Network Dashboard

Abstract

The Network Dashboard is a tool that checks and records how well a network is working. It measures key network information such as latency, packet loss, and route tracing. The backend runs simple tests like *ping* and *traceroute* using bash, saves to a CSV file, and then shows them on a dashboard. The dashboard lets users see real-time and past network data in charts. This project uses Bash and Go for the backend, basic web tools for visualization, and Docker to containerize the system for simple and consistent deployment. It shows how to use computer networking ideas such as connections, packets, and performance monitoring in a real program.

Introduction

Sometimes when using websites like ChatGPT or YouTube, the network feels slow and unstable. This usually happens because of high latency or packet loss. Tools like *ping* can test these issues, but they don't keep a record over time. The Network Dashboard solves this by running regular tests to measure latency and packet loss to servers such as Google and ChatGPT. The backend, written in bash and Go, saves the results into a CSV file. The dashboard then shows the data in simple charts, making it easy to see how the network changes over time.

Background

This project uses a few main tools to collect and display network data, and containerizes the project.

Go Programming Language

Go is used for backend because it is fast and has good support for network programming. It will just parse the CSV files and handle the API call from the frontend.

Bash

Two bash commands are used to perform the network tests. *ping* is used to measure how long it takes for packets to travel to a remote server and come back. It also shows packet loss if some packets are not returned. These two values, latency and packet loss, help describe how stable the network connection is. What's more, *traceroute* is used to find the path that packets take through routers to reach a destination. It helps detect where network delays or drops might happen along the route.

React and Chart.js

The dashboard frontend is built with React. It calls the backend API to get data and uses Chart.js to draw line charts for latency and packet loss. React makes it easy to update the dashboard in real time and build a simple interface for users to view network health.

Docker

Docker is used to containerize the application, including the Bash scripts and Go backend. It allows the entire system to be deployed with a single command while ensuring a consistent runtime environment across different machines.

Related Works

Speedtest by Ookla[1]

Speedtest.net is one of the most popular free websites for testing network speed. It measures download speed, upload speed, and latency between a user's computer and a nearby server. Users can start a test with one click and view results. It focuses more on throughput (bandwidth) rather than routing or packet loss. Compared with my project, Speedtest provides a full-speed test for users but does not store long-term data or show routing information.

Cloudflare Speedtest[2]

Cloudflare's speed test website measures latency, jitter, and download/upload speed using Cloudflare's global edge servers. It is browser-based and free to use. The site also provides more detailed data, like latency under load and connection stability. It is similar to my project in that it tracks latency, but my dashboard records and stores latency and packet loss data in a database for long-term monitoring rather than just a single session test.

Project Design

Overview

The Network Dashboard collects network data from several public servers and displays the results in a simple web interface. The backend is written in bash and Go, and it runs *ping* and *traceroute* commands regularly. It reads the results, calculates average latency, packet loss, and hop count, and then saves them into a CSV file.

[1] *Speedtest by Ookla* Ookla. (2025). *Speedtest by Ookla*. Retrieved from https://www.speedtest.net

[2] *Cloudflare Speed Test* Cloudflare. (2025). *Cloudflare Speed Test*. Retrieved from https://speed.cloudflare.com

The React frontend connects to the backend through an API and uses Chart.js to draw charts for latency and packet loss over time. Docker is used to containerized the project to make it easy to deploy.

Backend (Bash + Go)

- Runs *ping -c 10* to measure latency and packet loss
- Runs *traceroute* to count hops and track route changes
- Parses command outputs and saves data into a CSV file
- Provides a simple REST API for the frontend to get recent data

Frontend (React + Chart.js)

- Requests data from the backend API
- Displays latency and packet loss trends as line charts
- Show the number of hops or route changes in a clean table view

Project Plan

Week	Task	Due Date
Week 8 & 9	Define API, implement the frontend using mock data, and create the container	10/25
Week 10	Implement backend and connect frontend with backend use hard-coded data	11/08
Week 11	Implement bash commands to retrieve network data	11/15
Week 12	Testing	11/22
Week 13	Final check for the project and submit	11/29
Week 14	Prepare for presentation	12/06

References

Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks* (5th ed.). Pearson.

Linux Manual Pages. (2025). *ping(8), traceroute(8), netstat(8)*. Retrieved from https://man7.org

MySQL Documentation. (2025). *MySQL 8.0 Reference Manual*. Retrieved from https://dev.mysql.com/doc/

Chart.js Documentation. (2025). *Data Visualization Library*. Retrieved from https://www.chartjs.org

Ookla. (2025). *Speedtest by Ookla*. Retrieved from https://www.speedtest.net

Cloudflare. (2025). *Cloudflare Speed Test*. Retrieved from https://speed.cloudflare.com