Network Dashboard

Patrick Jiang
Dept. of Computer Science
Pacific Lutheran University
patrick.jiang@plu.edu

***Abstract*—This project introduces a web-based network dashboard that monitors connectivity performance for multiple websites. The system runs automated ping and traceroute tests through bash commands in a Dockerized environment to collect latency, jitter, and routing path data and save them into CSV files. A Go backend processes the CSV files, while the frontend visualizes the results with clear charts and hop displays. Docker ensures consistent deployment and execution across all environments. This tool helps users identify network delay patterns and routing issues simply and efficiently.**

Keywords—network monitoring, latency, jitter, traceroute, Docker, web application

I. INTRODUCTION

Network reliability directly affects user experience when accessing online services and applications. However, when delay or connectivity issues occur, users usually lack the necessary information to understand the cause of the problem. This project develops a web-based dashboard that provides visibility into network performance by presenting essential metrics in a clear and organized format.

The system uses bash scripts to run ping and traceroute commands on selected websites automatically. These tests collect latency, jitter, packet response, and routing path data, which are then written into CSV files for structured storage. A Go backend reads and processes the data from the CSV files and returns the results to the frontend in a usable form. The dashboard visualizes performance trends and hop relationships through charts and traceroute path displays, allowing users to observe changes in network behavior more easily and identify potential weak points in the connection.

Docker is used to ensure a consistent execution environment, so that network tests produce reliable and comparable results. Version control with GitHub supports iteration during development. This project aims to offer a simple, efficient, and accessible tool to help users monitor network performance and understand how different routing decisions and network conditions can affect Internet quality.

II. BACKGROUND

A. Bash (GNU, 2023)

Bash is used in this project to automate network measurement tasks in a consistent and reliable manner. It enables direct access to system-level commands, which is essential for collecting low-level networking data. The bash scripts repeatedly run ping and traceroute on selected websites, allowing the system to gather latency, jitter, and routing information over time. This automated execution reduces the need for manual testing and supports continuous performance monitoring. Bash is also used together with awk for processing raw command output. (*The Linux Information Projects, 2020*) While bash handles command execution and workflow control, awk is responsible for parsing specific fields from ping and traceroute results, filtering unnecessary text, and formatting data into structured values. With this approach, latency, jitter, and hop information can be accurately extracted and written into CSV files for backend usage.

The flexibility of bash makes it suitable for integration with other tools in a Docker environment, ensuring that the same network testing behavior is maintained across different deployment settings. By using bash and awk, this project demonstrates the importance of scripting skills in network diagnostics and data collection workflows, where stability, automation, and repeatability are crucial. This combination provides a reliable foundation for generating accurate data that supports deeper analysis in the later stages of the system.

B. Go (Gin Framework) (Gin web Framework, 2024)

Go is used in this project to build a lightweight backend service that provides data access for the web-based network dashboard. The backend reads the network performance results generated by bash scripts from CSV files, including latency, jitter, and routing path information. Using the Gin framework, the backend exposes a simple and efficient API that allows the frontend to request data for specific websites or categories. This design enables users to dynamically view updated measurements without modifying or interacting directly with the CSV files. The routing and request handling in Gin are straightforward, which makes the backend code easy to maintain and extend as needed. Since the data format and endpoints are well structured, the backend can support additional features in the future, such as filtering results by time or adding new categories of measurement. Go also provides strong type safety and clear syntax, reducing errors during development and improving code reliability. By using Go with the Gin framework, the project demonstrates the ability to build a clean and functional backend that focuses on reading structured data and delivering it to the frontend in a usable way. This approach ensures that the dashboard remains responsive and logically separated from the data collection process while keeping the overall system architecture simple and efficient.

C. React (Meta, 2024)

React is used in this project to develop the frontend interface of the network dashboard. It allows data to be displayed in a clear and interactive way, enabling users to observe the performance of different websites through visual elements such as charts and hop path displays. Component-based architecture helps organize the user interface into reusable elements, which improves maintainability and scalability as the project continues to grow. The system fetches processed results from the backend through API requests and updates the displayed information when the refresh button is clicked. The state management in React ensures that the dashboard responds smoothly to user input, such as selecting categories. This makes the user experience more dynamic and efficient compared to static interfaces. In addition, the flexible styling structure of React improves the visual clarity and accessibility of the dashboard, ensuring that users can interpret network metrics without difficulty.

After development, the React application is bundled using npm build. This generates a set of optimized static files that can be served efficiently by the backend. By using npm build, the final deployment becomes lightweight, and the performance of the web interface is enhanced. Through the integration of React and the backend API, the dashboard provides a clean and responsive platform for network monitoring.

D. Docker (Docker Inc., 2024)

Docker is used in this project to provide a consistent and isolated environment for running network tests and backend services. Since system tools such as ping and traceroute may behave differently depending on the operating system and installed dependencies, Docker ensures that the same configuration and tool versions are used across all machines. This prevents unexpected behavior during network measurement and improves the reliability of the collected data. The Docker environment includes both the bash scripts that collect network information and the Go backend that reads data from CSV files and serves it to the frontend. By packaging these components together, deployment becomes simplified, as the entire system can be started with a single build and run command. This approach significantly reduces the setup effort and lowers the chance of environment-related errors. Docker also supports reproducibility, which is important when validating results and maintaining system stability during future updates. The lightweight structure of containers ensures that resources are efficiently managed, and only the necessary components are included. By using Docker, the project demonstrates knowledge of modern software deployment practices and highlights the importance of environment control in network monitoring applications. This design enables the application to be easily migrated to different platforms without requiring changes to its underlying configuration.

E. Github (GitHub, 2024)

GitHub is used in this project as the primary version control platform. It allows the entire codebase, including the backend, frontend, Docker configuration, and bash scripts, to be stored and updated in a structured and trackable manner. By using Git, every modification can be recorded with a detailed history, which ensures that the project remains stable and errors can be traced or reverted when necessary. The repository is accessed through SSH authentication, which provides secure and convenient interaction with the remote server. SSH keys eliminate the need for password input during every push or pull operation and improve workflow efficiency. This approach is commonly used in professional development environments, especially when working with private repositories or deploying applications on remote machines. GitHub also serves as a backup solution in case of local machine failure, keeping the progress of the project safe in the cloud. The platform enables clear organization of development tasks, documentation storage, and version tagging, which helps maintain a predictable release structure. By using GitHub and SSH-based remote repository management, this project demonstrates practical skills in modern version control practices as well as secure and reliable software deployment workflows.

III. RELATED WORKS

A. Speedtest by Ookla (Ookla, 2024)

Speedtest.net is one of the most widely used web applications for evaluating Internet connection performance. It measures download and upload throughput, as well as basic latency between a user's computer and a nearby server. Tests are usually completed within seconds and provide a clear summary in the browser. Although useful for bandwidth evaluation, Speedtest focuses on a single connection session and does not expose packet-level behavior or routing paths. It does not track jitter or hop changes, and users cannot review historical performance unless they manually record results. Compared with this project, Speedtest only shows the outcome of the connection, but not where the delay occurs along the network route. It is designed for speed benchmarking rather than continuous monitoring or routing diagnostics.

B. Cloudflare Speed Test (Cloudflare, 2024)

Cloudflare speed test also measures latency, jitter, and throughput based on its global edge network. It provides more advanced metrics than Speedtest, including load effects under different connection conditions. The interface highlights both network stability and speed rating in a visually understandable format. However, Cloudflare still focuses on short-term testing, and the results do not persist after the session ends. The tool does not provide a hop-by-hop traceroute view, nor does it store data in a structured form for deeper inspection. It is more suitable for general users who want a quick evaluation rather than technical troubleshooting.

IV. Demo

The purpose of this experiment is to build a real-time web application called **Network Dashboard**, designed to monitor network performance across different categories of online services. The system continuously collects three key metrics, latency, jitter, and traceroute hop counts, by executing ping and traceroute commands on the backend. The results are written into CSV files and exposed to the frontend via API requests. The frontend processes and visualizes the collected data, enabling users to evaluate how well their current network condition supports different websites.

Once entering the application, users are greeted with a clean and well-designed dashboard layout with rounded cards placed on a light background for enhanced visual clarity. At the top of the page, three metric tabs, Latency, Jitter, and Traceroute, reflect the primary measurements monitored by the system. (*See Figure 1.*) Below them, a category selection dropdown allows users to filter websites based on service type, such as AI, Search Engine, AI, Video Streaming, etc.

The user workflow by choosing a service category. As shown in Figure 2, if "AI" is selected, the frontend will send a request to the backend to retrieve data:

http://localhost:8080/api/query?category=ai

The server reads recent data from CSV storage, filters the entries that belong to the selected category, aggregates them by hostname, and returns a unified JSON response. The frontend then dynamically renders multiple data cards, one for each monitored website under the chosen category. (See Figure 3.)

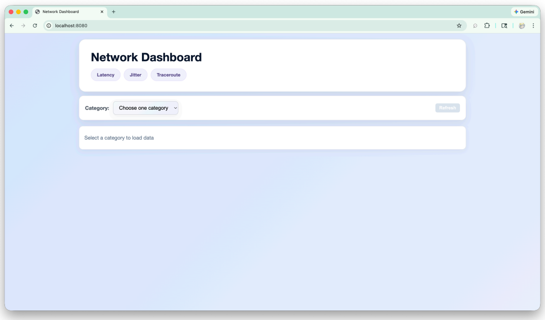

Figure 1. The initial interface state prompts users to select a category before loading data.

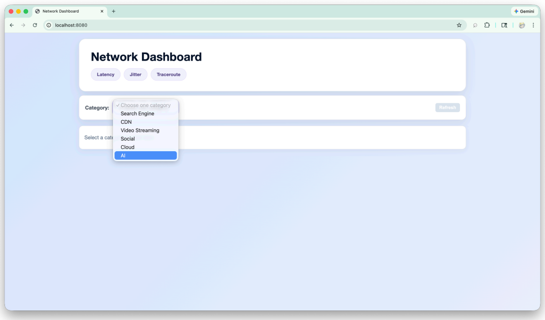

Figure 2. A category selection dropdown allowing users to filter monitored websites by service type

Each card contains (*See Figure 3.*):

- **A latency chart** showing how response time fluctuates over recent sampling points.
- **A jitter line chart** reveals variations in packet timing stability over time.
- **A traceroute summary** with a limit of a maximum of 10 hops and the hop-by-hop path.

As shown in Figure 3, the card for claude.ai shows that latency spiked above 35 ms around 10:43, then dropped to near 5 ms later, indicating unstable connectivity. The traceroute section shows the container's internal network path, with a maximum hop count of 10 and intermediate nodes such as PLU's network firewall gateway before leaving campus.

To ensure smooth interaction and prevent UI crashes, the frontend implements explicit loading, success, and failure states. If no category is selected, the dashboard displays a clear instructional message: *"Select a category to load data."* This avoids rendering empty charts and allows the user to understand how to interact with the system. The Refresh button triggers a new API request, allowing viewers to obtain the latest network measurements. This manual update option is extremely useful during network troubleshooting or when observing unexpected, rapid network behavior changes.

Overall, this demo successfully presents an end-to-end implementation of a lightweight but functional network monitoring tool. It validates the capability to acquire network performance data in real time, process it, and visually Thus, the Network Dashboard serves as a strong foundation for a scalable and practical network analysis application.

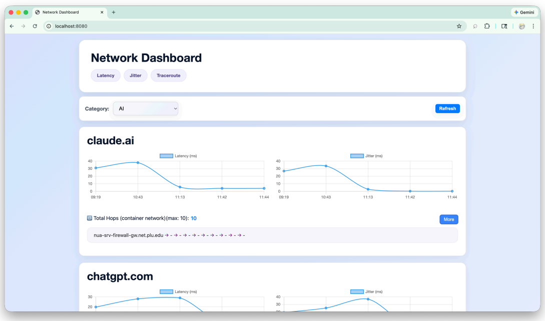

Figure 3. Network Dashboard visualizing latency, jitter, and traceroute results for AI services.

V. REFERENCES

Cloudflare. (2024). *Cloudflare Speed Test*.

https://speed.cloudflare.com/

Docker Inc. (2024). *Docker overview*.

https://docs.docker.com/get-started/overview/

Gin Web Framework. (2024). *Gin: HTTP web framework*.

https://gin-gonic.com/

GitHub. (2024). *About Git*.

https://docs.github.com/en/get-started/using-git

GNU. (2023). *Bash manual*. GNU Operating System.

https://www.gnu.org/software/bash/manual/

Meta. (2024). *React documentation*.

https://react.dev/

Ookla. (2024). *Speedtest by Ookla*.

https://www.speedtest.net/

The Linux Information Project. (2022). *awk*.

https://www.linfo.org/awk.html