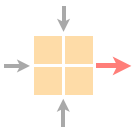
ETH zürich



Networked Systems
ETH Zürich — seit 2015

Collecting Storing and Serving of Network Power Data

Semester Thesis

Author: Andreas Hunziker

Tutor: Dr. Romain Jacob

Supervisor: Prof. Dr. Laurent Vanbever

June 2024 to August 2024

Abstract

This project aims to reduce energy consumption in network operations by creating a database to store and manage data on the power draw of network devices. The database centralizes this information, making it easier to analyze power consumption, identify inefficiencies, and support energy-saving efforts. To make the data easily accessible, a web app was built to provide API access, enabling users to gain data-driven insights that can drive improvements in energy efficiency. This project supports the goal of more sustainable network operations through better understanding and management of power consumption data.

Contents

Chapter 1

Introduction

1.1 Motivation

In an increasingly connected world, the energy consumption of network devices has become a significant concern for both environmental and economic reasons. Reducing energy consumption is not just a matter of turning off unused devices; it requires a deep understanding of the various factors that influence energy use across a network.

Gaining this understanding demands thorough analysis, which in turn depends on access to reliable, comprehensive data. To facilitate this, it is essential to have a structured repository where such data can be stored, managed, and made accessible for analysis.

This project aims to develop a database specifically designed to store and manage data related to the power draw of network devices. By providing a central hub for this information, we enable researchers, system administrators, and other stakeholders to analyze patterns, identify inefficiencies, and ultimately contribute to more energy-efficient network operations.

In summary, this project is driven by the need to better understand and manage the energy consumption of network devices. By creating a dedicated database, we provide the foundation necessary for the data-driven insights required to make meaningful reductions in energy use.

To achieve these goals, it's crucial to first identify the specific needs of the various stakeholders who will interact with this database.

1.2 User Requirements

This section outlines the various user groups and details their specific requirements. Three different user groups are identified, each with different requirements:

1. **Public Users**: Individuals such as scientists, system administrators, or anyone interested in accessing the data for research or analysis.

2. **Technical Users**: Entities such as Python scripts, CI/CD pipelines, or other technical components that interact with the database programmatically.

3. **Admin Users**: Personnel responsible for maintaining and developing the database and its surrounding codebase.

Public User Requirements:

- **Convenient access to data**: Users need to easily retrieve data for analysis and research purposes.
- **Documentation of access**: Transparency and traceability of data usage must be ensured.
- **Availability to public**: Data should be openly accessible to foster research and collaboration.
- **Database structure**: A well-defined database structure is necessary for users to understand how data is organized.
- **Documentation of structure**: Clear documentation should be provided to help users navigate and utilize the database effectively.

Technical User Requirements:

- **Option to contribute**: Technical users should be able to add data or make changes programmatically.
- **Error handling**: Robust error handling mechanisms are required to ensure smooth operation and troubleshooting.

Admin User Requirements:

- **Limit users able to change data**: Only authorized users should be able to modify the database to maintain data integrity.
- **Technical documentation**: Detailed documentation is needed to assist in the maintenance and development of the database.
- **Functionality is tested**: Automated tests must be in place to ensure the reliability and correctness of the database functions.
- **Adequate complexity**: The database should be as simple as possible while still meeting all requirements.
- **Database management tool**: Tools for effective database management and development are necessary for ongoing maintenance.

1.3 Technical Requirements

To fulfill the user requirements outlined in section 1.2, a relational database combined with an access API was chosen. Relational databases are well-suited for managing structured data, and APIs provide a user-friendly interface for both public and technical users. This section presents the technical requirements, outlining the necessary solutions and tools needed to effectively meet these identified user needs. Table 1.1 details how these technical requirements are met using specific tools.

By building a relational database and a dedicated API we can fulfill all the identified user requirements. The design of the API and the database is documented in chapter 3.

Requirement	Achieved by	Technical Tools
Convenient access to data	GET requests to API	Flask [10]
Documentation of access	Public documentation of the API	Swagger/OpenAPI [12]
Availability to public	Web server, WSGI	Nginx [8], Gunicorn [6]
Database structure	Relational database	PostgreSQL [11]
Documentation of structure	Public ER diagram	dbdocs.io [4]
Option to contribute	POST requests to API	Flask [10]
error handling	Different return to failed request	HTTP status codes
Limit users able to change data	Reject unauthorized POST requests	Authentication token
Technical documentation	README files, comments	Markdown
Functionality is tested	Automated tests	pytest [7]
Adequate complexity	Simplest applicable solution	-
Database management tool	Object Relational Mapper (ORM)	SQLAlchemy [13]

Table 1.1: Fulfillment of Requirements

1.4 Motivation of Tools

The tools used in this project are widely recognized for their simplicity, reliability, and versatility. Each tool is well-documented and supported by a large community, making them easy to use and maintain, while providing robust solutions for building and managing scalable databases.

- **PostgreSQL**: Chosen for its robustness, scalability, and support for advanced SQL features, making it a reliable option for managing complex databases.
- **Flask**: Selected for its lightweight and flexible framework, allowing rapid development of APIs with minimal overhead.
- **SQLAlchemy**: Integrated with Flask to provide a powerful ORM that simplifies database interactions and enhances code maintainability.
- **Nginx and Gunicorn**: This combination provides a secure and efficient way to serve Flask applications, with Nginx handling static files and acting as a reverse proxy, while Gunicorn manages multiple Python processes for handling requests.
- **Swagger**: Used for creating interactive API documentation, improving usability and accessibility for developers by providing a clear, visual interface to understand and test the API.
- **dbdocs.io**: Chosen to make the ER diagram publicly accessible, fostering transparency and collaboration by allowing stakeholders to easily review and understand the database structure.
- **pytest**: Utilized for writing tests due to its simplicity and powerful features, ensuring robust and reliable application performance through extensive testing capabilities.

By leveraging these tools, the project aims to create a robust, scalable, and user-friendly database solution that meets the diverse needs of its users.

Chapter 2

Background and Related Work

2.1 Background

This section provides a brief overview of key concepts relevant to this thesis:

- **Relational Database**: A type of database that organizes data into tables (relations) of rows and columns, allowing for efficient data storage and retrieval. Relational databases use SQL for querying and managing data.
- **Primary Key**: A unique identifier for each record in a table, ensuring that each entry is distinct and can be referenced reliably.
- **Foreign Key Constraint**: A field in one table that uniquely identifies a row of another table, used to establish and enforce a link between the data in two tables.
- **One-to-Many Relationship**: A relationship in which a single record in one table can be associated with multiple records in another table, commonly used to represent hierarchical data structures.
- **ER Diagram (Entity-Relationship Diagram)**: A visual representation of the entities within a system and the relationships between them, used to model database structures conceptually.
- **API (Application Programming Interface)**: A set of protocols and tools for building software and applications, allowing different programs to communicate and share data effectively.

For more detailed information on relational databases, see Fourny (2023) [5].

2.2 Related Work

This section reviews existing projects and studies related to the development of energy consumption databases specifically for network devices.

One notable initiative is Boavizta, a French association dedicated to helping organizations assess, manage, and reduce the environmental impact of their digital infrastructures [2]. Boavizta's Energizta project is particularly relevant as it focuses on the collaborative collection and dissemination of open data regarding the energy consumption of servers [1]. Energizta provides a platform

for aggregating energy usage information, which helps in understanding the environmental impact of digital operations. While this project offers valuable insights into server energy consumption, it is still in its early stages, with limited data and features, and does not yet include information on network devices.

Another relevant project is the development of the "Autopower" system, which was created as part of a bachelor project. Autopower is designed to collect power data from power meters and store this information in a centralized database [3]. This database is partly accessible via the web app developed in our project, demonstrating a proof-of-concept for connecting the two databases.

While there is substantial research on energy consumption in data centers and IoT environments, there is a clear gap in publicly accessible energy consumption data specifically for network devices. This thesis addresses this gap by developing a dedicated platform for collecting and sharing this data, thereby supporting more sustainable network operations.

Chapter 3

Design

3.1 Overview of Components

This section details the various components of the project and illustrates how they are interconnected. See fig. 3.1 for an overview of all components. The following subsections will explain the components in detail and set them into context of the project.

3.1.1 Users

Users, in the context of fig. 3.1, are public users as defined in section 1.2. While public users can access the data, they are generally unable to contribute directly. However, the public Data Sheets GitHub repository allows indirect contributions. [9].

3.1.2 Data Sheets

This is an open-source project dedicated to collecting and maintaining detailed information about the power consumption of various network devices [9]. The information is primarily sourced from manufacturers' data sheets and is directly accessible in the repository. Users can also access the NetPowerDB via the public API. The data from the repository is in sync with the database and new entries are automatically added to the database by a CI/CD script.

3.1.3 NetPowerDB

The PostgreSQL database, known as NetPowerDB, is central to this project. It stores data uploaded via the private API and makes it accessible through the public API. As a relational database, it organizes information into multiple tables. Essentially, NetPowerDB houses the following types of data:

- **Measurements**: Data from both the power supply unit (PSU) and network interfaces.
- **Devices**: Information on routers and their interfaces, each uniquely identifiable.
- **Models**: Details on router and transceiver models, containing information from the data sheets of their manufacturer. This data is taken from the Data Sheets repository.

The structure of the database is discussed in more detail in section 3.2. The reasoning behind choosing PostgreSQL can be found in section 1.4.

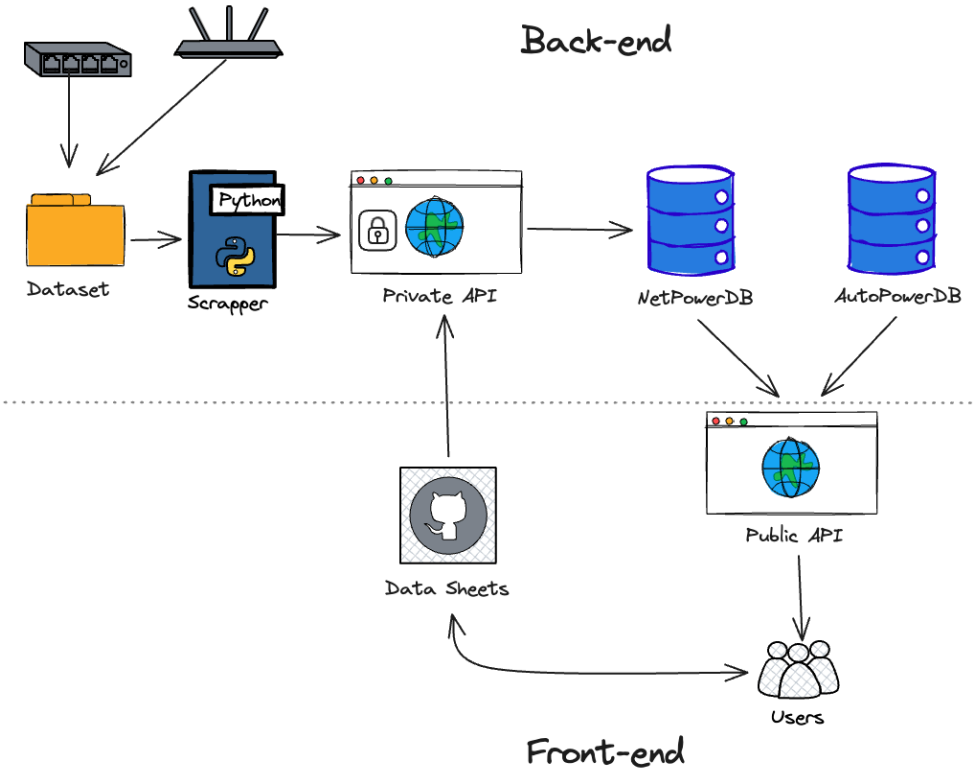


Figure 3.1: Overview of Components

3.1.4 AutoPowerDB

AutoPowerDB is a database that contains data similar to NetPowerDB, but collected through different methodologies. For more information on the AutoPower project, refer to section 2.2. As part of our project, we attempted to integrate the two databases. To facilitate this, a proof-of-concept API endpoint was implemented. Some of the data from AutoPowerDB is accessible through the public API but the data collection process remains independent of this project. AutoPowerDB includes specific logic for data collection, which led to the decision to create a new database instead of merging it with NetPowerDB, despite their similar content.

3.1.5 Public & Private API

The API is responsible for making the database accessible to the users. The API serves different user groups, which are detailed insection 1.2. Public API endpoints provide data from NetPowerDB and partially from AutoPowerDB to both public and technical users. Private API endpoints are utilized for adding new data and are not publicly accessible; they are primarily intended for technical users. To use the private API, users must include an authentication token in their request. The API is hosted on a Flask application, which interacts with the database through SQLAlchemy [10] [13]. The rationale for selecting these tools is explained in section 1.4.

3.1.6 Dataset

The dataset consists of measurements provided by Switch LAN, the Internet Service Provider (ISP) of ETH Zurich. These measurements include data from the power supply units (PSUs) and network

interfaces of various routers. Each router is identified by a unique ID, and its model type is recorded. Each measurement is accompanied by a timestamp, ensuring accurate tracking of when the data was collected.

3.1.7 Scrapper

The scrapper is a Python script responsible for transferring all data from the dataset to the database. Within the context of user groups, the script functions as a technical user and it utilizes private API endpoints to upload data to the database. The script depends on a specific dataset structure, which is predefined. Note that creating the dataset is beyond the scope of this project. Future improvements may include automating the processes of data collection and upload.

3.2 ER Diagram

This section provides an overview of the NetPowerDB structure. Refer to fig. 3.2 for a detailed view of all tables and their relationships.

The table names in the ER diagram are descriptive of their contents; therefore, the diagram's details will not be reiterated here. It is assumed that the reader has reviewed and comprehended the ER diagram. Still, there are a few points to clarify:

- The term routers is broadly defined to include switches, meaning the `routers` table may also contain switches, and `router_models` may also contain switch models.
- Primary keys have been given descriptive names to clarify their construction.
- The `power_models` and `benchmark_measurements` tables are placeholders for future database development. Currently, they do not contain any data and are not accessible via the API.
- The maximum number of PSUs per router is set at eight, simplifying the schema compared to creating a separate table, as done for interfaces. However, this introduces a hard limit on the number of PSUs
- The attributes `typical_power_descr` and `max_power_descr` provide additional descriptions of the typical and maximum power values, respectively, including conditions such as a specific temperature or a certain network traffic volume.

Figure 3.2: ER Diagram

Chapter 4

Conclusion and Outlook

4.1 Conclusion

This project successfully developed NetPowerDB, a centralized database designed to store and manage data on the power draw of network devices. Key components implemented include a PostgreSQL relational database for structured data storage and a Flask-based web application that hosts public and private APIs, providing flexible and user-friendly access to the data. The project is available online at networkpowerzoo.ethz.ch, allowing users to explore and interact with NetPowerDB.

The project also includes a Python-based scrapper for automating data uploads and a synchronization mechanism with the open-source Data Sheets repository, ensuring that the database remains up-to-date with the latest power consumption data. These tools provide a comprehensive framework for analyzing network energy consumption, supporting more efficient and sustainable network management practices.

4.2 Outlook

The implementation of NetPowerDB lays the groundwork for extensive data analysis opportunities. One immediate application is the potential to correlate network traffic volume with energy consumption, which could provide insights into the relationship between network usage and power draw. Additionally, the database could be used to calculate more realistic typical power draw values that go beyond the estimates provided in manufacturers' datasheets.

Currently, the `power_models` table serves as a placeholder, but there is significant potential for its future use. This table could be developed to calculate expected power consumption based of network devices, offering a more precise tool for predicting energy needs.

Automating the data collection process is another key area for future development. Automating the ingestion of power draw data would help maintain an up-to-date repository with minimal manual intervention, increasing the efficiency and reliability of the database.

Lastly, extending the API functionality to support more sophisticated queries would enable users to perform detailed and complex analyses directly through the API, further enhancing the usability and impact of NetPowerDB.

Bibliography

[1] Boavizta. Energizta. https://github.com/Boavizta/Energizta, 2024. Accessed: 27.8.2024.

[2] Boavizta. Evaluate the environmental impact of digital technologies across organizations. https://boavizta.org/en, 2024. Accessed: 15.8.2024.

[3] Chung, J. Automated power measurement for network devices: Collecting data reliably made simple? https://www.research-collection.ethz.ch/handle/20.500.11850/685212, 2024. Bachelor Thesis, ETH Zurich.

[4] dbdocs.io. https://dbdocs.io/. Accessed: 23.8.2024.

[5] Fourny, G. *The Big Data Textbook - teaching large-scale databases in universities*. Independently published, 06 2023, ch. 2.3.

[6] Guncicorn. https://gunicorn.org/. Accessed: 23.8.2024.

[7] holger krekel, and pytest-dev team. pytest documentation. https://docs.pytest.org/en/stable/. Accessed: 23.8.2024.

[8] Nginx. https://nginx.org/. Accessed: 23.8.2024.

[9] open source. PowerDB_Datasheets. https://github.com/nsg-ethz/PowerDB_Datasheets/tree/main. Accessed: 28.8.2024.

[10] Pallets. Flask documentation. https://flask.palletsprojects.com/en/3.0.x/. Accessed: 23.8.2024.

[11] PostgreSQL. https://www.postgresql.org/. Accessed: 23.8.2024.

[12] SmartBear Software. OpenAPI Specification. https://swagger.io/resources/open-api/. Accessed: 23.8.2024.

[13] SQLAlchemy. https://www.sqlalchemy.org/. Accessed: 23.8.2024.