, from C to Rust, to demonstrate the adaptability of our monitoring tool across different technology stacks. As outlined in Figure X, we designed the bots in each language to send a startup message to a log channel upon connection.

This approach not only tested the bots' functionality but also showcased the language-specific implementation nuances.

However, we faced two primary challenges: the variability of language documentation and dependencies. To address the first issue, we carefully reviewed and adapted to each language's documentation, adapting our code to fit the specific syntax and style guidelines.

The second challenge, managing dependencies, was initially more demanding due to the different installation processes required for each language. To streamline this, we developed a bash script capable of automating the installation of dependencies for each language used. This script significantly expedited our setup process, ensuring that we could maintain focus on the core functionalities of our bots, a small example can be found below.

Listing 1.1. Downloading Dependencies

```
# ! Go
echo "${YELLOW}Checking if Go is installed${YELLOW}"
if ! type "go" > /dev/null; then
    echo "${RED}Downloading Go${RED}"
    sudo apt-get install -y golang
fi

echo "${YELLOW}Downloading Go dependencies for the bot${YELLOW}"
go get github.com/bwmarrin/discordgo
go get gopkg.in/yaml.v2

echo "${GREEN}Everything is downloaded!${GREEN}"
```

2.3 Main Server

The Main Server is crucial for setting up and managing worker nodes on Digital Ocean. It uses a startup script that automatically configures each worker, making them ready to use right away. Our system includes a feature called Packet EMUlation (PEMU) that checks for changes in network traffic and packet headers and sends packets between nodes.

Listing 1.2. PEMU Definition

```
class Packet:
    @staticmethod
    def send(address, data):
        pass

    @staticmethod
    def send_sip_invite(address):
        pass

    @staticmethod
    def send_http_get(address, port):
        pass

    @staticmethod
    def send_rtp_packet(address):
        pass

    @staticmethod
    def send_dns_request(address):
        pass
```

The data collected, such as when packets arrive and leave, is stored in a dataset for analysis. We also use DNS sniffing to track network paths and create a map showing how data travels to each worker's specific Discord IP, like madrid719.discord.media. This mapping helps us see the network structure and improve data flow.

2.4 Workers

To effectively simulate and manage our project across multiple geographic locations, we used Digital Ocean, using a $200 credit available to us as students. This allowed us to deploy multiple nodes, each serving as a worker in different locations, and they are tasked with running different bots, each designed to perform specific network tasks and simulations.

Each worker bot is programmed to send a PEMU message to the main server, which is a critical function for assessing Traffic Shaping strategies under different network conditions. This allows us to analyze how data prioritization, rate

limiting, and other traffic management techniques affect network performance globally. Furthermore, to improve our network monitoring, each bot conducts routine network utilities such as:

Ping and ping-requests They're used to measure network latency, providing a basic sense of how fast data travels from one point to another in our network.

Traceroute Traceroute helps us to verify which path that data packets take through the network, which is essential for identifying potential bottlenecks and ensuring that data routes are optimized.

All the data collected from these diagnostic utilities are automatically transmitted back to the main server, where they undergo a comprehensive analysis.

2.5 Data Preview

In figure 2, we can see an example of a message sent by one of the workers to the channel where we can observe values such as latency, public IP or the language used.


```
{"latency": 147.626400, "public_ip": "92.250.97.136", "hostname": "zephyrus", "lang": "go", "timestamp": 1714601608}

{"latency": 353, "public_ip": "92.250.97.136", "hostname": "zephyrus", "lang": "nodejs", "timestamp": 1714601622953}

{"latency": 150.00, "public_ip": "92.250.97.136", "hostname": "zephyrus", "lang": "rust", "timestamp": 1714601629708 }

{"timestamp":1714601635000,"public_ip":"92.250.97.136","hostname":"zephyrus","latency":"4000.00","lang":"luvit"}

{"latency":"1000.00","public_ip":"92.250.97.136","hostname":"zephyrus","lang":"ruby","timestamp":1714601641000}
```


Fig. 2. Example of a message sent by worker

2.6 Dashboard

To ensure that the data from our project is not only accessible but also easily interpretable, we've developed a web-based dashboard that can be accessed through a browser, utilizing the Flask API for backend support. Flask serves as a foundation to host our web application, but it does not handle the visualization aspects directly. For this purpose, we integrated *Plotly*, a powerful tool that specializes in creating interactive dashboards offering advanced features, such as the ability to zoom into specific data points and select areas for detailed analysis. These functionalities make it significantly easier for users to interact with and derive insights from the data, providing a user-friendly interface for complex datasets.

Fig. 3. Initial Page

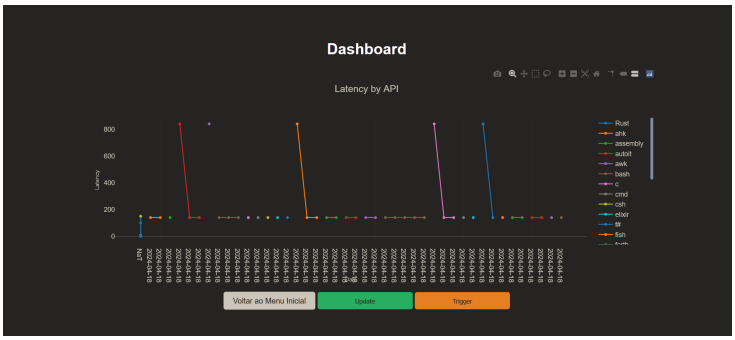

Fig. 4. Dashboard Preview (These aren't the real results, they are just a preview!)

As illustrated in Figure X, the dashboard not only allows users to visualize the data in a comprehensive manner but also includes three interactive buttons:

Voltar ao menu inicial By clicking this button, users can quickly exit back to the menu, simplifying navigation and enhancing the overall user experience;

Update If the JSON file containing the data is updated, users can simply click the "Refresh Data" button to ensure that the changes are accurately reflected on the dashboard. This button acts as a crucial tool for maintaining up-to-date information, allowing for real-time data synchronization without the need to manually reload or navigate away from the current page;

Trigger Finally, this button is designed to initiate the connection of bots programmed in different languages, allowing them to start transmitting data. By clicking this button, launch multiple bots are launched simultaneously, ensuring that the flow of information to the dashboard is both continuous and consistent. This feature is essential for integrating diverse data sources and providing a holistic view of the system's performance.

2.7 Results

To oversee and regulate the data flow throughout the network, we created Traffic Shaping datasets as part of our network management approach. By identifying important latency locations, these datasets were created expressly to evaluate and improve performance. We merged the datasets into our dashboard in order to obtain a thorough analysis.

We were also able to monitor and analyze the latency patterns visually with this connection, with a particular emphasis on the network's intrinsic latency problems and interactions with the Discord API. As a consequence, we were able to extract a wealth of information from which we could later draw conclusions. The results that follow show important trends and possible directions for development.

In figure 5, we can see that initially around 7.30pm the latency was around 125ms and there is a drop in latency between 8pm and 8.30pm and this can depend on many factors, such as the internet connection or location. And then in figure 6, we have more detailed information about the internet latency behind the bots, where there are several options to analyze, such as DNS or HTTP, each line basically represents a worker, as explained above, and the variation in the behavior of each node turns out to be interesting.

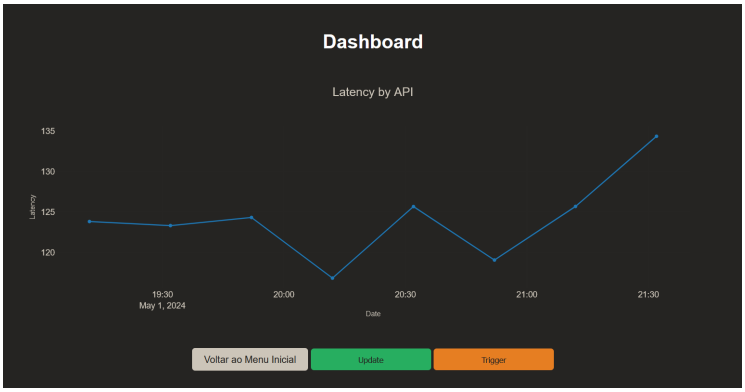

Fig. 5. Some results obtained when calculating the latency via the discord API

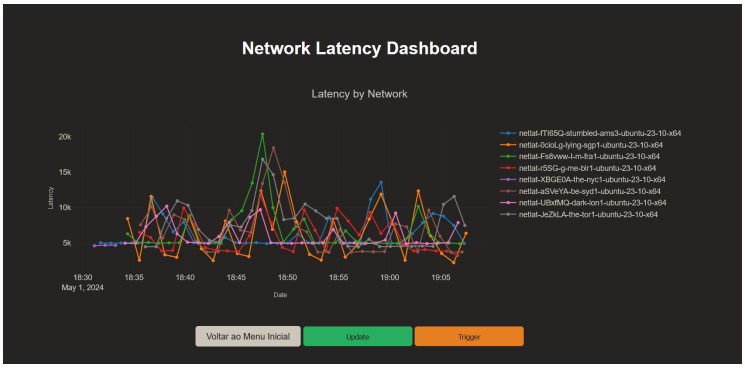

Fig. 6. Some results obtained when calculating the latency via the network

3 Conclusion and future work

It is evident that our project, despite having achieved all its intended goals, still holds potential for further development. In terms of the user interface, one immediate improvement could be to automate the dashboard updates, eliminating the need for manual update. Expanding the dashboard's capabilities to include data aggregation by specific time frames—such as daily, monthly, or yearly—would greatly enhance its analytical functionality.

On the backend, our exploration of various programming languages has set a strong foundation, yet the continuous emergence of new APIs provides opportunities for further integration and improvement. These enhancements will not only elevate the system's overall performance but also ensure its adaptability to future technological advancements.

We believe that we did a good job and that we met all the intended requirements.