ETH zürich

Networked Systems
ETH Zürich — seit 2015

Data Visualization of Switzerland's Research and Education Network

Group Thesis

Authors: Simon Englert, Leonard Wechsler

Tutor: Dr. Romain Jacob

Supervisor: Prof. Dr. Laurent Vanbever

September 2023 to December 2023

Acknowledgements

We would like to thank Prof. Dr. Laurent Vanbever, Dr. Romain Jacob and the Networked Systems Group for giving us the opportunity to work on this project. We would also like to express our sincere gratitude to our tutor, Dr. Romain Jacob, for his guidance during the project. Last but not least, we would like to thank our families for their support.

Abstract

This thesis aims to answer two challenging questions: First, what part of network traffic and power data is worth showing? Second, in which way can it be visualized best? As an use case we decided to combine two independent data sources. On the one hand we analyze traffic and energy consumption data of Switzerland's National Research and Education Network. On the other hand we use carbon dioxide emission and actionable electricity data. Based on that, we design and implement a GUI, while having UI/UX experience in mind and ensure reliable, modular and scalable code. We hope that our work can be of help to researchers and students in the field of computer networks, network operators in industry, or anyone else interested in this field of research in general.

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface	1, 3, 8, 14
CD	Continuous Deployment	14
CI	Continuous Integration	1, 4 f., 14, 19
DCC	Dash Core Components	17
GUI	Graphical User Interface	ii, 1–4, 7, 12, 18 f.
NREN	National Research and Education Network	3, 17
UI	User Interface	ii, 2, 19
UX	User Experience	ii, 2, 19
YAML	yet another markup language	5

Chapter 1

Introduction

1.1 Motivation

The Information Age brings many disruptions with it. One particularly groundbreaking one is the explosion of data: We witness an exponential growth of data. Huge amounts of information are being generated, stored and processed.

This can be overwhelming at times. How can we put this data to use? How can we extract meaningful insights? What part of a data set is worth showing and how can certain aspects of the data be visualized best?

We aim to address these questions in this thesis. On the one hand we investigate traffic data from Switzerland's Research and Education Network (Switch) [1]. On the other hand energy and carbon emission data from Electricity Maps' Application Programming Interface (API) [2] is analyzed. We want to unravel intriguing observations about this network's traffic behaviour. What fresh revelations will the combination of two independent data sets yield?

Ultimately, we want to provide intuitive ways of data exploration of a network. We hope our work might assist researchers and students of Computer Networks, network operators in industry or generally be interesting to anyone who is excited about this field of research.

1.2 Task and Goals

To achieve this, we identified the following objectives:

- Solving the difficult questions: Which part of the data is most useful to visualize for the intended users? What is the best way to visualize it?
- Design of a user-friendly GUI with three panels: an "Overview" panel summarizing traffic, energy consumption and carbon dioxide emissions, while interactive "Traffic" and "Energy & Emissions" panels allow users to explore data, view time series graphs and virtually relocate the network to compare carbon dioxide emission data.
- Concrete realization of the GUI in software, including the implementation of a backend for data parsing and evaluation, a modular frontend for rendering and interactivity and the development of unit tests with a Continuous Integration (CI) pipeline for the reliability of the application.

1.3 Overview

Chapter 2 introduces the data sources we use as well as the tech stack the application is built upon. In addition we relate our work to other papers on the topic. Chapter 3 presents our UI/UX design as well as the GUI's software architecture. In Chapter 4 we assess our system. Responsiveness along with computation-time of the data analysis are evaluated and discussed. Chapter 5 illuminates possibilities of future work. Especially the integration of big data promises exciting insights. Last but not least in Chapter 6 we give a short summary and final thoughts on the project.

Chapter 2

Background and Related Work

2.1 Background

In order to be able to analyze and discuss data, you first need sources to obtain it. Our data sources are listed in section 2.1.1. Furthermore, the various tools and technologies used to create the application are presented in section 2.1.2.

2.1.1 Data Sources

Switch

Switch is a Swiss foundation established in 1987. It operates Switzerland's National Research and Education Network (NREN) [3]. A NREN is a specialized internet service provider focusing on the research and education communities within their country. They often provide a high-speed backbone network, designed to meet these communities' needs. They offer i.e. dedicated channels for a given research project. Recently NRENs expanded their services beyond their main task: Switch for example leverages the research network as infrastructure for managing Switzerlands and Liechtensteins top level domain names. A nice piece of trivia is, that the used nameserver is located in the IFW building of ETH Zurich.

We anticipated Switch to provide large data-sets containing information about deployed routers, their approximate geographic locations, physical connections between them and the traffic on these links. Also we hoped for data on the energy consumption of these devices. We obtained data about the routers, their locations and the connections between them. Unfortunately we only received about two days worth of traffic data and no energy consumption data at all.

To fill in the gap until more data is gained, we were forced to generate some dummy-data to be used in the GUI. It serves as a placeholder and can easily be exchanged with real-data once available.

Electricity Maps

Electricity Maps was founded in 2016 by Olivier Corradi with the intent to make data usable for combating the challenges of climate change. They developed a free visualization platform showing real-time data that suffices scientific standards. Further they offer a commercial API with a user base of over 40.000 daily users, including high profile ones such as Google and Microsoft.

Their service focuses on providing accurate information on:

- Electricity consumption

- Carbon dioxide emissions
- Carbon footprint of electricity [2].

2.1.2 Tech Stack

A multitude of different tools and technologies were used to build the application.

Backend

We use python, a high-level, genral purpose programming language. Python is dynamically typed and has garbage-collection. It also supports multiple programming paradigms such as object oriented- and functional programming [4].

This enables us to leverage the pandas module for the analysis of data. Pandas is a library written for python. It is specifically designed for data manipulation and analysis and thus offers data structures and operations for e.g., manipulation of numerical tables and handling time series data [5]. This led to it being widely used in the field of data science.

Frontend

Dash is a framework used for designing interactive web applications. It is built upon Flask as well as React and was developed by Plotly a prominent data visualization company. Originally it only supported python but has been extended to R, Julia and F#.

Dash allows developers to create GUIs with diverse data visualizations and interactive features. This is achieved by:

- Providing a declarative syntax for definition and styling of the layout and individual elements.
- Supporting real-time updating.
- Facilitating callbacks that enable interactivity and thus foster data exploration [6].

Dash is deeply integrated with the Plotly library, which is used for the creation of interactive and visually appealing plots and charts like heat maps, scatter plots, line- , bar- and pie-charts [7].

They allow end users a great degree of interaction like zooming and panning or hovering over data points to show additional information. The produced graphics also are of high quality and are therefore production ready.

Last but not least, Cytoscape is used for displaying a network graph, i.e. the position of nodes and the links between them. Cytoscape is an open source software platform that was initially designed to visualize molecular interaction networks and biological pathways. Since an agnostic approach was chosen, it can be used for network analysis and visualization in general. Cytoscape's software architecture allows the usage of many differenty plugins that provide specialized functionality [8].

Testing and Continuous Integration

Ensuring a robust and reliable application can be achieved through testing.

We use python's unittest module. It was inspired by JUnit a well established testing framework of the Java language that played a crucial role in supporting test-driven development. We only utilize basic features, but unittest is a mighty tool that employs a lot of object oriented concepts such as test fixture, test cases, test suites and test runners [9].

CI is a method in software engineering that is closely related to testing. It involves continuous building and testing of iterative code changes. The goal is, to catch bugs early and enforce agreed upon standards through the development process.

The process, which consists of tasks, dependencies and so on, is defined using a configuration file, which follows the yet another markup language (YAML) format. The job itself then gets executed by runners that are already integrated in the GitLab server of ETH Zurich's Electrical Engineering and Information Technology Department.

Once set up, each push triggers a pipeline consisting of different stages (e.g.: general, frontend, backend) and jobs. The result of this pipeline can then be analyzed through GitLab's web interface [10].

2.2 Related Work

There has been done plentiful work on the subject of visualizing network related data. Therefore we can only present a small selection of papers.

Two interesting reads on the topic are ‘A Dual-View Approach to Interactive Network Visualization’ [11] by Galileo Mark Namata Jr., Lise Getoor, Brian Staats and Ben Shneiderman of the University of Maryland as well as the research paper ‘Network Visualization’ [12] by Andreas Lodde of the University of Munich.

The first paper addresses the challenges of visualizing complex network data by implementing a tool called DualNet. Using this tool, they perform a case study on an email communication network. Compared to us, this paper chooses a similar approach: Developing a generic tool and testing it on specific data. We went the other way round: Developing a tool that visualizes specific data which now can be used for other data-sets as well.

The ladder of the two papers discusses network visualization, specifically with network managers and system administrators in mind. The paper advocates for data-aggregation and appliance of filters when serving the purpose of clarity. Since monitoring is important, from the administrators perspective, usage of real-time data is discussed. We explore the possibility of real-time data inclusion into our software in Chapter 5.

Second, we want to highlight Teodora Rajkovic Bachelor Thesis on the topic “21st Century Data Visualization for Enhancing Student Learning” [13] which was also written in the Networked Systems Group of ETH Zurich’s Electrical Engineering and Information Technology Department. In her thesis, Teodora redesigns an ‘interactive web application supporting students during the Advanced Topics in Communication Networks course project‘ [13]. Likewise to our work, this application is built using the dash framework. Our tutor encouraged us to study her application prior to designing our own. This was helpful in seeing dash in work and it inspired several ideas.

Richard A. Becker, Stephen G. Eick, Allan R. Wilks at AT&T’s Bell Laboratories contributed greatly to this research field with their work “Visualizing Network Data”, which was published in 1995 [14]. They used the software SeeNet to visualize network data. Similar to our panel-approach, they created different views. Interestingly enough, they also concluded that it is worth to show some data on a static view and other data on a highly interactive view. They came to astonishingly resembling conclusions as we did: ‘Static displays can be swamped with large amounts of data; hence we introduce direct- manipulation techniques that permit the graphs to continue to reveal relationships in the context of much more data. In effect, the static displays are parameterized so that interesting views may easily be discovered interactively.’ [14]. This approach is carried out analogously in our application in the traffic panel, as explained in the Design Chapter 3.

Chapter 3

Design

The design chapter discusses two topics:

1. The design process of the GUI: What data is interesting and how can we visualize it?
2. The design process of the application: How do we ensure flexibility, scalability and reliability?

3.1 Dashboard Design

Designing a visualization tool is challenging. As mentioned in section 1.2, the first step is, to filter out relevant data and to extract meaningful information. This goes hand in hand with the decision on how the data should be visualized.

As the sources provide large amounts of different kinds of data, an intuitive, non confusing approach is needed. Theses challenges boil down to an application with several panels. Each panel displays a different interesting aspect of the data.

3.1.1 The Overview-Panel provides a high-level summary

As there is a lot of data, a good entry point for the user is an overview of the data-set (Figure 3.2). The overview panel shows distilled insights from the other panels and the data as a whole.

To summarize the network's traffic, the *Total Network Traffic* is displayed in a time-series graph. This requires the aggregation of all interface data Switch provides.

When analyzing the data-set, we noticed that a fraction of routers carries the majority of total traffic. To emphasize this interesting fact, a bar-graph shows the *Traffic per Node*. With one glance, one can distinguish between the important and insignificant routers.

One of the interesting insights that can be gained from the combination of the Electricity Maps and the Switch data-sets, is the carbon footprint of the network. We wanted to visualize the development of the emissions over the span of a day. As this varies from month to month, we decided upon a *Carbon Emission Heatmap*. The heatmap allows to compile all this information into one single picture.

3.1.2 The Traffic-Panel allows for fine-grained traffic analysis

One interesting aspect we wanted to visualize, was the *Network Traffic* of the Switch network (Figure 3.3). The data-set includes time-series for most interfaces. But it is not apparent, where the data is going to and how the relation between the interfaces look like.

For this reason, we decided to create a *Network Graph*. This graph is constructed from the topology data that comes with the data-set. It shows a node for each router and an edge for each interface between. Plotting one big graph is way too messy (Figure 3.1), so another layer of abstraction is necessary. We grouped together subsets of the nodes to form a *City Graph*. Clicking on a city reveals the subset of underlying routers in a *Router Graph*.

With this approach, we can display the *Network Traffic* over time: i.e. by clicking on a link or node. The graph is updated accordingly, when the corresponding element is selected.

The values can be displayed as absolute data rate or as utilization in relation to the bandwidth of the link.

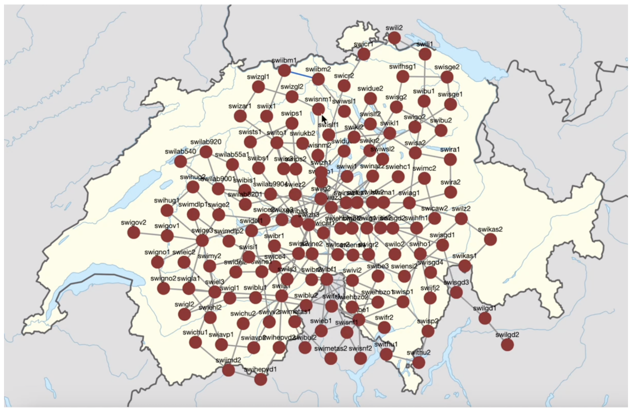

Figure 3.1: Displaying the complete router graph is messy. (Early version of the app)

3.1.3 The Energy & Emissions-Panel combines information on energy consumption and carbon emissions.

We visualize energy data in two different ways (Figure 3.4): A time-series graph shows the *Energy Consumption* at a specific point in time and an *Average Day Energy Consumption* line graph. This graph averages the energy consumption of one month. Mean, maximal, and minimal energy usage are then shown for every hour of the day. The month can be chosen with the *Month Selection Slider*.

It really gets interesting when combining the energy data with the carbon intensity, obtained using the Electricity Maps API. Multiplying yields the energy emission. The *Carbon Emission* is plotted in the same graph as the *Energy Consumption*. This allows for direct comparison of the two. In the same way as the *Average Day Energy Consumption*, carbon emission and carbon intensity are averaged over a selected month and displayed in the *Average Day Carbon Emission* graph. Again, the *Month Selection Slider* updates the graphs to display the data of the selected month.

The carbon intensity varies for each country. It is interesting to see the emissions, if the network would be based in another country than Switzerland. Using the *Country Selection Dropdown*, the user can select a country and the data is fetched using the API. All graphs are updated accordingly.

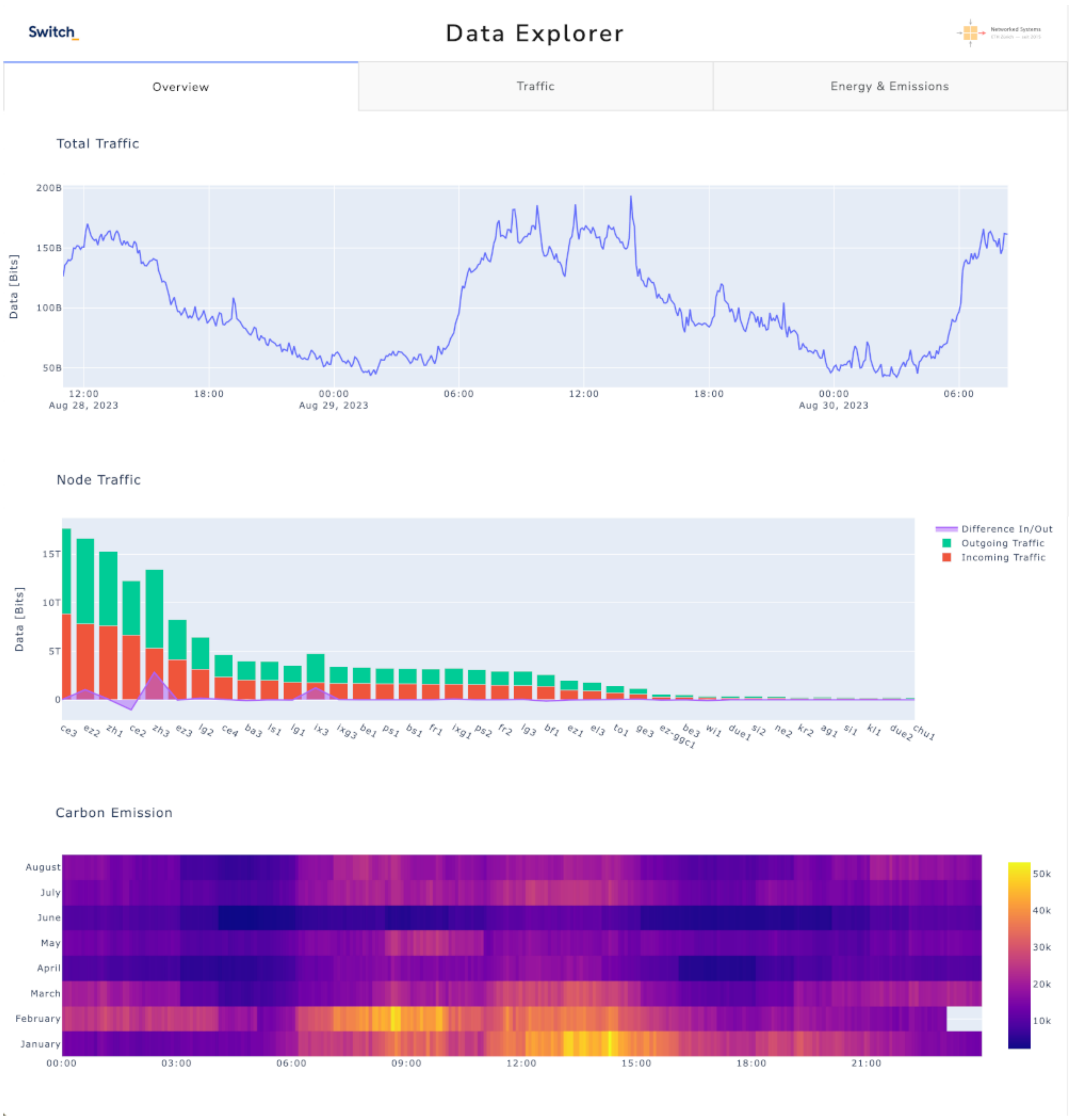

Figure 3.2: Overview Panel

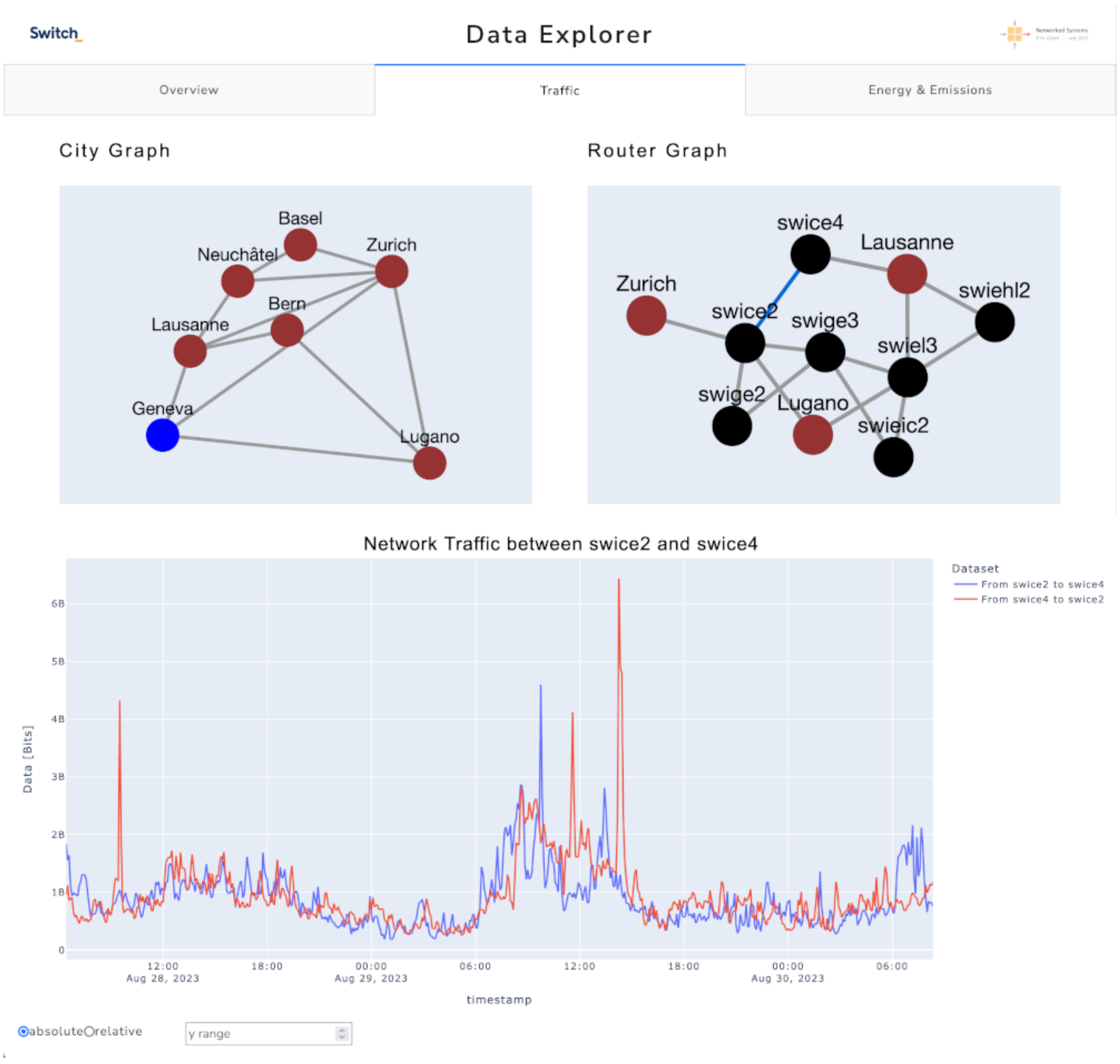

Figure 3.3: Traffic Panel

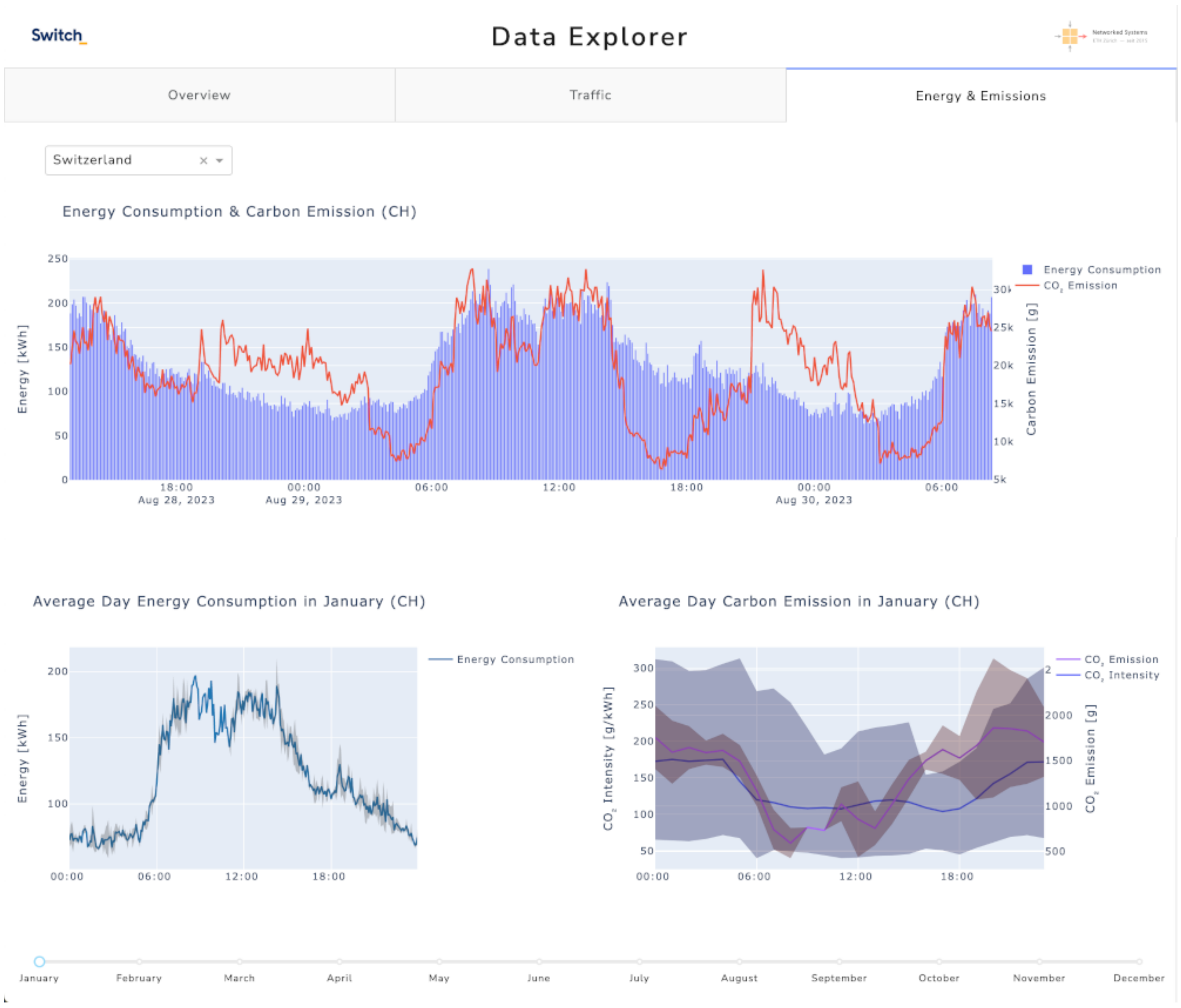

Figure 3.4: Energy & Emissions Panel

3.2 Software Application Design

The application is designed to host the GUI and process the given data. In order to achieve good scalability, we decided on a modular approach. We used functional programming, which can improve this kind of code-structuring [15]. Additionally, Dash is based on the React.js framework, which uses functional programming. So it is natural to leverage this programming paradigm for the purpose of this project.

3.2.1 Scalability Through Modularity

As seen in Figure 3.5, the application consists of a backend and a frontend section. With this approach, frontend and backend features can be developed independently. It also allows for easy modification and addition of new features. Data between front- and backend is passed and modified using functions. This allows to use lazy evaluation, which increases performance and responsiveness.

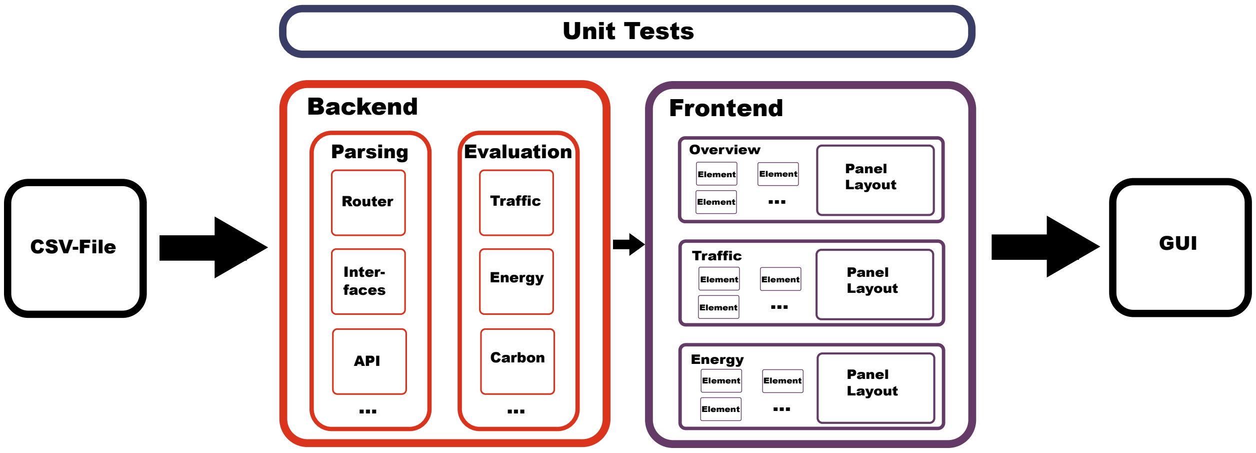

Figure 3.5: Software Application Design

The **backend's** responsibility is, to first fetch data from the respective sources. Secondly, it parses it into an usable format using pandas' dataframes. Next, relevant information is extracted and calculated. Modular parsing and evaluation functions exist to accomplish these tasks. The processed results are stored in a cache. This drastically increases performance, because large computations only need to be done once. Subsequent calls use the stored data instead of recalculating it. To simplify the access to the extracted data, the `backend_handler.py` interface is provided.

If a user now interacts with the GUI, callbacks are triggered in the **frontend**. These callbacks cause visual elements to update. If required, the frontend can request data using the `backend_handler.py` interface. For example, 3.6 implements a callback for the energy panel, seen in 3.4. The graph reacts to changes in the country selection dropdown and the month slider. It receives the dataseries from the backend, and then renders it in a scatter plot.

```python
@app.callback(
    Output(ids.ENERGY_CARBON_TIME_SERIES_GRAPH_03, "figure"),
    Input(ids.ENERGY_CARBON_DROPDOWN, "value"),
    Input(ids.ENERGY_CARBON_TIME_SERIES_GRAPH_SLIDER, "value")
)
def create_figure(zone, month):
    # Create figure
    fig = go.Figure()

    # Get data using backend_handler
    df_carbon_intensity = get_montly_carbon_intensity(zone,month)

    # Create a line chart
    fig.add_scatter(
        name = "CO2 Intensity",
        x = df['time'],
        y = df['carbon intensity direct']['mean']
    )
    return fig
```

Figure 3.6: Carbon Intensity Callback

Another special frontend feature is the separation of layout and components. Each component represents an element on the dashboard. The alignment of these individual components is defined in a separate layout. This approach is beneficial for modularity, as it decreases entanglement between different components and achieves encapsulation.

3.3 Quality Control and Automation

To ensure that the application runs without problems, it needs to be tested. We implemented unit tests, that can test any part of the code independently. Several aspects of the application are tested, for example:

- If the application runs without crashing
- If individual functions execute correctly
- If the electricity maps API has network access and is working
- If functions return the right data types

It is difficult to create realistic tests. But as this was not the focus of the project, some basic tests like the ones above are sufficient. If one publishes or sells the app, more sophisticated unit tests would be necessary.

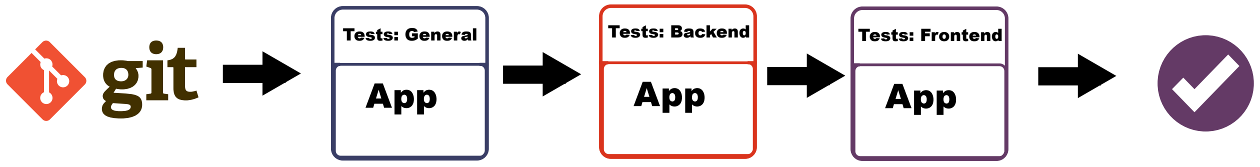

Figure 3.7: Continuous Integration Pipeline

Executing the unit tests manually one by one and revising the results by hand is tedious and error-prone. Here, Continuous Integration/Continuous Deployment (CD) pipelines, called CI/CD, come into play. They provide a workflow that allows for automation of the testing process. They especially excel in combination with version control tools, like Git [16].

For the project, a CI/CD pipeline was hosted on the ETH GitLab server. Whenever a new version is pushed, the pipeline automatically runs all general-, frontend-, and backend-tests (Figure 3.7. A result of the test results is compiled and can be accessed over the GitLab web interface. This makes it possible to keep an eye on older versions of the code and performance during testing.

Chapter 4

Evaluation

There are many ways to evaluate an application. One quantifiable measure is the responsiveness of the app, or in other words, how fast the application reacts to user input.

The test setup consists of a server, which runs inside a virtual machine on the Networked Systems Group's server and a Firefox web browser as client, which runs locally on a MacBook Air M1 2020. Both of them are on the same network. All test results are averaged over five measurements.

4.1 User Input Delay

We measure the delay of interacting with certain elements in the app using Firefox's network profiler [17]. This allows us to measure the delay of an action.

This measurement of course includes some overhead that can not be prevented, like the user's network speed. But we can compare the different actions to each other nevertheless.

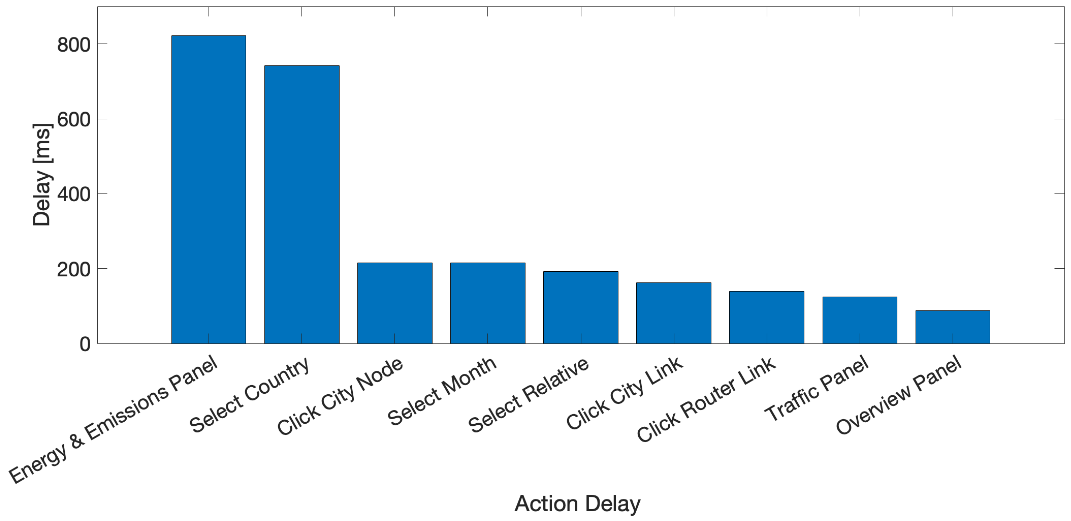

Figure 4.1: Delay for Different Actions

An interesting phenomenon to notice, are the long delays when measuring the rendering of the Energy & Emissions Panel and the selection of a country. These delays are caused by interacting with the Energy Maps API. To interact with the API, a request needs to be send to the Electricity

Maps server. For the result, we need to wait for the server's response. As the API interactions happen in real-time and are not cached, it takes some time for the results to arrive, which explains the sizeable delays. All the other actions happen locally at the server and are therefore significantly faster.

They have a base-delay that is caused by the Dash/Plotly framework. This adds up with the backend delay triggered by the user's action. Apart from two outliers, the delays only differ slightly, depending on the actual computational effort.

4.2 Caching Data

The implementation of a cache is discussed in Chapter 3.2.1. But is it actually worth to use a cache or is there too much operational overhead? To answer this question, the cache's performance is evaluated.

In these tests, we measure the time it takes to perform three different calculations. The tests are taken with and without caching.

Performed Operation	Without Cache [ms]	With Cache [ms]
Router Overview Computation	4201	3.5
Parsing Network Topology	120	9.0
Total Traffic Computation	14361	3.4

Table 4.1: Execution Time by Performed Operation

From Table 4.1 we can conclude that caching improves performance dramatically. The improvement ranges between a factor of 13 to 4700.

This stems from the fact that there is actually no computation whatsoever, when caching is used. Instead the pre-computed data is read from the cache. This means that the computation time does not decrease, but instead is performed at an earlier point in time. In practice, the computation can be done by the backend independently, so a client would not notice a delay at all.

Chapter 5

Outlook

Having only a limited amount of time when working on a project, there are often much more ideas than could be implemented. We would like to present the most interesting of these and explain the effects of their implementation.

5.1 Live Data

Administering Live Data into the application would have exciting technical challenges one would have to think about. The most prominent being the regular updating of graphs to show the real-time data. Fortunately "Live Updating Components" are already integrated in the Dash Core Components (DCC) Library: 'The dcc.Interval element allows [...] to update components on a predefined interval.' [18].

Seeing the data displayed updating according to real events would better user experience even further:

- Increased interactivity: Live updates allow users to dynamically explore and interact with the data.
- Monitoring: Real-time updates enable an immediate response to specific events.

5.2 International Data

From a technical perspective it would be highly favourable to incorporate data of additional networks into the application: Because of the application's modular and scalable architecture, such an integration could be carried out easily. After possibly adjusting the parsing functions to handle new input formats, the rest of the code could be used, as is, to visualize any network.

We think a user could highly benefit from such an implementation. One would be able to explore other NRENs such as the pan-european GÉANT or Internet2 of the United States of America. Investigating these networks alone is quite interesting but additionally, one would be able to examine similarities and differences between NRENs.

5.3 Big Data

As mentioned earlier, unfortunately we did not have access to large amounts of data since Switch could not provide enough data in the time we worked on the project. We think, however, that this would be the most exciting improvement to the application.

There would, again, be technical challenges. With larger data-set computation time in the backend becomes much longer. This is a worrying thing since responsiveness is critical to a GUI. Thankfully this can be easily mitigated using caching as explained in Chapter 3.2.1. Another interesting aspect is temporal resolution: the problem of being able to display large time frames of data as well as really fine grain visualisations. One approache to tackle this could be aggregation of time intervals.

Displaying large quantities of data would allow for new insights. Users would be able to recognize patterns like long-term trends. Possibly, they could even carry out a predictive analysis.

Chapter 6

Summary

We investigated which data is best suited for visualization to give users interesting insights into the Switch network. We also investigated different methods to visualize the given data. With this in-depth analysis, we not only wanted to improve the user experience. We wanted to provide valuable and understandable information that contributes to a deeper understanding of the intricacies of the Switch network.

Building upon that, we design a GUI, having UX/UI best practices in mind. We decided to divide the GUI into three panels. The “Overview” panel, is static and condenses the traffic, energy consumption and carbon emission behaviour of the Switch network. The “Traffic” and “Energy & Emissions” panels are highly interactive. They shall give a user the ability to explore the network data by themselves. The “Traffic” panel holds relational data between cities as well as individual routers inside the network. Being interactive, by selecting a city, router or a link, traffic data gets shown in a time series graph. As the name suggests, the “Energy & Emissions” panel holds data about the energy consumption and carbon emission of the network. We show this data for a typical day of a selected month. The user also can select a country and thus virtually move the network to this location. Now adjusted carbon emission data gets shown.

After this design process we had to actually implement the GUI. We decided to follow the functional programming paradigm. We implemented a backend, which parses the sometimes messy data and subsequently evaluates it. Next we realized a frontend, taking the data, rendering the GUI and enabling its interactivity. Throughout the project we took care for the code to be modular and scalable. We also implemented unit tests in combination with a CI pipeline to ensure the reliability of the application.

Last but not least, we evaluated our application in terms of its responsiveness and the computing time of the data analysis.

Bibliography

[1] Switch. The Switch Website, 2023. Last accessed 23 December 2023.

[2] Electricity Maps ApS. Electricity Maps, 2022. Last accessed 21 December 2023.

[3] Switch. The Switch Website: "Die Switch Stiftung", 2023. Last accessed 22 December 2023.

[4] Python Software Foundation. The official python website, 2023. Last accessed 22 December 2023.

[5] The Pandas Contributors Communits. The official pandas website.

[6] Plotly. Dash python user guide, 2023. Last accessed 22 December 2023.

[7] Plotly. The official plotly website, 2023. Last accessed 22 December 2023.

[8] Cytoscape Consortium. The official cytoscape website, 2018. Last accessed 22 December 2023.

[9] The Python Software Foundation. Unit testing framework, 2023. Last accessed 22 December 2023.

[10] GitLab B.V. Use gitlab (manual), 2023. Last accessed 22 December 2023.

[11] Galileo Mark Namata Jr., Lise Getoor, Brian Staats, Ben Shneiderman. A dual-view approach to interactive network visualization. *Conference Paper*, 2007.

[12] Andreas Lodde. Network visualisation. *MediaInformaticsAdvanced Seminar on Information Visualisation*, 2008/2009.

[13] Teodora Rajkovic. 21st century data visualization for enhancing student learning. *ETH Zurich Research Collection*, 2022.

[14] Allan R. Wilks Richard A. Becker, Stephen G. Eick. Visualizing network data. *IEEE Transactions on Visualizationand Computer Graphics*, 1(1):16–21, 1995.

[15] J. Hughes. Why Functional Programming Matters. *Computer Journal*, 32(2):98–107, 1989.

[16] Git Community. Git-scm, 2023. Last accessed 22 December 2023.

[17] The Mozilla Foundation. Firefox profiler, 2023. Last accessed 22 December 2023.

[18] Plotly Technologies Inc. Dash Python: Live Updates, 2023. Last accessed 21 December 2023.