Trabajo Profesional - Bachelor Thesis

Monitoring System for the Internet in Latin America

Student:
Malte Hansen

Tutors:
Prof Dr.-Ing. Meiko Jensen
Dr. Ing. J. Ignacio Alvarez-Hamelin

28th August 2019

Contents

List of Figures

Listings

1 Introduction

1.1 Motivation

The Internet is an always changing and expanding landscape, that, to this day, is still very hard to keep track of without a constant access to a huge amount of data. In Internet topology research exists an eminent interest to follow these developments and the differences between the state of the IPv4 and IPv6 architecture. The lack of available IPv4 addresses and the resulting increasing demand for the expansion of the IPv6 network is further driving this interest forward[11].
This is especially relevant when trying to research the Internet topology of a specific geographical region. While platforms such as RIPE Atlas[42] and CAIDA Ark[1] have been collecting various forms of data about the Internet topology worldwide for years, e.g. ping and traceroute outputs and information about autonomous systems (ASes), there currently does not exist a platform that offers the collection, processing and visualization of concrete and reoccurring data to researchers and groups interested in the development of one specific geographical region.
As the necessary transition between the IPv4 and IPv6 architecture[17, 36] is strongly influencing the current Internet landscape, the Complex Networks and Data Communication Group[3] is repeatedly conducting research about the Internet topology in Latin America. They already ran a number of different measurement campaigns to explore the network and closely observed the gap between the IPv4 and IPv6 architecture[28]. To aid in these exploration campaigns there exists a desire to develop a monitoring system for the Internet in Latin America.
The possibility and requirements to create such a system, as well as the design and analysis of an exemplary implementation will be discussed in this thesis.

1.2 Objectives

The objective of this project was to create a monitoring system that repeatedly collects data about both the IPv4 and IPv6 Internet topology in Latin America and visualizes the results, as well as making them accessible over a download function. This data is then to be analyzed to displays disparities between the two IP architectures by creating a network graph of the ASes and showing the state and inter-connectivity of the ASes, as well as the speed of the network traffic.
To achieve these goals the systems needs to be able to:

- Collect data about the ASes in Latin America
- Select sources and targets for traceroute measurements
- Schedule the measurements
- Process the measurement results to obtain the distinct metrics of interest

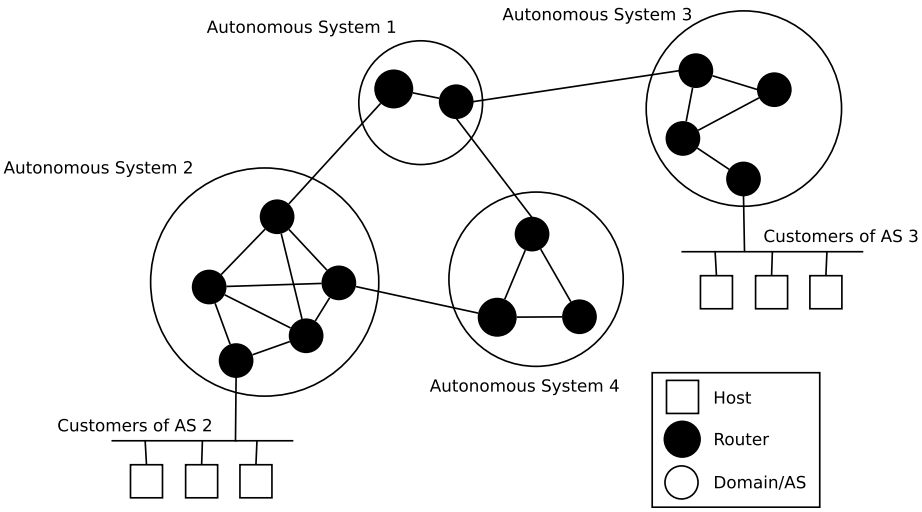

Figure 1: Exemplary Internet Topology

- Save the results and processed results to a database to make them accessible
- Present and visualize the current and historic results on a web page
- Provide a download function for selected results on the web page

1.3 Internet Fundamentals

1.3.1 Internet Network Topology

To select sources and destinations for the measurements and to be able to process and visualize the collected data in a meaningful way, we have to understand the architecture behind the Internet topology first.
A network topology in general describes the layout of a network. Within a network topology the physical topology can be seen as a map of the location of all physical network components and their cable layout. Due to the high cost of changing the physical topology it is seldom a target of changes. The logical topology, on the other hand, changes more frequently and describes the flow of data in a network. It displays which path any information would take to reach its destination.[23]
The Internet network topology consists of many different, interconnected subnetworks, called domains or autonomous systems, and their internal structure. Each AS belongs to a separate administrative authority and connects either hosts to the Internet or other ASes and itself with each other (see Figure 1). Inside one domain exist several routers that are forming the internal structure of the AS and serve as access points for the hosts to connect to the Internet, the intra-domain level. Here, each router is seen as a node. However, you can

also describe the Internet at the inter-domain level. Here, every AS represents a single node in the network topology. The use of network protocols differs between both levels to each address one node of the corresponding level.[20]
Due to the sheer size, lack of available information about each router and prevalent use of private IP addresses inside a single AS topology it is very difficult and demanding to extract meaningful information about the geographical layout and inter-connectivity of a specific geographical region. Moving forward, the focus was therefore set on the inter-domain level.

1.3.2 Internet Routing

You can project the two levels of the Internet topology, the intra- and inter-domain level, directly onto Internet routing.
Intra-domain routing protocols, also known as interior gateway protocols, determine the flow of a packet inside an AS. The most prominently used protocols for this purpose are the Routing Information Protocol (RIP)[25] the Open Shortest Path First (OSPF)[34]. RIP uses the route with the fewest routers between a source and destination, while OSPF implements the Dijkstra algorithm for the shortest path[13]. Inter-domain routing handles the packet flow between different ASes. The most frequently used protocol in inter-domain routing is the Border Gateway Protocol version 4 (BGP)[37]. BGP allows an AS to:

- Obtain information about subnet reachability from other ASes
- Forward this information to the internal routers
- Use this information and the AS policy to determine preferred routes to subnets

In conclusion BGP secures that every subnet is known in the Internet by forwarding its prefix and the path to reach said prefix over the whole network.[30]
However, these routing protocols do not assure that a package from one specific source to a specific destination always takes the same path over the network. Today's routers are commonly using load-balancing to reduce the load of a single path. Depending on the network strain, a package stream can then be split up and send out to the same destination over different output interfaces and therefore producing altering paths. This may happen by either per destination, per flow or per packet routing[16]:

- Per destination routing assigns each IP address of a subnet to a different output interface
- Per flow routing assigns all packets with the same flow identifier (consisting of IP source address, IP destination address, protocol, source port, and destination port) to the same output interface
- Per packet routing assigns the output interface for each packet independently

1.4 Internet Measurements

Before talking about creating Internet measurements, the question is, why do we need to measure the Internet?
When you use the Internet, you send a package from your device to a destination address using the TCP/IP-Stack. Your package is forwarded into the intra-domain network of your Internet Service Provider. It is then, over several other intra- and inter-domain networks, routed to the destination and the response is send back. There is no way for us to follow our packet and know where in the Internet topology we currently are. To gain knowledge about that, we need to gather information about the path of our traffic, by running measurements and combine the results to build our topology. For that, one needs to look at how to conduct network traffic measurements in general first.
Traffic needs to be created, which in turn, can then be dissected and analyzed by using sniffing tools or something similar. When researching the Internet topology specifically, the traceroute tool is often used[40]. Traceroute sends a packet to every router between a source and a destination by periodically increasing the time to live (TTL) of the packets sent by one. After each packet, the IP address of the destination router for the current TTL is displayed. If a packet is lost on the way, an asterisk is displayed instead. This only works, though, if every router on the way is forwarding each packet through the same interface each time. As discussed in the last chapter, this is not always the case, as load-balancing can alter the path of a packet stream. This means, that a traceroute can potentially show misleading results regarding the supposed network topology when two or more packets of the same traceroute are subject to a change in the routing, as seen in Figure 2. The risk of altering paths increases with the size of the network, as there are usually more possibilities to reach a single destination and each router is processing more traffic in general. This is, why a tool to reduce the occurrence of these anomalies is needed to conduct meaningful network traffic measurements on a larger scale.[31]
Paris traceroute[15] is an open-source software, expanding on the commonly known traceroute with the goal of drastically reducing the appearance of anomalies caused by load-balancing while running traceroute measurements. This is achieved by modifying the packet header fields to create the same flow modifier for all packets of a traceroute. Specifically, fields, not used for load balancing, in the first eight octets of the header are manipulated. For example, the Sequence Number field in TCP probes. This greatly increase the probability the same path is taken when facing per-flow load-balancing. Another strength is, that Paris traceroute offers users a way to identify whether per flow or per-destination load-balancing is being used. It also flags paths which may contain anomalies reliably. For this, the probe TTL field was implemented. It displays the TTL of the packet when the router decided to discard it. When facing normal traceroute behavior, this value is one as it will then be discarded due to an exceeded TTL, which is why a different value marks an anomaly. As resources are needed to conduct Internet measurements, a high quality of the measurement results is indispensable. Hence, the reduction in occurrences of anomalies

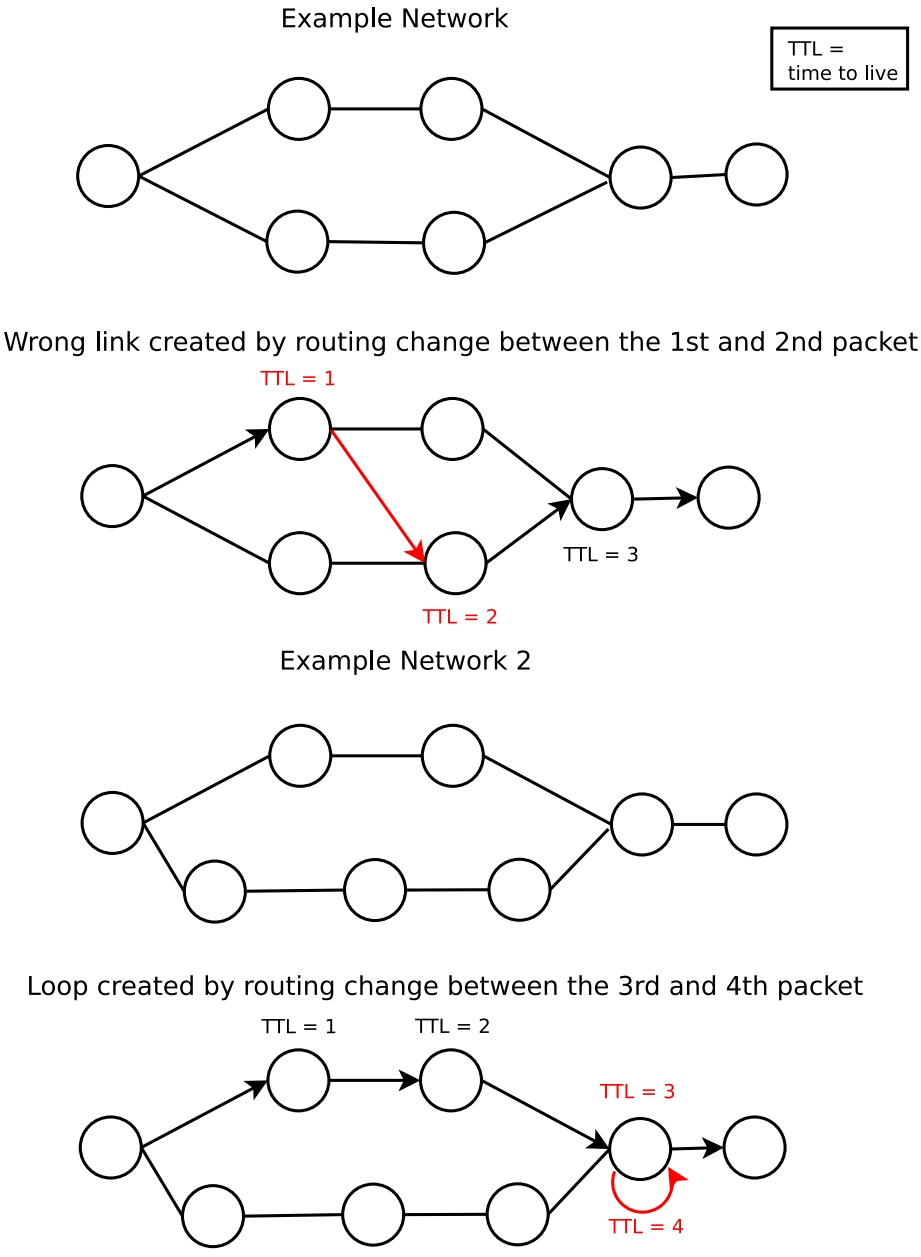

Figure 2: Exemplary anomalies in traceroutes caused by load-balancing

makes Paris traceroute the preferred tool to use.
Applying the above stated for Internet measurements means: To achieve an overview of the inter-domain Internet topology one needs to create network traffic between a multitude of sources and destinations and extract information about the path of the package, for example using Paris traceroute. The challenge when applying this to the Internet is creating the traffic. Since the Internet is fragmented into different ASes with different ownership, it is not possible to simply run a measurement from any origin to any destination. To tackle this problem, platforms launched probe projects, where multiple probes are connected to the Internet, usually by program partners, worldwide to create an own global network. These probes can then be used to act as the source of traceroute, ping or other measurements to explore the Internet infrastructure and obtain information about response times between certain network participants. Some of these platforms will be introduced in the next chapter.

2 Project Description

2.1 Data Collection

2.1.1 Platforms

Before we can collect any data we have to find platforms that can provide us the data needed. To implement the full scope of the project we need a platform to provide us:

- A list of ASes that are located in Latin America
- A means to identify an IP address of each AS to serve as a measurement target
- A list of network probes located in Latin America to serve as a measurement source
- A means to schedule and run Paris traceroute measurements

To collect the AS information the AS-Rank project from CAIDA was used. AS Rank provides detailed, estimated data about ASes using Internet topology data sets derived from traceroutes and BGP tables[12]. Importantly we can extract information about the geological location and country name of the AS and the size of their customer cone. Combining this information we can conclude a profound assumption that the AS is in fact located in Latin America and run by an Latin American organization. Because the country name of an AS data set can, at times, be inaccurate the customer cone size can be used to assert the status of the AS. A huge international organization will most likely not maintain a small network on a different continent to a small number of customers.
Next up was the assignment of an IP address to each AS. For this purpose data sets mapping IP prefixes to ASes, created from RouteViews Project data, were used[2][9]. This is done pairing the ID of the AS with an IPv4 or IPv6 prefix.

Lastly, the other two requirements could be fulfilled with the functionalities of the RIPE Atlas API[7]. The API allows to extract selected data about the RIPE Atlas probes around the world. The probe data contains, once again, information about the geographical location and country of the probe, that can be used for the source selection. Besides that, the API also allows the creation of measurements using the RIPE network. The measurement can be customized and offers a lot of relevant fields for us, such as:

- An IP address or URL as the target
- An array of RIPE probes as the source
- The IP protocol to be used
- The Type of measurement, e.g. traceroute or ping

For traceroute measurements there are a couple of other relevant fields:

- The number of packets to be used for a single traceroute
- The transport layer protocol to be used
- Configuration of Paris traceroute, if wanted

For further possibilities to include into the data payload see Listing 1. When using the RIPE API for measurements there are a couple of limitations to consider. Every measurement costs a specific number of credits, for example a single traceroute has a unit cost of 10 * N * (int(S/1500) + 1), with N equal to the number of packets and S as the packet size in octets [8]. Besides the credit costs there are also limitations for the number of simultaneous measurements, daily credit limit and more in place.

2.1.2 Target Selection

To pin down the target selection, we first have to define which ASes, derived from CAIDA AS-Rank, qualify as a valid measurement destination. Each AS should meet the following criteria:

- Support for both IPv4 and IPv6 traffic
- A geographical location (longitude and latitude) in Latin America
- An assigned country name of a Latin American country
- A customer cone size of less than 50

The customer cone describes the set of ASes, IPv4 prefixes and addresses that an AS can reach following only customer links.
The first criteria can be checked by searching for a prefix of both IP protocols in the IP-to-AS mapping data sets for each AS. This step is extremely important,

```
exemplary_request = {
        "definitions": [
                {
                        "target": "202.152.100.2",
                        "af": 4,
                        "description": "Example measurement",
                        "protocol": "TCP",
                        "packets": 8,
                        "size": 48,
                        "first_hop": 2,
                        "max_hops": 255,
                        "paris": 16,
                        "interval": 3600,
                        "type": "traceroute"
                }
        ],
        "probes": [
                {
                        "value": [1526, 2435,6536, 6783, 7536],
                        "type": "probes",
                        "requested": 5
                }
        ],
        "start_time": "2019-07-07T12:00:00Z",
        "stop_time": "2019-07-07T15:00:00Z",
        "bill_to": "ripe.user@test.com"
}
```

Listing 1: Exemplary data payload of a RIPE Atlas API measurement request

since measurement results from a traceroute to an AS which can only communicate with one IP protocol can not be used to compare the differences in the two architectures. To specify the list of Latin American countries, the LANIC country directory was used[6]. Lastly, the limit for the customer cone size was defined by previous experiences. Organizations with a size of 50 or smaller can reliably be assumed to be a regional organization.

2.1.3 Source Selection

Lets take a look at the criteria for the source selection to see which RIPE probes qualify as a potential source:

- Support for both IPv4 and IPv6 traffic
- Being connected and active in the network
- A geographical location (longitude and latitude) in Latin America

To see if a probe is currently operational and capable to use both IP protocols is fairly easy. Both the current status and IP interfaces can be extracted from the initial probe data received from the RIPE Atlas API. The API allows to filter for specific arguments when first collecting the probe data. Therefore the parameters for longitude and latitude can be used to restrict the discovery on Latin American probes. Afterwards the criteria can be asserted and unwanted data, such as probes from bordering countries, as the United States, can be eliminated by running the AS belonging to the probe against or AS data set.
As the chosen platform may be hosting multiple probes per AS, this process may leave behind more than one probe for one AS. However, as stated in Chapter 1.3.1, the focus was set on inter-domain networks. It can be assumed that one specific AS will always forward traffic to another specific AS over the same outgoing interface. Hence, we can eliminate all but one probe for each AS, as we can expect that the costs of the additional traceroutes will outweigh the information gained from them.

2.1.4 Frequency

In chapter 2.1.1 we briefly touched upon the costs and and limitations for RIPE Atlas measurements. When talking about the frequency of the data collection process these are very relevant.
Before discussing the scheduling of measurements the preconditions need to be looked at. Both the procedures for target and source selection can be run without any costs at any given time. This means that before any scheduled measurement the most recent data can simply be obtained via the given platforms.
The relevant limitations, per user, for our purpose are the following:

- No more than 100 simultaneous measurements
- No more than 100,000 results can be generated per day

- No more than 1,000,000 credits may be used each day

To give some context to these limitations, let us take a look at the estimated scope of one complete measurement of the entire Latin American inter-domain network. Considering current data (16. July 2019), we can calculate with roughly 60 source probes and 2000 destination ASes. This means that a complete campaign from every source to every destination would require around 120,000 traceroutes and 2000 measurements to be run for each IP protocol. Adding the cost of roughly ten credits per traceroute, per packet, a full campaign with a packet size of one has an estimated cost of 1,200,000 credits per protocol.
Given the context and limitations, since one traceroute also produces one result, a full campaign with minimal packet size would take three days to complete. This means, that to acquire a higher frequency than that a user would need to get into contact with RIPE to increase these limitations.
In conclusion, a normal RIPE Atlas user can, given that he owns the needed credits, frequent a full campaign roughly every three days. However, since needs and limitation may differ between users the possible frequency of scheduling measurements varies between users and should be able to be adjusted accordingly in the program.

2.1.5 Storage

The last step of the data collection is the storage of the results. To achieve that, a database needs to be available. Expanding on this, all the results and data accumulated in the data collection process until this point need to be transformed into a format that is compatible with the selected database. This will be discussed in detail in chapter 3.3, Data Management. However, the storage of the generated graphics needs to be thought through beforehand. While it is definitely possible to store the images locally on the server, on which the system will be hosted, it is desirable to also store them in the database, that will necessary to store the results of the data collection. The images can then be managed and accessed easier and the data requests of the web server can be run without any additional interfaces involved. The requirements of these servers and the database will be discussed in chapter 2.3.1, Components.
Considering the above stated and looking at the data we need to process through our database, we need to design a system that contains:

- Entries for each scheduled measurement campaign
- Graphics that belong to one campaign each
- Measurements that belong to one campaign each
- Traceroutes that belong to one Measurement each
- *optionally*, Probe and AS data used in each campaign

This leaves us with the simplified Entity Relationship Diagram in Figure 3.

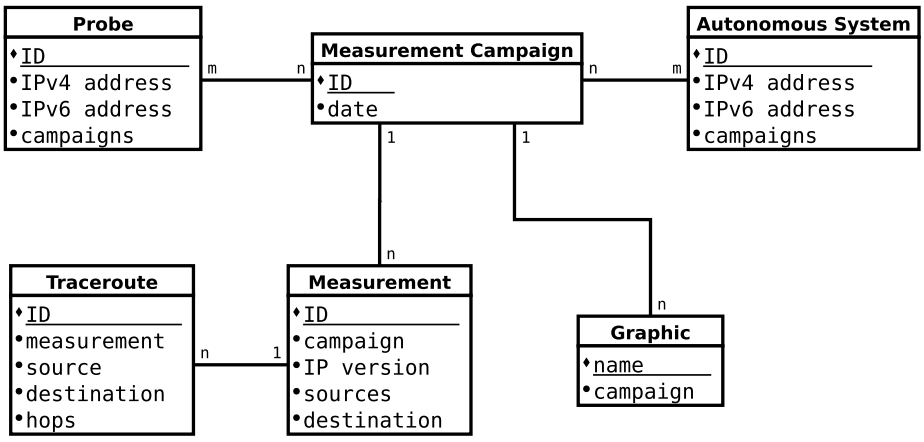

Figure 3: Simplified ER Diagram of the database

2.2 Presentation

2.2.1 Graphic Selection

To get an overview over which results can be visualized in a meaningful way, let us see first, what data is available and how it can be used. We will receive several traceroute responses, from which we can, directly or using supplemental tools, draw out information about:

- The IP addresses of every router along the path
- The AS belonging to the IP address
- The country of the AS
- The round-trip time (RTT) of every hop
- The protocols used

Looking at this information, we can use it to see which ASes are directly connected with each other or which ASes are laying in between them. We can also see the state of the connection by using the RTTs. Grouping the ASes by country the results can also be scaled to a country or region like scale. Another thing this can be used for is the degree of an AS, the number of neighbors it has. When applying all this for both IP protocols and then comparing the respective infrastructure, we can draw conclusions about the current state of the IPv6 deployment.
To see possible solutions for the visualization, related papers can be used as a reference[28]. Firstly, we can use the results grouped by country in combination with the RTTs to create a heatmap, that displays the delay between a source and destination. Since, depending on the scale of the campaign, the sample size can be very high it is very likely that some results with extreme values would

tamper with the average distribution. Hence, the median is to be used to create the heatmaps. Next up, the AS connections can be used to create graph edges. These edges can then be turned into a network graph, which also displays the concentration of the degree of a single node.
The degree specifically offers additional value for other visualizations. To explain this, metrics can be taken from the above referenced paper[28].
The degree distribution depicts the frequency of a node degree in the complete network and can be plotted with the equation,

$$P(d) = \frac{1}{|V|} \sum_{i \epsilon V / d(i)=d} 1,$$

with $d(i)$ as a function that gives the degree of vertex i and $|V|$ as the number of nodes in a network.
Next, the clustering coefficient describes the percentage of interconnected neighbors of a node, with,

$$cc_i = \frac{2n_{LINKS}}{d(i) * (d(i) - 1)},$$

where n_{LINKS} is the number of links between neighbors of node i. It is used to express the average clustering coefficient per degree, as,

$$Cnn(d) = \frac{1}{n_d} \sum_{\forall i / d(i)=d} cc_i,$$

where $n_d = |V/d(v) = d|$ is the number of nodes in the network with degree d.
Finally, the average neighbor degree displays the average degree of neighbors of any node per degree, with,

$$Knn(d) = \frac{1}{n_d} \sum_{\forall i / d(i)=d} \frac{1}{|V(i)|} \sum_{\forall r \epsilon V(i)} |V(r)|,$$

where $V(i)$ are the neighbors of a node i.

2.2.2 Data Access

An important part of the system is to make the collected and processed data available for the user, so he can use it to supplement his own studies and campaigns. A simple method to realize this is to implement a download feature for the data and graphics.
To prevent the user to have direct access to the backend, the downloads are to be made available over the web server. This means that a user needs to be able to select his desired content somewhere on the website and initialize a download afterwards.
While the graphics are all created by the system itself, the same does not apply for the measurement results. Since the measurement are run by RIPE Atlas, the results are therefore also their property. As a consequence, to avoid any

legal issues regarding these topics, the measurements results must not be directly available over the system. This means that, as a workaround, a script is to be distributed as a downloadable option, that, when run, downloads the results over the freely available API of RIPE Atlas. To realize this, the ids of the desired measurements must be offered and communicated to the script.
Concerning the form of the data, it should be available in a format, that allows for an easy implementation into the program of the user and should not require a demanding transformation process. Due to its high popularity, wide application area and being the already existent format of the API results, JSON is the desired data format.

2.3 Architecture

2.3.1 Components

Summarizing the above stated requirements, we can conclude the components needed to implement the monitoring system.
Firstly, a data server is necessary, that can:

- Request data from the platforms RIPE Atlas, CAIDA AS-Rank and CAIDAs AS-Mapping based on the RouteViews Project
- Schedule Paris traceroute measurements over the RIPE-Atlas API using the collected data
- Process the data to create multiple, different graphics
- Save the data and results to the database

Further, a database is to be implemented, that stores both the measurement results and the, processed, data, received from the data server. It must also offer the results to the web server in an efficient way, that does not produce too long waiting time.
Lastly, a web server is needed, that can:

- Retrieve and display the results from the database
- Offer the results to the user over a download interface
- Configure the parameters for scheduling new measurements and initialize the process on the data server

For a look at all the components and at once, see Figure 4.

2.3.2 Interactions

When talking about the interactions of the components, it becomes clear, that the data server will need to have several interfaces to interact with external APIs. This means that the data server will send requests data from the AS-Rank, RouteViews and RIPE-Atlas APIs to get the most recent AS data, AS

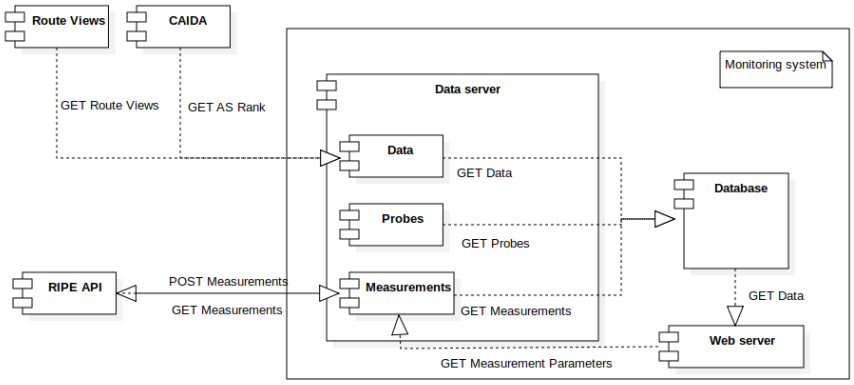

Figure 4: Component Diagram of the monitoring system, consisting of external APIs, data server, database and web server

to IP mapping and probe data respectively. The data server will then save the complete data into the database.
Further interactions from the side of the data server are needed to manage the measurements. At first will ask the web server for the parameters to start the measurement. These parameters are then used to start the measurement. The data server will fill the gaps in the parameters, namely sources and destinations, by getting the collected data from the database. The complete measurement request is then send to the RIPE Atlas API by the data server.
After the measurements are completed, the data server will get the results from the RIPE API and process them. The processed results and graphics are once again injected into the database. From the database the graphics and results are then requested by the web server to show the user on the web page.
Figure 5 describes how the internal operations will be carried out in a sequence diagram.

3 Implementation

To create a functioning system, the requirements and components described in chapter 2 were divided into different scripts and functions. In this chapter the implementation and interaction of the different scripts that build the system will be described briefly.

3.1 Tool Selection

An important purpose of the monitoring system is, that it is freely usable by any organization pursuing to research the Latin American Internet infrastructure. This means that there should not be any conflicts regarding software licenses.

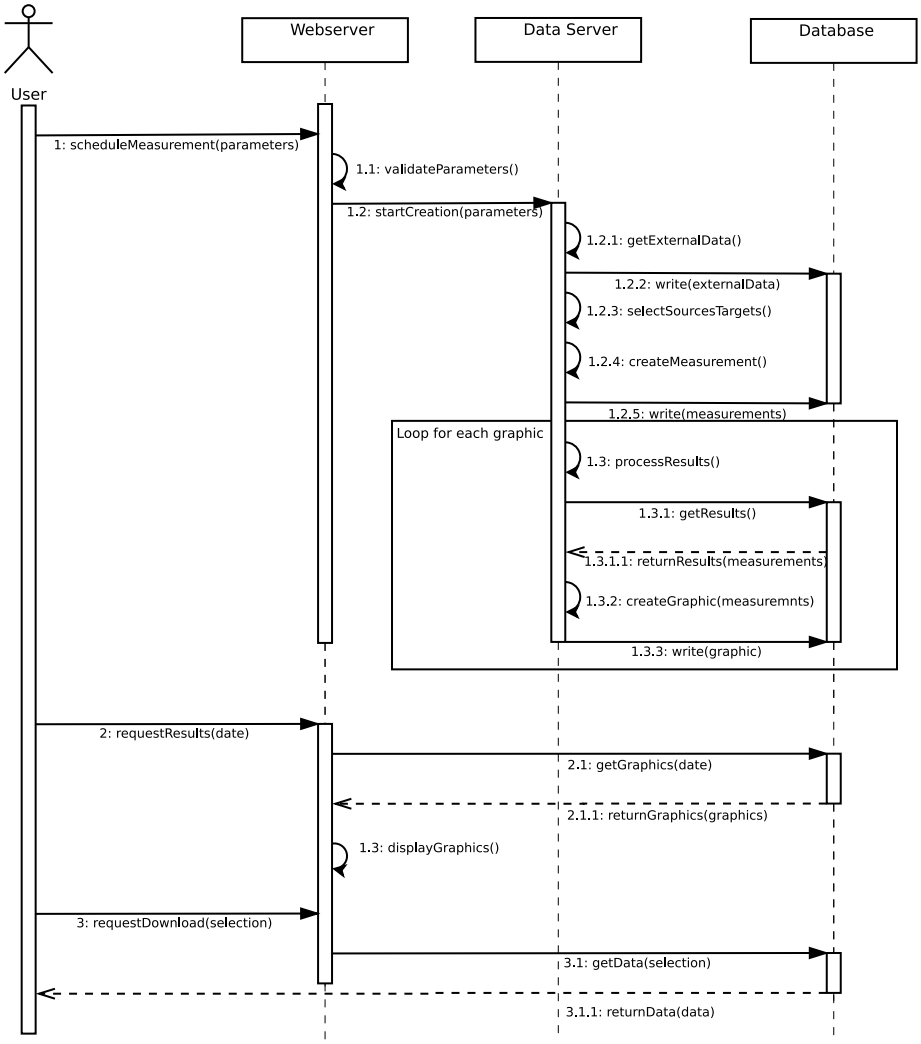

Figure 5: Sequence diagram of the monitoring system

Hence, only open source software or software licensed in a similar way was used to implement the system.
Firstly, the programming language for the data server needed to be selected. To pin down the available options, the requirements needed to be looked at. The programming language should contain support for:

- RESTful API[39] interaction, to communicate with the external data servers
- Database interaction
- Web development
- Data statistics and numerical handling
- A graph library

It should also be easy to learn and maintain, as the system may be tweaked to fit the individuals user's needs.
Taking these points into consideration, two languages were investigated more closely: JavaScript[21] and Python[45]. Both of these very popular languages offer a wide array of open source libraries and frameworks to fulfill the requirements. While in total the two are very comparable and flexible, there were two important points concerning the project. JavaScript has a better performance than Python, largely accountable to the Chrome V8 engine NodeJS is based on. However, Python got the advantage as the superior data analytics tool. As the main focus of the system is to monitor and process large amounts of measurement data, Python was chosen, due to having the edge in that regard. This means that moving forward, the other decisions were based on their compatibility with Python.
Next, the database software for storing the collected data and processed results was to be elected. The first decision to be made was, whether an SQL or NoSQL database was to be used[43]. NoSQL systems are designed to archive large volumes of data of any type and to process them with a high performance. Compared to SQL databases they are often cheaper and faster. On the other hand, the advantage of SQL systems are the execution of complex queries and data integrity. As the focus lies on efficient archiving of data and graphics of different types and no overly complicated queries are needed, NoSQL was selected.
Moving forward, open source, NoSQL databases with good Python support were researched. A closer look was given to MongoDB[19] and Redis[18]. Both are easy to setup, high performing databases that support a huge variety of data types. However, Redis requires some knowledge of Lua to use its full functionality, which makes it harder to use for the general user. Also, MongoDB offers professional support to its clients, which is why it was selected in the end.
The last important tool to be selected was the web framework. Required was a Python web framework with MongoDB and REST support. Django[26] and Flask[22] were the two frameworks considered. Django is a full-stack framework with a vast amount of libraries available. It is very efficient and features

things as URL routing, authentication and a template engine. Flask, on the other hand, is a micro-framework, meaning for production it has be deployed on a web server. This allows for an integration into an web server, that may already be present, such as an Apache server. Flask is very flexible, lightweight and compatible with a lot of outside tools to realize almost any requirement. Besides the rich documentation, Flask also includes RESTful request dispatching and default security features, such as protection against injection attacks and data integrity checks. Even though the initial customization time is higher compared to Django, these points make Flask more optimal for the project.
All the other tools and libraries used will be elaborated in the following description of the implementation.

3.2 Data Collection

When implementing the process of the data collection, it was important to design it in an efficient way. This means, that the existence of redundant data was to be prevented and the unnecessary operations should be avoided. If a user were to conduct, for example, lots of small campaigns at the same day a recurring execution of the initial data collection scripts would be very wasteful. Therefore, before collecting the AS, probe and RouteViews data over the corresponding APIs, a verification needed to be implemented to assert, that the initial collection is only run once per day. This was achieved by checking the database for already existent entries for the current date.

3.2.1 Target Selection

To select the measurement targets, they first need to be retrieved from the CAIDA AS-Rank API. Recalling the criteria stated in chapter 2.1.2, requested is the data of all ASes located in Latin America, run by regional companies. Unfortunately, the API does not include a filter with these parameters for Get-requests. This means that, using the Requests library[27], it has to be iterated over each of the roughly 90000 available ASes. The given data (see Listing 2) can be transformed directly into JSON by Requests. We then extract the latitude and longitude, along with the country parameter, to see if the AS fulfills our geographical requirements. ASes that meet the requirements are then inserted into the database. These build the base of all ASes in Latin America.
Next, to select the actual targets out of the Latin American ASes, they are filtered by their cone size. In the data this is represented by the "asns"-field in the "cone"-array. This is done by using a query that returns every entry with an acceptable cone size and writes them into a pandas DataFrame[32]. Pandas is a Python library for data structures and data analysis tools, where a DataFrame is like a table object for data manipulation with integrated indexing.
The leftover ASes then need to be checked for IPv4 and IPv6 support. Since the data does only contain the number of prefixes of one AS, but no network id, it has to be mapped manually, using the RouteViews data. As it can not be communicated directly with an API for this, the correct URL has to be

```
{
        "data": {
                "clique": "true",
                "source": "e5e3b9c13678dfc483fb1f819d70883c_ARIN",
                "org": {
                        "name": "Level␣3␣Parent,␣LLC",
                        "id": "LPL-141-ARIN"
                },
                "cone": {
                        "prefixes": 516117,
                        "addresses": 1293145968,
                        "asns": 36019
                },
                "latitude": "36.0978209554736",
                "rank": "1",
                "country": "US",
                "name": "LEVEL3",
                "country_name": "United␣States",
                "degree": {
                        "peers": 95,
                        "globals": 5178,
                        "siblings": 9,
                        "customers": 5083,
                        "transits": 5177
                },
                "longitude": "-91.335620170744",
                "id": "3356"
        }
}
```


Listing 2: Exemplary AS data from CAIDA AS-Rank

```
185.115.216.0/22        34762
185.115.220.0/22        34560
185.115.228.0/22        15622
185.115.228.0/24        15622
185.115.229.0/24        15622
185.115.232.0/22        204062
185.115.236.0/22        204195
185.115.240.0/24        42831
185.115.241.0/24        42495
185.115.242.0/24        202365
185.115.243.0/24        51167
185.115.252.0/22        16186
```

Listing 3: Excerpt of an IPASN data file consisting of IP prefix, prefix length and AS number

accessed first, by using the current year and month and to open the right subdirectory. Then, the most recent document will be grabbed with Requests and BeautifulSoup[38], a library to extract data out of HTML and XML files. The process must be repeated for the other IP protocol.
The extension module pyasn allows fast IP address to AS number lookups by returning either the AS belonging to an entered IP address or all IP prefixes belonging to an entered AS. To work, pyasn requires the AS and prefix mappings in a specific format, an IPASN data file. Hence, the RouteViews data is transformed into that format. This leaves us with a file containing IP prefixes and their prefix length and AS number, as seen in Listing 3.
Now, the DataFrame with the ASes can be filtered for entries with at least one IPv4 and IPv6 prefix. As IP prefixes are no valid targets for a measurement, a host address of a prefix needs to be assigned to the AS. Because the intra-domain network of the AS is of no interest to us, any prefix and any host address can be used. To get a valid host address the prefix and prefix length are passed to the hosts-function of the Python library ipaddress[5].
The DataFrame with all valid destinations can now be passed to the measurement creation.

3.2.2 Source Selection

The source selection follows a similar general pattern as the target selection. Meaning, at first we get the probe information from the RIPE API with Requests. A key difference here is, that we can already apply filters for the latitude and longitude in our URL for the Get-request. This drastically increases the speed of the data gathering, as we have to iterate over less data to further filter in our script. The received data (see Listing 4) is then checked, to see if the probe is connected to the network and has both an IPv4 and IPv6 interface. This way, probes that would not respond in both IPv4 and IPv6 measurement

requests are getting disposed of beforehand. Acceptable datasets are then written into a DataFrame.
Now, the AS number of both IP interfaces of the probe is taken from the DataFrame. It is checked, whether both numbers are belonging to an acceptable, Latin American AS, by searching for an up to date entry with this number in the collection of ASes in the database. Probes without results are dropped from the DataFrame.
Lastly, we will group the data structure by both AS numbers of the probes to eliminate duplicate entries. Recall, that only one source per AS is needed, as the intra-domain architecture has no relevance for the project. The finished DataFrame is then written into the database.

3.2.3 Measurement Creation

Before starting the measurement creation, the parameters entered by the user had to be defined.
The first thing to consider were the user limitations of the RIPE Atlas platform, described in chapter 2.1.4. Assuming a user has his own limitations in place, parameters were configured for:

- The number of concurrent measurements
- The number of measurements between a single source and target
- The number of sources selected
- The number of targets selected

The second thing was to define, which fields of the payload of the measurement request, seen in Listing 1, were to be manipulable by the user. It was decided for:

- The description
- The protocol (TCP, UPD or ICMP)
- The number of packets
- The first hop to be included
- The maximum number of hops
- The Paris traceroute configuration
- The billing address

As a final parameter the API-Key of the user was also passed, to be parsed onto the request.
Having defined the parameters, the program can now use them to create the measurements. After the source and target selection are done, a script is called to limit the size of them according to the respective parameter for each. This

```
{
        "prefix_v4":"190.12.160.0/19",
        "status":{
                "since":"2019-07-22T22:10:20Z",
                "id":1,
                "name":"Connected"
        },
        "prefix_v6":"2001:470::/32",
        "description":"Cdlt2 Network Probe",
        "last_connected":1563992518,
        "geometry":{
                "type":"Point",
                "coordinates":[
                        -68.8615,
                        -32.9195
                ]
        },
        "type":"Probe",
        "address_v6":"2001:470:da03:23::36",
        "address_v4":"190.12.188.175",
        "total_uptime":25404868,
        "country_code":"AR",
        "is_public":true,
        "id":27927,
        "asn_v4":28114,
        "asn_v6":6939,
        "status_since":1563833420,
        "first_connected":1538236454,
        "is_anchor":false
}
```


Listing 4: Excert of the probe information received from the RIPE Atlas API

is achieved by taking the DataFrames with the valid sources and targets and returning a sample of the desired size for both.
Following is the repeated execution of the Post-request for IPv4 and IPv6 respectively. One request requires the list of sources, passed parameters for the payload, as well as the IP address of the target. This means that we need to create requests equal to the number of targets times the desired number of repetitions between one source and target, given from the user.
There are a number of things to consider when doing this. As the number of concurrent measurements is limited, the program needs to wait for the current batch of measurements to finish before starting the next one. To avoid running more repetitions than necessary, as each repetition means additional waiting time, it has to be asserted that each batch is containing as many measurements as the given limit for concurrent measurements. This becomes a problem, when considering the desired repetitions between sources and targets by the user. This means, the process can not simply be repeated that many times, because the modulo of the number of targets and concurrent measurements might not be 0. Therefore two loops had to be implemented. The first is repeated $repetition * numTargets/limitConcurrent$ times. The second is nested inside and repeated according to the limit of concurrent measurements. Due to that, when calling the execution request in the second loop, the index of the target DataFrame has to be adjusted to consider the repetition. Lastly, it has to be checked, if there are actual targets left in the DataFrame.
To actually run the Post-request, the payload with the parameters is passed to a subprocess. Afterwards, the response of the API, containing the measurement id, is appended to a DataFrame.
After starting one batch of measurements, the program waits for them to finish. As the API does communicate when a measurement is finished, a fix waiting time was configured, after which the measurements are terminated manually. This is done by sending a Stop-request to the API for each measurement id. To avoid losing results due to terminating measurements too early, the waiting time is configured for 20 minutes. This value was decided upon by experience. Before starting the next batch, the measurements results are saved to the database and the DataFrame with the measurement ids is emptied. For this, a separate thread is started. It receives the measurements ids to get the data from the API. Outsourcing this process saves time, as the next batch does not need to wait for the Get-requests and database writing process to finish.
For the flow of the measurement creation, see Figure 6.

3.3 Data Management

A huge advantage of the usage of MongoDB is their document oriented storage. The data is stored in the format of JSON like documents. The monitoring system can use this very efficiently, as the data it receives from the external APIs is already delivered in JSON. Thence the all the external data can be inserted directly into the database, without any additional processing.
Not reducing the data subsequently increases the storage required. Even so,

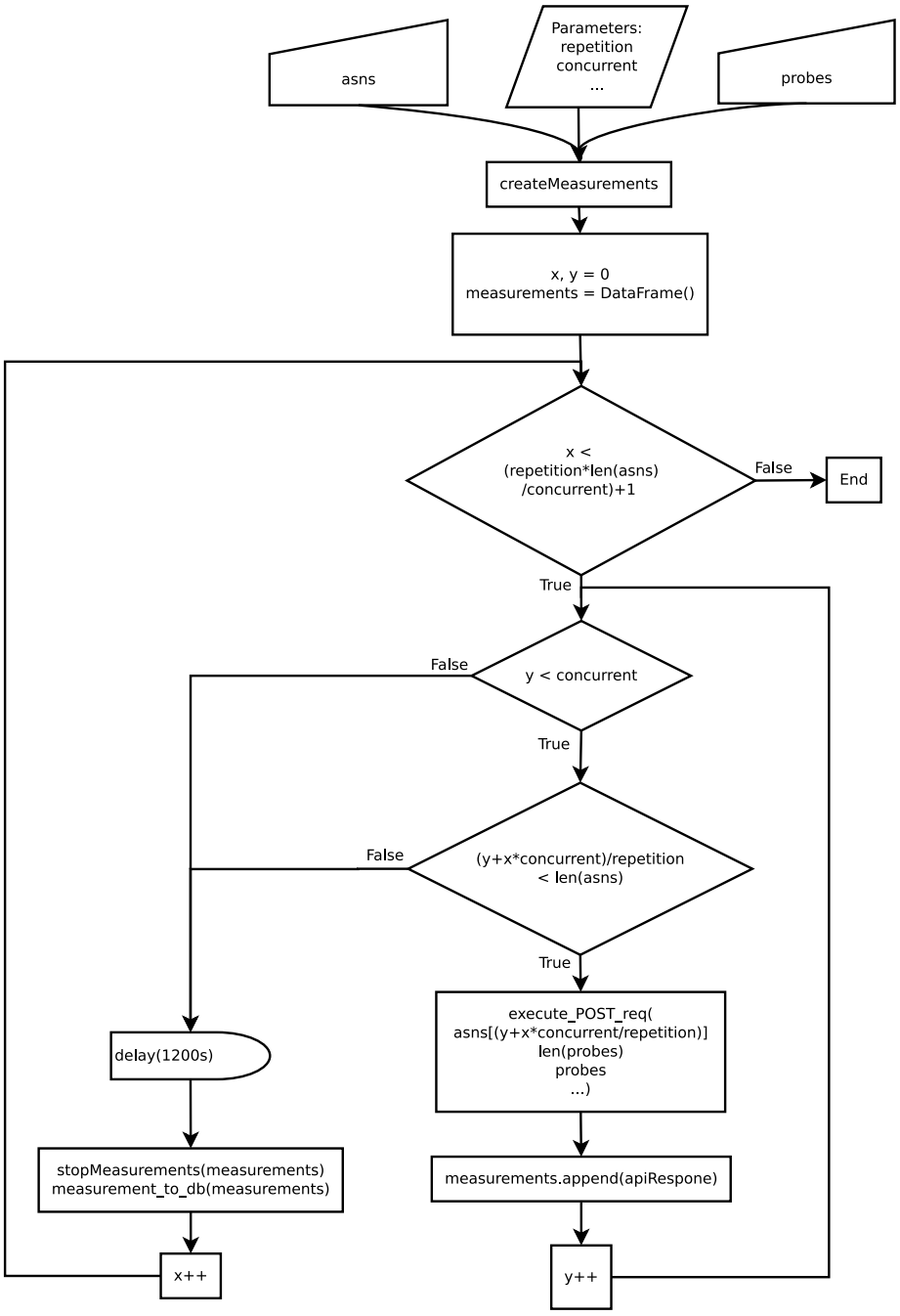

Figure 6: Flowchart of the measurement creation, limited by the number of concurrent measurements

most of the data is essential for conducting meaningful results and keeping the other fields allows us to use the data for other research at any later date. Also, MongoDB inherently uses the WiredTiger storage engine[10]. An engine that provides data modification by multiple users at the same time, checkpointing, compression and more. This further reduces the storage gained by cutting a small portion of the results compared to the potential usefulness in the future.
Another concern was, how the system generates the association between a measurement campaign and its generated data. Besides the two collections for GridFS to store the images in, that will get explained in a moment, the system consists of five collections. One each, for:

- (Campaign-)Dates
- Traceroutes
- Probes
- ASes
- Edges

In the dates collection, a entry with the scheduled date for the campaign will get created, when the initial data collection is started. In all the other collections each document will obtain an additional field with this date, that connects it to the campaign. As with the time needed for a campaign and limitations, most likely, in place, it is very unlikely to conduct multiple campaigns that sprout meaningful results. Thus, the system uses the schedule date instead of the id of the campaign, as it provides better readability, when reading data from the other collections. This is subject to change, according to design philosophy.
To avoid duplicate data sets for the probes, ASes and edges collections, the schedule date field for these is an array. When data is to be inserted here, the program checks, whether an entry for the data is already existent. If yes, the date gets added to the array. If not, the entry is inserted as normal.
Lastly, the images needed to be handled. To achieve this, GridFS[4] is used. GridFS is a specification for storing and retrieving larger documents, such as files or images. The document is divided into parts, so called chunks. Then each chunk is stored as a separate document into the chunks collection. At the same time the metadata of the file is stored into the files collection. This way, every graphic is inserted into the GridFS collections and a reference is passed to a field in the corresponding campaign document. Using this reference the image can be retrieved and put back together for download or display.

3.4 Presentation

3.4.1 Graphic Creation

To create a network graph and implement the metrics stated in chapter 2.2.1, edges between the nodes, the ASes, are required.

Before creating these AS edges from the measurement results, the requirements needed to be defined. To qualify as an edge, there must exist a direct connection between two ASes. To prove this connection, any two adjacent hops of a traceroute got to have an IP address belonging to two different ASes.
Implemented, this means, that, for each measurement of the campaign, the traceroute results get aggregated from the database. To reduce the amount of times the script has to iterate over the hops, all entries without a result are dropped. Now, for each traceroute, the number of the hops are compared. If they are directly adjacent, they get passed to further compare the ASes.
With the help of the lookup-method of pyasn, the IP addresses of the two adjacent hops are replaced with their corresponding AS number. After checking, whether the two ASes are different ASes and do in fact exist, they get inserted into the database collection, along with the IP version used for the traceroute.
Next, the edges from the db can be used to create a DataFrame. To create a network graph, the DataFrame can simply be passed to the LaNet-vi software[14].
When configuring the parameters of the program, some traits of the graph need to be clarified. We can assume, that, if an AS can directly reach another AS, the same is true for the reverse. Therefore, the graph will be undirected. While each edge could theoretically be weighed by, for example, the RTT, it would require additional information to make a profound statement about the connection between two specific ASes, as the RTT may be misleading depending on the context. Also, to simply visualize the network, no weight is required, which is why the created graph is unweighted.
To realize the metrics, the Python package NetworkX[24] provides wide reaching functionalities. A graph object can be created by adding the existent edges. The degree of each given node can be accessed directly over a method that returns a list with the degree of every node.
Examples of the graphs that the system creates can be found in chapter 4.2.2, Result Quality.
To calculate the degree distribution,

$$P(d) = \frac{1}{|V|} \sum_{i \epsilon V / d(i)=d} 1,$$

the program simply counts the frequency of each degree d in the degree list obtained from the graph. The pair of degree and degree count is then plotted (see Figure 17).
The clustering coefficient,

$$cc_i = \frac{2n_{LINKS}}{d(i) * (d(i) - 1)},$$

is available over passing the graph to a function integrated into NetworkX. To create the average clustering coefficient,

$$Cnn(d) = \frac{1}{n_d} \sum_{\forall i / d(i)=d} cc_i,$$

the clustering coefficient of every degree d is divided by the degree count used for the degree distribution (see Figure 19).
At last, the average neighbor degree,

$$Knn(d) = \frac{1}{n_d} \sum_{\forall i / d(i)=d} \frac{1}{|V(i)|} \sum_{\forall r \epsilon V(i)} |V(r)|,$$

is created by dividing the results of the NetworkX average neighbor degree function by the degree count (see Figure 18).
The plotting to an image for all these cases is done by using Matplotlib, a Python 2D plotting library. To produce the final product, a bar chart is plotted and the axes are customized. The x-axis is always the degree. As the bulk of the results will most likely have a lower degree value and only a few with a very high value a normal plot of the charts would be very skewed towards the upper values. Hence, a logarithmic scale is used to counteract this phenomena.
The last graphics to create are the heatmaps. For this purpose, the traceroute data (see Listing 5) has to be retrieved and processed first. The important information here are the source, destination and RTTs. When looking at the structure of a normal traceroute result, you can see that, while the source and destination are easy to locate, the RTTs are embedded in two arrays. This results in a query, that has to unwind every traceroute twice, before the RTTs can be accessed and written into a DataFrame for further processing. For every traceroute of a campaign, the median of the RTTs of the last hop with valid results is taken. The last hop is taken, so the result is the closest to the actual destination and therefore the most accurate.
With the help of Pyasn and the AS-data, the IP addresses of the source and destination are now translated to their respective country code. Following, the data is grouped by destination and source country and the median of the RTTs for each country pair is built.
After transforming the DataFrame into a pivot-table a cluster-map is build, using the on Matplotlib based library Seaborn. The goal of the Euclidean clustering of the data is to increase the readability of the graphic. Clustering will produce a heatmap with country pairs placed next to each other that have similar RTT values.
To further increase the readability, the indexes of the cluster-map are saved to be used for the heatmap of the other IP version. This will result in two heatmaps with the same indexes, that can be compared to each other very handily.
The re-indexed pivot tables are now transformed into a heatmap and labeled accordingly with Seaborn, before getting saved into the database.

3.4.2 Web-Hosting

The basic structure of the implemented web app, created with Flask, exists out of app routes, forms and HTML templates. The templates are further using the Bootstrap toolkit[41], a front-end framework, to apply different styles to them (see Figure 7).

```
{
"af":4,
"dst_addr":"148.243.99.1",
"dst_name":"148.243.99.1",
"endtime":1561477097,
"from":"200.59.16.60",
"fw":4790,
"group_id":22101633,
"lts":277,
"msm_id":22101633,
"msm_name":"Traceroute",
"paris_id":0,
"prb_id":2928,
"proto":"TCP",
"result":[
        {"hop":1,
         "result":[
                {"from":"200.59.16.33","rtt":2.784,"size":88,"ttl":64},
                {"from":"200.59.16.33","rtt":1.853,"size":88,"ttl":64},
                {"from":"200.59.16.33","rtt":1.842,"size":88,"ttl":64}
        ]},
        {"hop":2,
         "result":[
                {"from":"186.176.7.73","rtt":2.707,"size":28,"ttl":253},
                {"from":"186.176.7.73","rtt":3.05,"size":28,"ttl":253},
                {"from":"186.176.7.73","rtt":2.051,"size":28,"ttl":253}
        ]}],
"size":48,
"src_addr":"200.59.16.60",
"stored_timestamp":1561477206,
"timestamp":1561476687,
"type":"traceroute"
}
```


Listing 5: Excerpt of a traceroute result

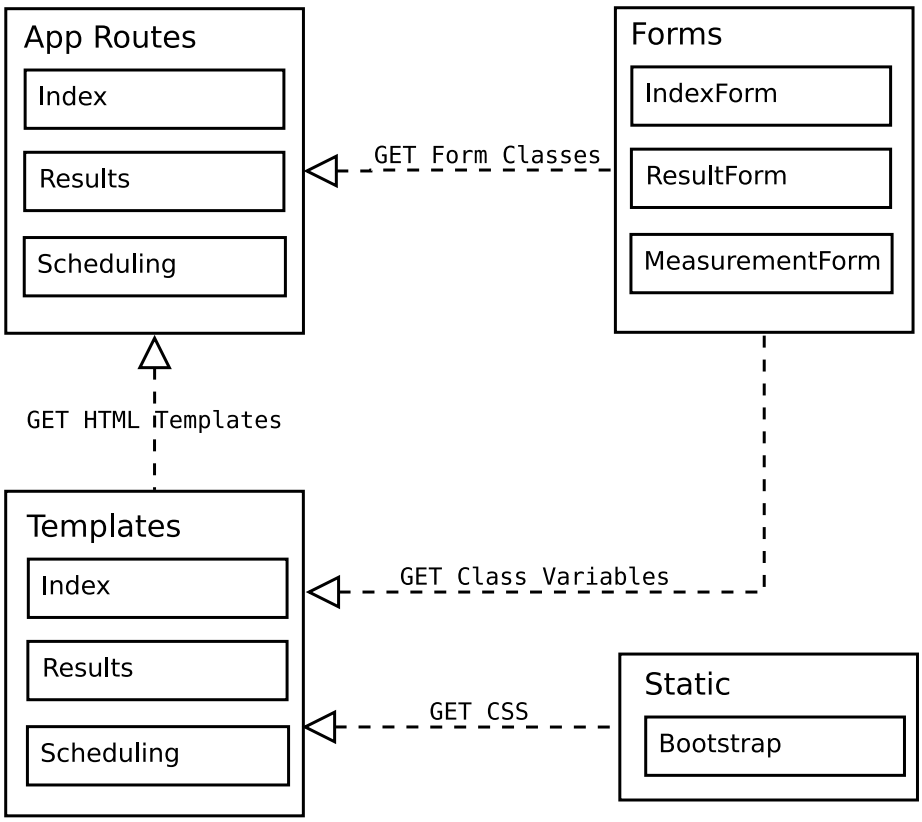

Figure 7: Components of the Flask web app and their interactions.

The app routes define a page of the website by specifying the URL, the methods, as GET and POST, and a function, which is run, when the page is called. Inside the function other functions and scripts can be run and the web page can be put out on the screen by returning HTML code and a form. The website of the system contains three different pages. Every page has a app route that renders the output by using its own HTML file and form.
The index page simply redirects the user to either the page to schedule new measurements or to the measurement results for a selected page, depending on his choice. For the results, the user can choose any campaign in the database. This is done by getting the schedule dates of every campaign from the database. To achieve this, Pymongo[33], a Python distribution to work with MongoDB, can be included inside the app configuration in Flask.
The results page loads every available graphic onto the page. This is done by requesting the image from the database via GridFS. A combination of the schedule date and graphic type, e.g. heatmap_v4, is used to assign the images to the right containers. For better readability, IPv4 and IPv6 versions for each graphic are placed adjacent. Below the graphics, a download can be requested. The user can specify, which data he wants to download. To realize this feature as a single file download, the program gets the requested data and images from the database and then transform them into Zip file. This Zip file is then returned as a response.
Lastly, there exists a page to schedule new measurements. Here, the user can specify the different parameters for his measurement campaign. After successful confirmation of the parameters, they are passed to the data server over a new thread. The creation of a new thread is crucial, as the web app would have to wait for the measurement campaign to finish, before continuing browsing, otherwise. Further, the scheduling page saves some parameters as a cookie, so a user can keep some data, that is unlikely to change, such as his billing address or API key, for his next campaign.
For implementing the forms, WTForms[44] were used. WTForms offers flexibility and validation for HTML forms and is handily integrated into Flask and the templates. It offers multiple fields for user input, such as integer, select or text fields, and includes a validation. With the help of these functionalities, a fitting field for every user input was created. They are restricted to only allow specific values, contain default values and require to pass validation before the web app will use its data.
The whole Flask web app is deployed to an Apache Webserver using the mod_wsgi package[29].

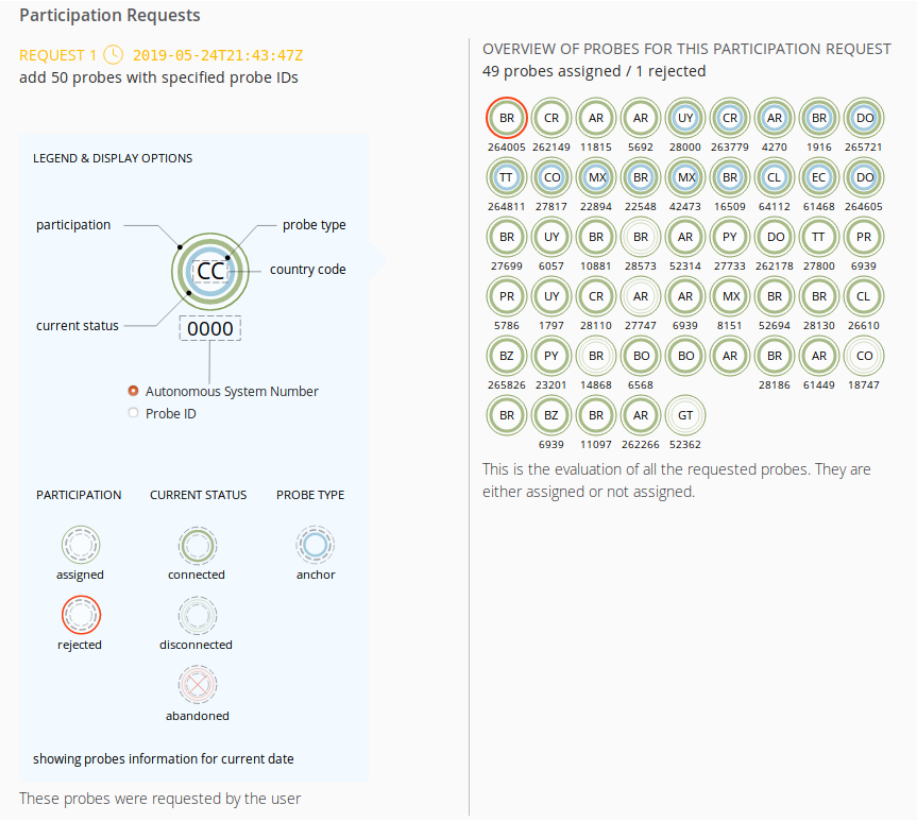

Figure 8: Screenshot of the participation of requested source probes for a measurement, taken from the RIPE Atlas website. Accesed on 27th July 2019

4 Analysis of the Implementation

4.1 Analysis of the Data Collection and Processing

4.1.1 Performance and Reliability

As the systems needs to communicate with several external APIs, its performance and reliability is highly dependent on the availability and communication with the external platforms.
During all of the system tests, not a single problem with an API being unreachable or retrieving results occured. There did exists cases, where single probes did refuse to participate in a campaign (see Figure 8). Even so, as a non participation does not cost any credits and the number of source probes to assign to a measurement is unlimited, this only slightly hinders the systems, when a limit for sources is specified.
Concerning the performance, the system was tested on a machine with 8GB

of RAM and an Intel Core i5-4200M Processor. The load times for on the website were usually instantaneous. To provide the downloadable data, the systems needed between one and four seconds, depending on the amount of data selected. To ensure a stable system, the data is usually iterated over serially. This resulted in tests without any hardware problems, however the execution time for some processes was increased as a results of this.
Waiting time is required when creating measurements. As a batch of measurements needs roughly 25 minutes to finish to reliably yield results and limitations by RIPE Atlas often requires a campaign to run multiple batches, this waiting time is to be expected and can not be avoided. As for the performance of the data processing, the measurement results are written into the database directly after the measurement is stopped. Since another thread is used to achieve this, the start of the next measurement is not delayed. As every hop of every traceroute needs to be revisited by the system to create the network edges and data for the graphics, this part needs the most time outside of the creation. For a campaign with 50 sources and 1750 targets with 10 packets each, the system on the test machine needed around four hours to process the data and create the graphics for both protocols.
Overall the system is stable and delivers reliable results. As at times there are three to four threads running simultaneously the host machine needs to be able to handle this workload. While the measurement creation and processing requires some time the finished graphics can be accessed immediately, without any problems.

4.1.2 Data Selection and Measurement Creation

The data selection offers various different options. The system selects targets and sources according to the requirements. Most importantly, it can reliably be asserted, that the selected targets and sources are active in a network located in Latin America.
Also, the system offers the user a high amount of customization, when setting up the measurement process. Over the web page (see Figure 9) , the user can work around his limitations and amount of available credits, by setting:

- The number of sources and targets to participate in the campaign
- The amount of times each source targets a destinations
- The number of packets for each measurement
- The number of concurrent measurements

Furthermore, additional options to adjust the traceroutes are available:

- The protocol
- The first hop
- The maximum number of hops

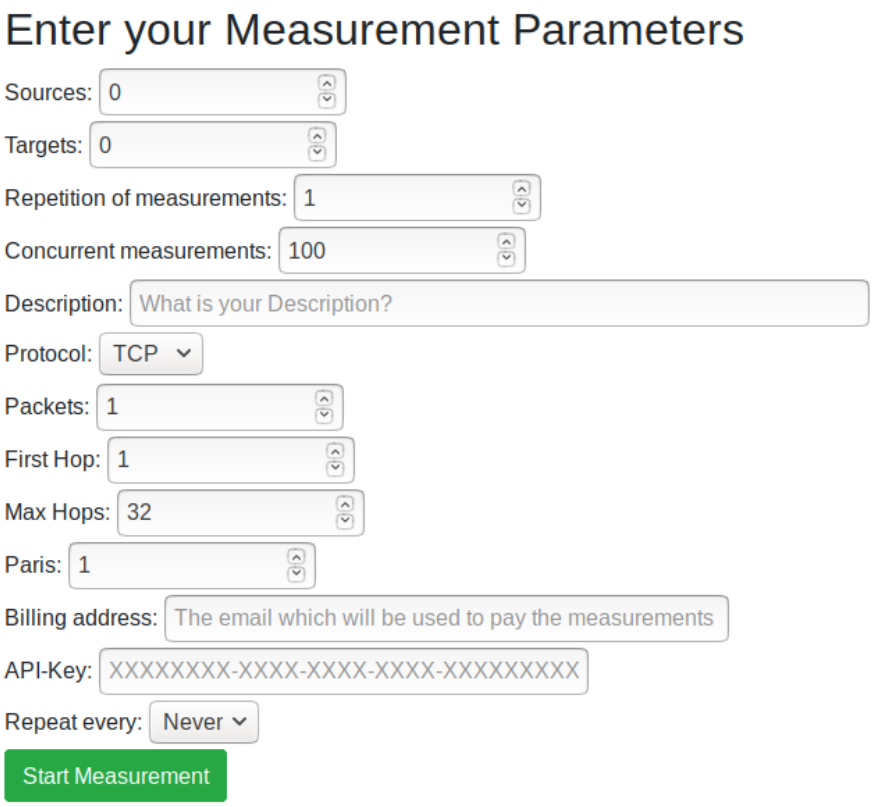

Figure 9: Screenshot of the measurement creation page and the available options for the user to manipulate

- The Paris traceroute configuration
- The number of packets, again

Lastly, the user can also configure options on a more abstract layer, such as:

- The description
- The billing address
- The API-Key

A feature that is not available right now, is to reuse the exact same measurements used in a specific campaign in another. However, this could easily be implemented by adding a field that specifies which probes and ASes were used in a specific campaign into the corresponding collection documents. In consideration of development time, it was refrained from offering more choices concerning the reuse of targets, as the priority of the data is, that it is up to date.
Lastly, the design behind the scheduling has to be examined critically. As of right now, the system works on the assumption, that the user has his own RIPE limitations in place and can therefore run a campaign without consideration for daily restrictions for credit usage. Consequently, the campaign size is limited by the daily limitations. As a workaround, the system could pause the measurement creation after reaching the limit and continue the process at the next day.
In conclusion, the data selection is reliable and highly flexible, that could be expanded with further features, if so desired.

4.1.3 Data Quality

Firstly, lets talk about the quality of the probe data. The probe data is complete and can be filtered effectively and efficiently to deliver reliably available sources that contain all the necessary information.
Next, the AS data, as well as the RouteViews data accessed over CAIDA, also, provide us with all data needed for our purpose. However, a bit more processing and more intense filtering is needed here to make it easy to work it.
The high quality of this data, together with the probe data, is essential, as the system uses this data as the foundation and works on the assumption, that it is correct.
The best way to analyze the quality of the measurement result data is to take a look at the results themselves and see what information we can extract from it. In the following, we will refer to the results seen in Listing 5 and, for a more clear view and another example data set, Figure 10.
We can see that the probe id and its IP address, as well as the destination address are available for each traceroute. The addresses can always be converted into the corresponding AS numbers with Pyasn on demand. The data also contains entries for the measurement id, a timestamp and the parameters used for the (Paris) traceroute. Therefore, each traceroute can always be put into context.
When looking at the results of the traceroutes specifically, we can see that all

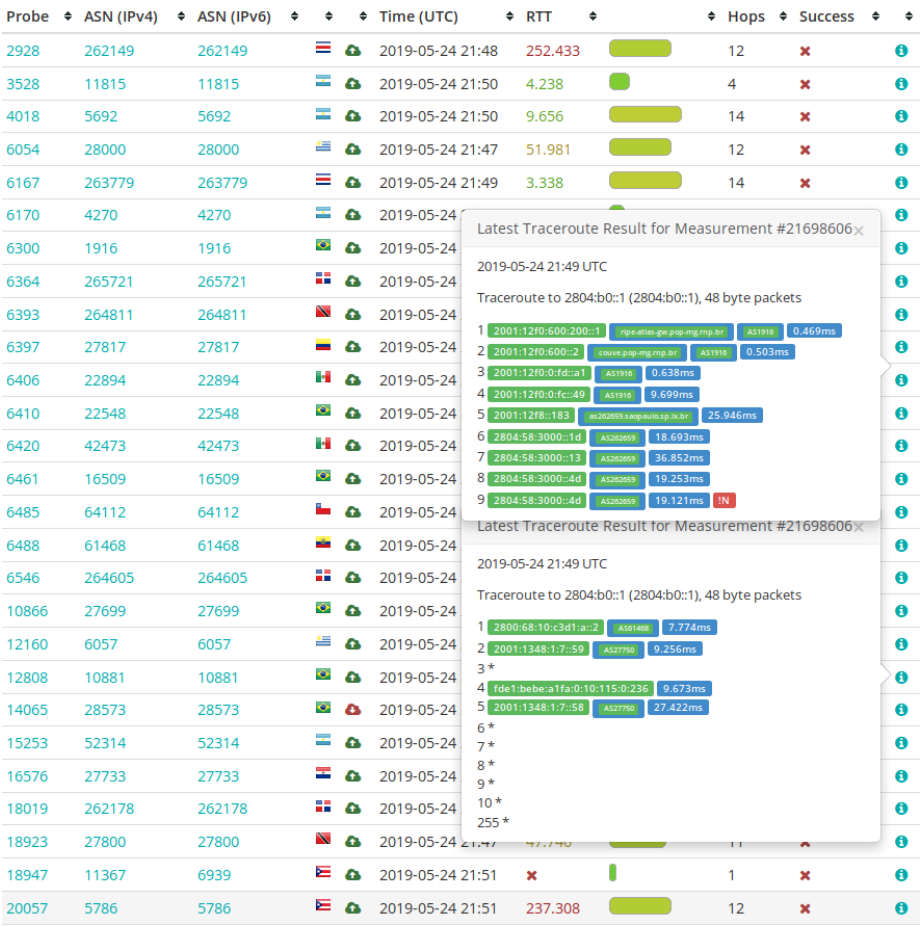

Figure 10: Screenshot of measurement results, taken from the RIPE Atlas website. Accesed on 27th July 2019

the information normally included in a traceroute are provided. Further, Paris traceroute greatly improves routing. For each traceroute almost all packets take the same path along the network and anomalies are avoided. Unfortunately, sometimes, some packets are lost along the way and the results may contain holes. These have to be taken care of, while processing the data and in turn cause additional workload for the system.
Another thing to notice is, that the last hop usually does not respond. This happens due to only knowing the network id of an AS when specifying the destination. Therefore, there is no guarantee a device to respond to the selected IP address in the network exists. But since the intra-domain network of the AS is not relevant for the rest of the program and the packets reach a router in the target AS anyways, this can be ignored.
Another good quality is, that the JSON format, the data arrives in. It does not require any additional processing or converting to be inserted into the database or to be used by the other parts of the program.
In conclusion, the data of the measurements is very detailed and delivered in a advantageous format. Every relevant information can be taken either directly from the RIPE API data or indirectly by using additional resources, such as the IP to AS number mapping available. At times, the hops do contain holes in the responses though, which can not be improved easily.

4.2 Analysis of the Presentation

4.2.1 User Experience

To analyze the user experience, the user interface has to be inspected closely. As all interaction between the system and the user happens over the web app, let us take a closer look at the different sites.
Firstly, the index page (Figure 11). It quickly introduces the user to the project and mainly serves as a bridge between the other two pages. The campaign results are displayed in a descending timeline, to assure that the most recent result will always be on top.
The scheduling page (Figure 12) allows the user to customize all measurement parameters. To reduce user errors and increase security, WTForms fields and validators are used to assert that only the correct values in the correct range can be entered. This means for example that the value for the field “Packets” can not drop below one as that is the minimum number of packets. For most fields standard values exist to reduce trivial inputs. The values for the text fields are stored in a cookie, so the user does not have to enter reoccurring data, as his API-Key, a second time. In case a value is missing or an incorrect one is entered, the page will keep the other values and mark the wrong entries with a red box and an error message. To help in filling out the form, hovering over a field shows a description of it. After successfully scheduling measurements the user is redirected to the index page, so he can view the results. However, as the campaigns needs some time to process the user has to wait before he can see his own results. While the campaign is running the user is unable to see the

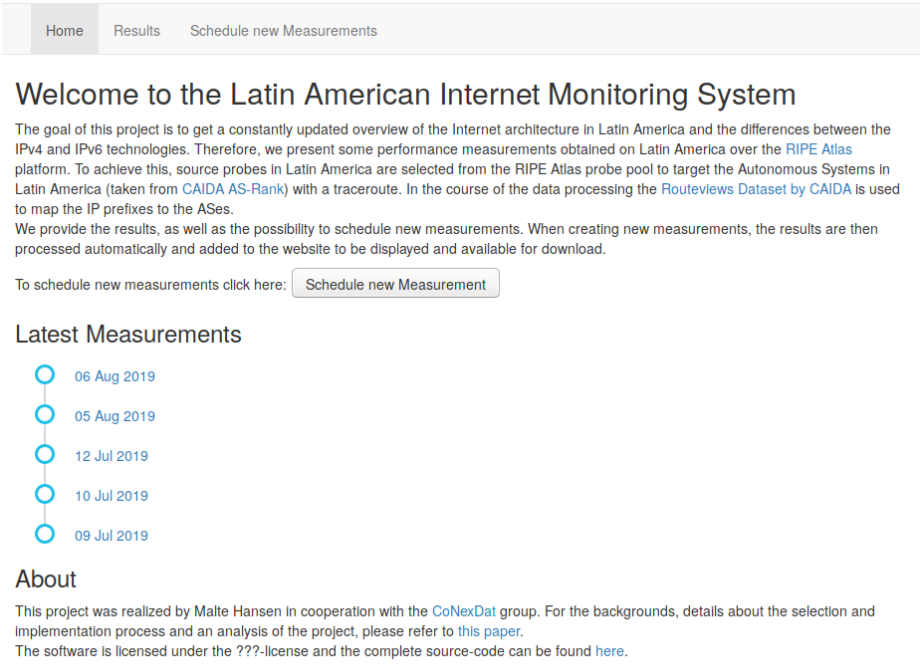

Figure 11: Screenshot of the index page of the web app. Taken on the 19th of August 2019.

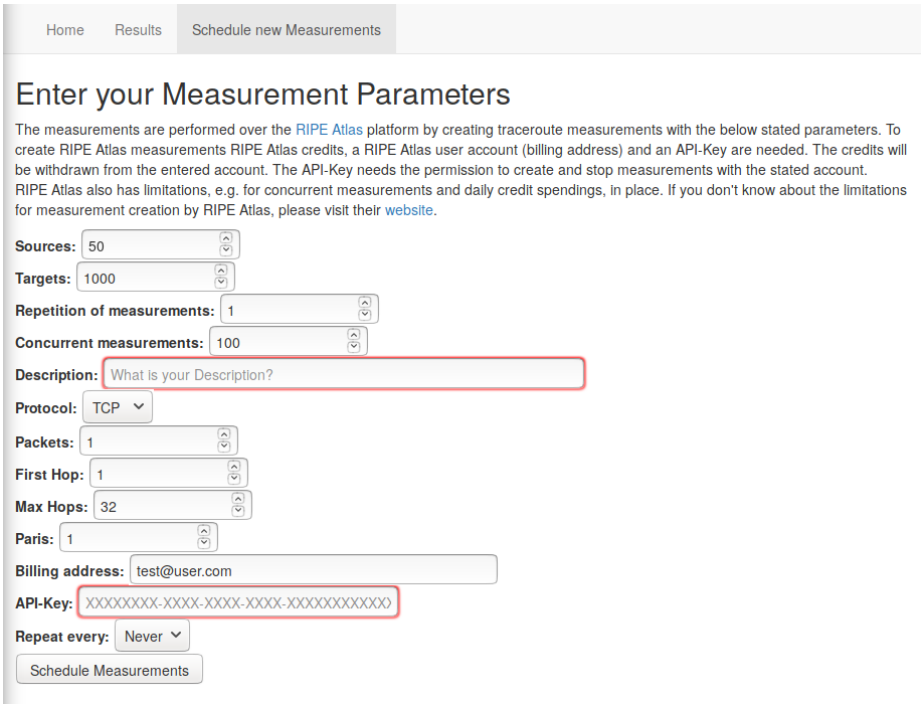

Figure 12: Screenshot of the scheduling page of the web app. Taken on the 19th of August 2019.

progress of the campaign in the web app.
Lastly, the results page (Figure 13 & 14) displays all the graphics of a campaign. Each graphic has a headline to explain the figure. The campaign date can be changed at the top, so the user does not have to return to the index page, each time he wants to view different results. It can be exactly selected which data is to be downloaded and which not. This secures that the user gets only the data he wants. To avoid ownership issues, the "Download Script" field, also saved as a cookie, further provides the user with a script to download the measurement data directly from the RIPE Atlas API. An explanation on how to run the script is included.
On the whole website, every paragraph has a brief explanation, so the user knows what he has to do or what the data is and how it was created. External links aid the user if he needs additional information. A navbar at the top of each page allows the user to freely switch between the pages. As stated in chapter 4.1.1, while the measurement creation needs time to conclude, the interactions on the web page are without delay.
All in all, the system offers a lot of convenient features and information for the user, is responsive and loads quickly. Due to the limited development time the design and some additional features to further improve the usability were not

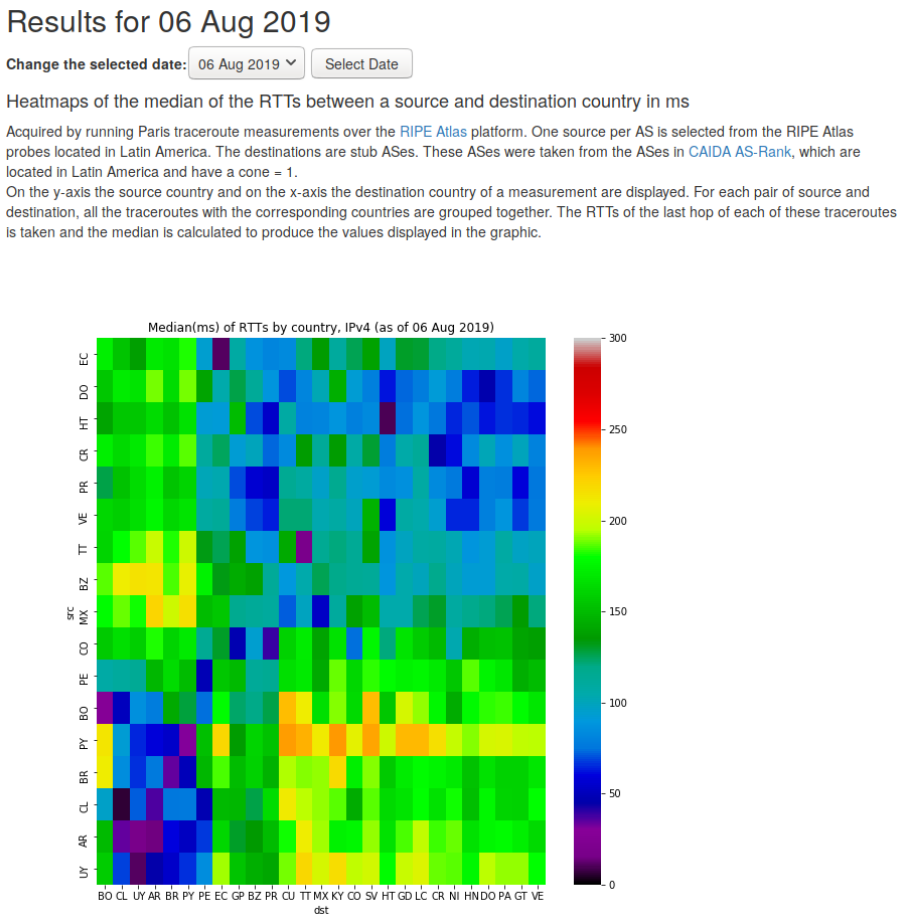

Figure 13: Screenshot of an excerpt of the results page of the web app. Taken on the 19th of August 2019.

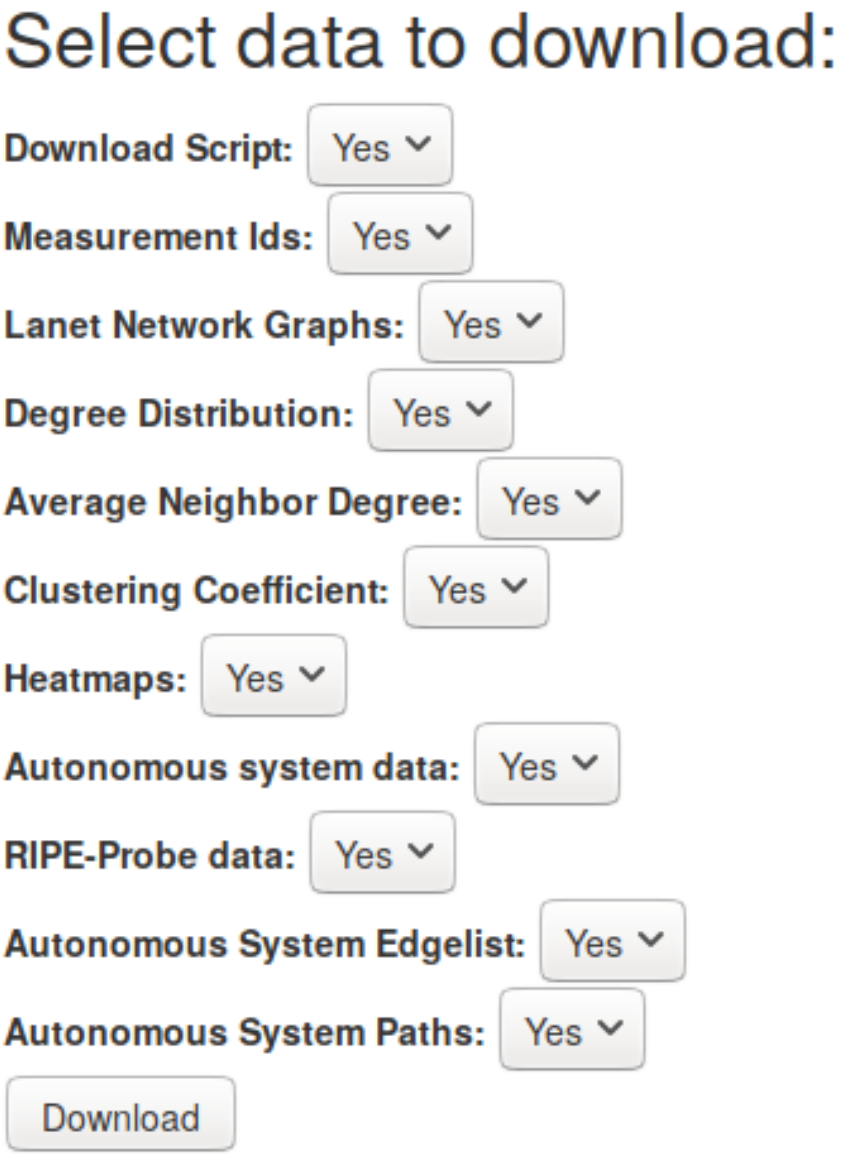

Figure 14: Screenshot of the download function of the results page of the web app. Taken on the 19th of August 2019.

included though. Especially the current state of a measurement would be very interesting for the user.

4.2.2 Result Quality

The system successfully creates graphics for the IPv4 and IPv6 network respectively.

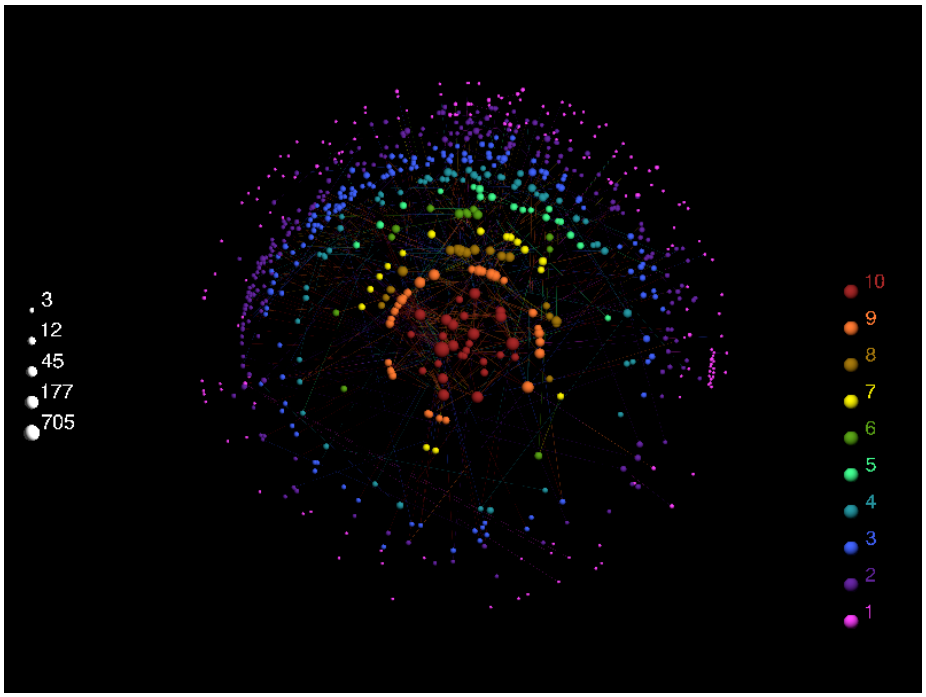

Figure 15: The k-core decomposition of the Latin American IPv4 Internet topology.

The first thing to analyze are the k-core decomposition of the network graphs (see Figure 15 & 16). The AS-edges created are used to build the graph. On the left we got a degree indicator and the shell index on the right. We can associate a shell and degree with each node in the graph. The nodes of the inner shells are more densely connected and have a higher degree size. This results in information about the differences in number of nodes, their degree, the density and, to a certain extend, their inter-connectivity between the two architectures, when comparing the maps. For our test results, a higher number of nodes can be seen in the IPv4 topology. This comes with more shells and higher degrees for the nodes as well. Hence, we can deduce, that the smaller number of nodes in the IPv6 topology hints to a smaller network. While the distribution of nodes between the shells seems mostly identical, a difference can be seen between the fifth and seventh shell. Here, the IPv4 shows a lesser frequency than the IPv6

network. However, as no numbers can be given without making the graph less readable, the supplemental graphics have to be used to confirm this suspicion. Overall, the decomposition graphs offer a first overview of the network and allows for some assumptions that need additional data to verify.

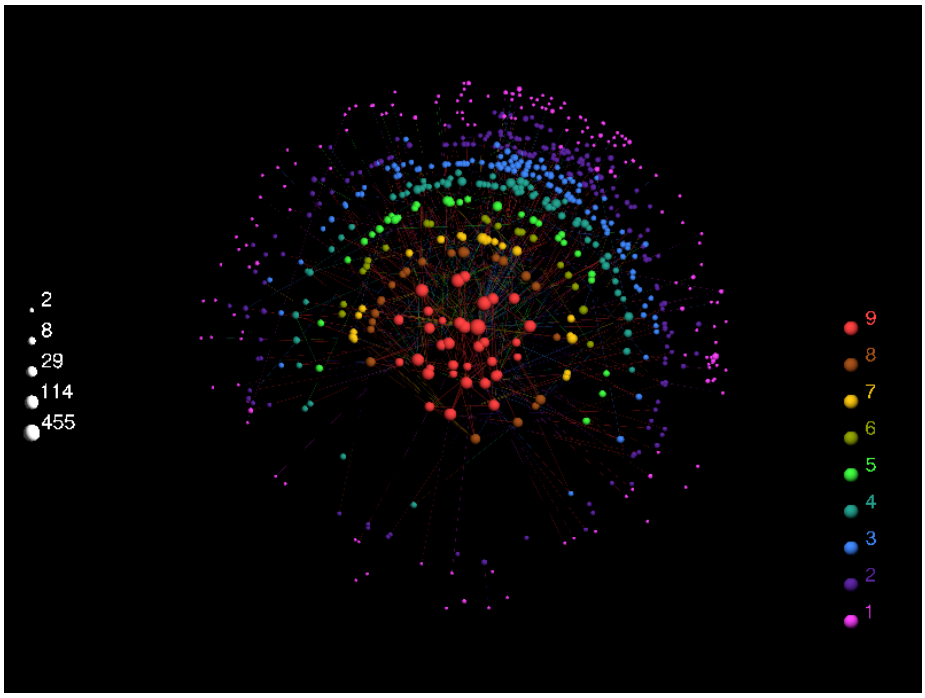

Figure 16: The k-core decomposition of the Latin American IPv6 Internet topology.

Next, the degree distribution displays the count of each given distribution with a logarithmic scale (see Figure 17). On the x-axis the degree of a node is shown, while the count of each occurring degree is displayed on the y-axis. The use of a logarithmic scale increases the readability of the graph, as it would otherwise be very skewed towards the higher degree values. However, single values, as the one node with a degree of 705, seen in Figure 15, are barely visible due to this. Comparing the count numbers, can once again confirm, that the IPv4 network contains more nodes than the IPv4 network. Noticeable is, that the IPv6 graph shows a slightly higher count for the degree sizes between five and eight than the IPv4 graph. This could explain the apparent difference from the k-core distributions. We can also notice that the IPv4 architecture has a higher count for larger degree sizes. This is suggesting, that the IPv4 network has a number of larger ASes that are connecting the smaller ASes with each other, while the IPv6 network has more ASes in the middle shells to to compensate for the lack of large, densely connected ASes in the most inner shells.

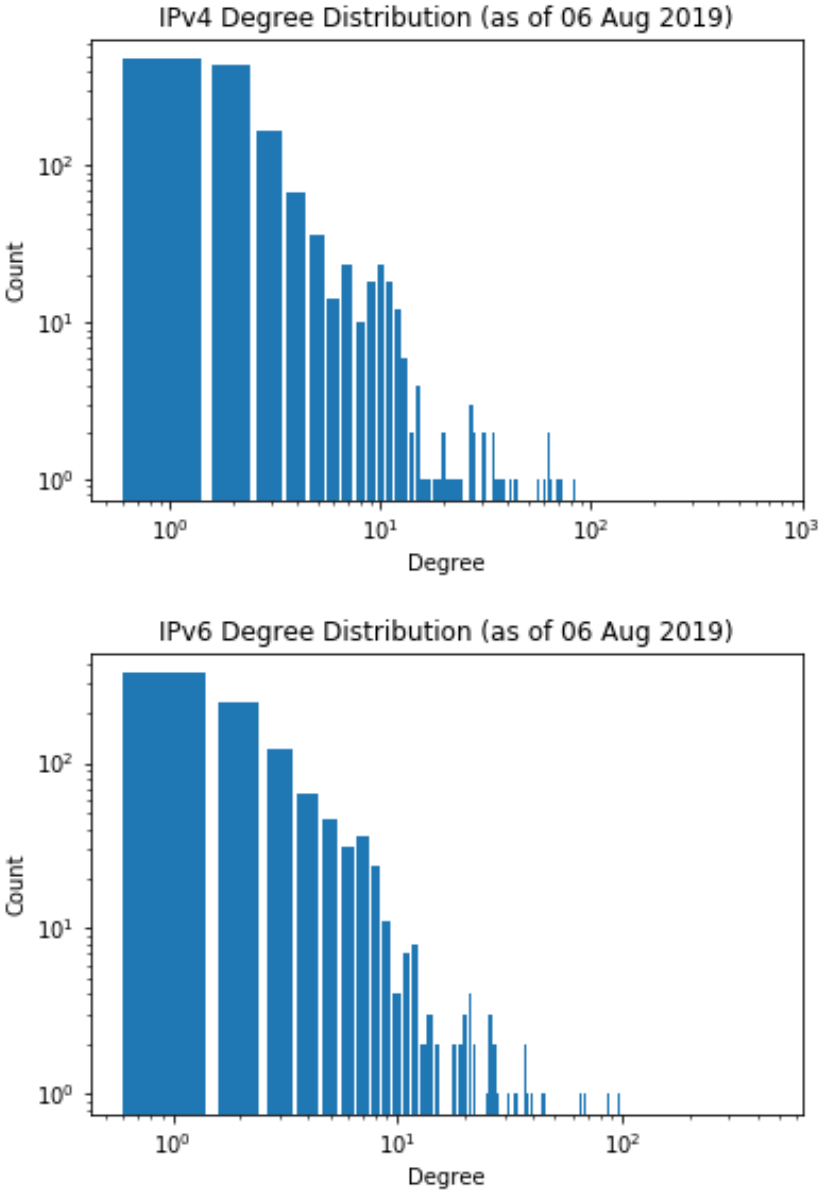

Figure 17: Graph of the degree distribution of ASes in the Latin American Internet topology

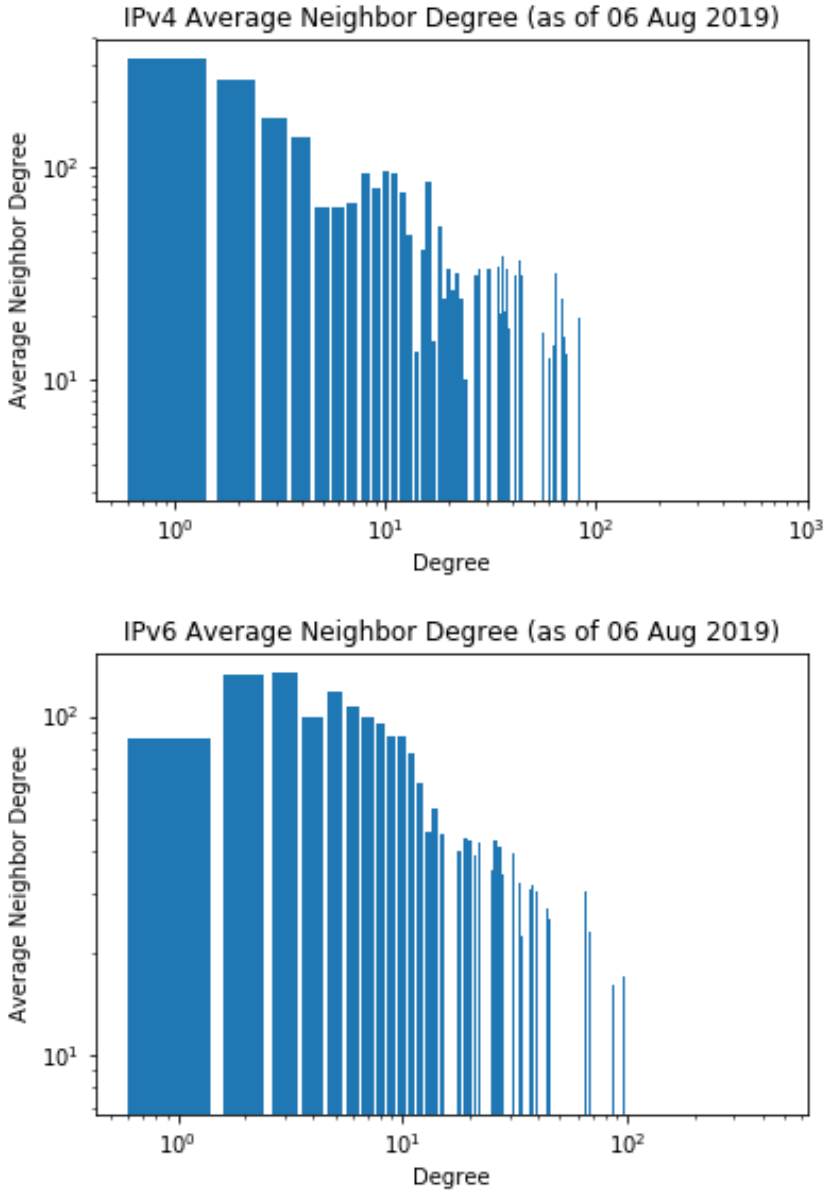

Figure 18: Graph of the average neighbor degree of ASes in the Latin American Internet topology

The average neighbor degree diagram, provides us, again, with the degree size on the x-axis and the average neighbor degree on the y-axis (see Figure 18). The graphs show, that the average neighbor degree gets lower with a higher degree size. This is very reasonable, as nodes in the center will be connected to many outer nodes with a low degree, therefore reducing the average. Striking are the different peaks of the graphs. While the IPv4 degrees with a size of one have the highest average neighbor degree, the IPv6 degrees with a size between three and eight show a higher average in comparison. We can conclude, that in the IPv4 network more nodes with a low degree are connected directly to the inner shells. Meanwhile, in IPv6 topology more lower degree nodes are connected to nodes in the middle, which explains the lower number for the degree size one, as well as the higher average for the sizes between three and eight.
In the graph of the clustering coefficient on the y-axis, versus the degree size on the x-axis (see Figure 19), we can see that there is no value for the degree value one. This has to be the case, as a node with only one neighbor can not have connected neighbors. Further, the clustering coefficient mostly gets smaller, the higher the degree. As these nodes connect many isolated nodes, this is, again, to be expected. Interestingly, both graphs are showing some outliers to the general tendency of a decreasing clustering coefficient at several points. This shows that, the relatively small sample size can produce and display unexpected results.
Lastly, the heatmaps are to be analyzed (see Figure 20 & 21). They display the median of the RTTs of every traceroute from a source country, on the y-axis, to a target country on the x-axis. It is very easy, to identify different values, due to the high contrast in colors. For ordering the countries on the x- and y-axis the Hierarchical Clustering Algorithm[35] is applied onto the results of the IPv4 measurements. The same order is used for the IPv6 figure. This way correlated results are displayed close to each other and country pairs of similar value are grouped together. Further, the identical layout lets us compare the differences between the architecture directly. Together, this allows us to quickly get an impression about the quality of the connection between two countries and locate weak links and differences, as the high IPv6 results from Costa Rica to any country or the lack of results when trying to reach Guadeloupe over the IPv6 network. In total, we can see that the IPv4 architecture still has better RTTs on average. But, some countries, such as Uruguay for example, already get better results for IPv6 traceroutes and seem to be developing their IPv6 network well.
In conclusion, the different graphics display their respective metric or statistic and have a layout with great readability. The headlines of the graphics provide context concerning the timeline. If we combine the information from each, we can come to various assumptions that can be substantiated. It has to be noted though, that, as the number of results to be included in a single graphic is very high, single values can be hard to identify. The results display a vast overview over the whole networks and allow for comparisons between them, but to investigate the details, the data itself has to be consulted. As this can be done with the download feature, the level of the displayed results fulfills its purpose.

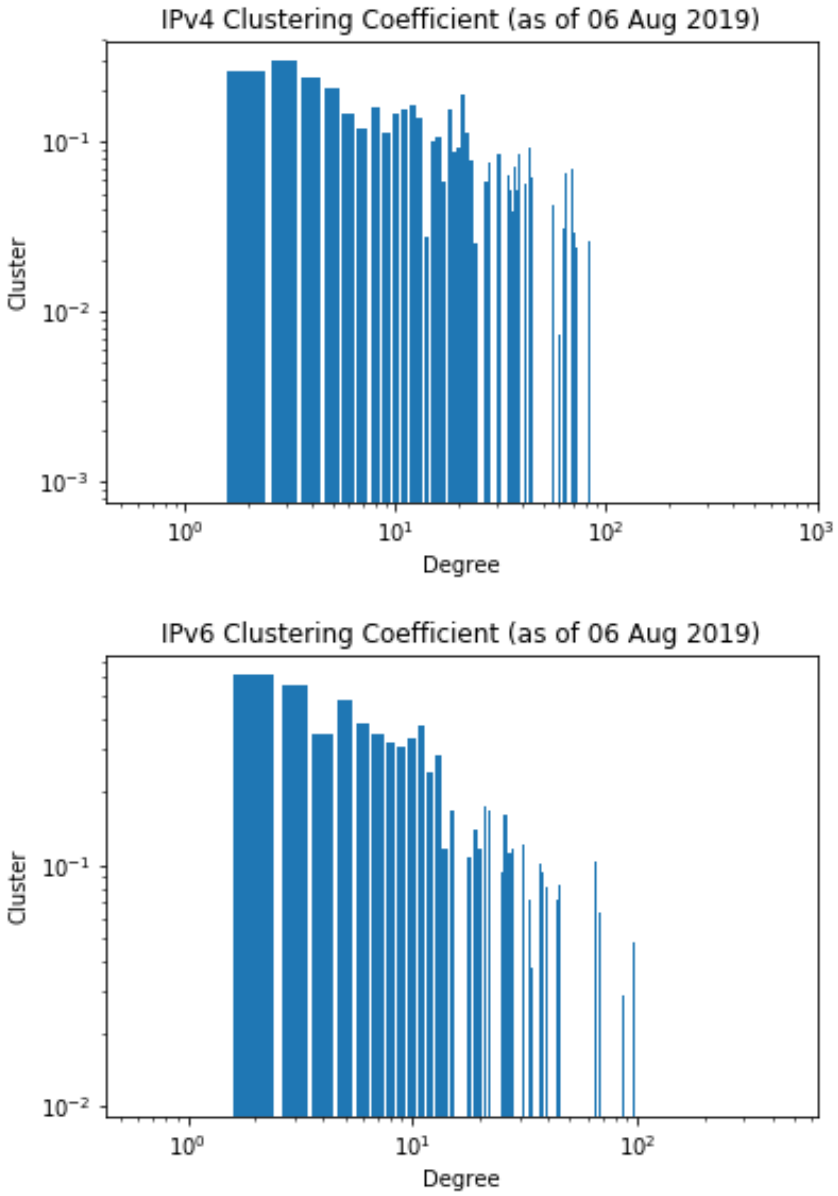

Figure 19: Graph of the clustering coefficient versus degree for the ASes in the Latin American Internet topology

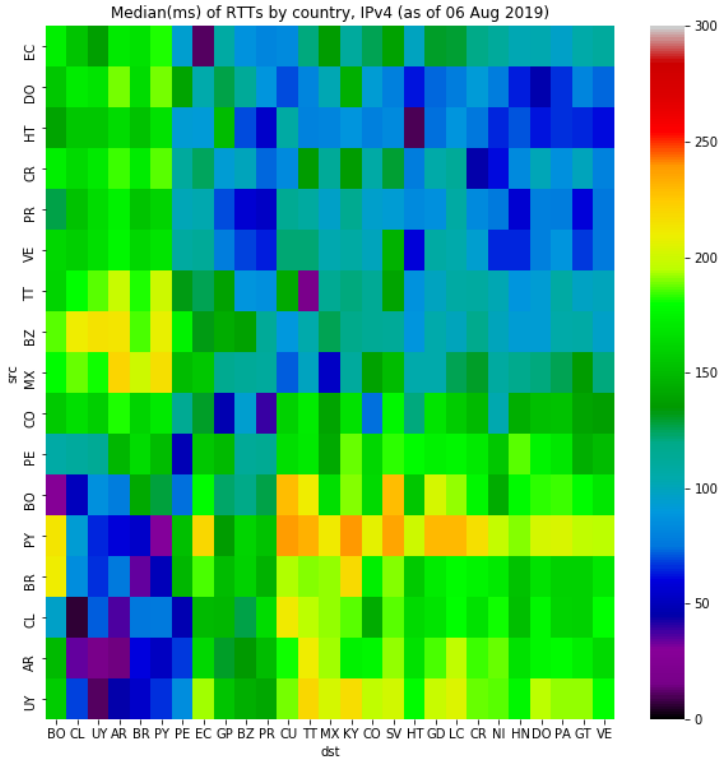

Figure 20: Heatmap of the median Round Trip Time between sources and targets in Latin American IPv4 Internet topology by country. Black results had no data to analyze.

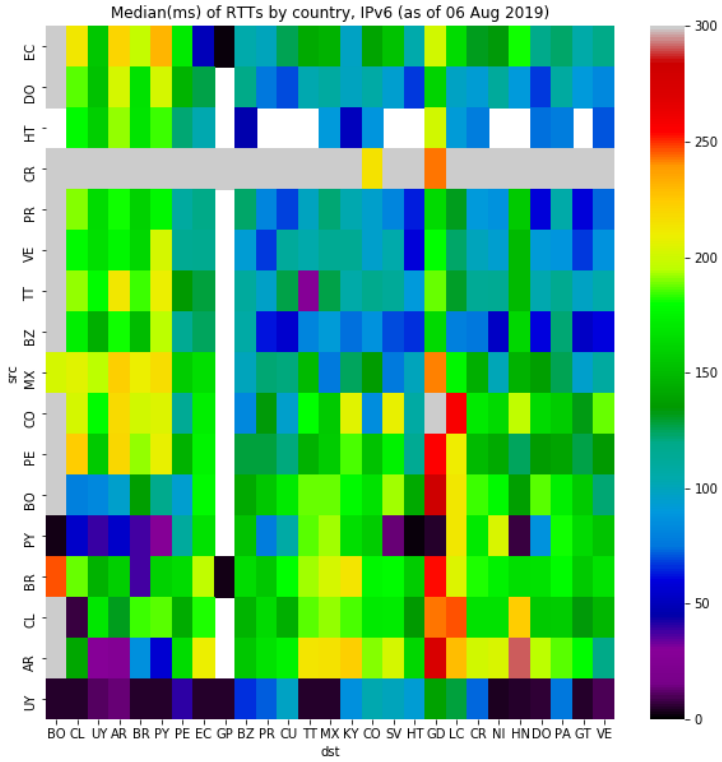

Figure 21: Heatmap of the median Round Trip Time between sources and targets in Latin American IPv6 Internet topology by country. White results had no data to analyze.

5 Conclusion

5.1 Summary

Looking back, a working monitoring system for the Internet topology in Latin America was created.
By interacting with platforms that collect data about the Internet themselves, as RIPE Atlas, CAIDA and RouteViews, the system collects data. With this data sources and targets for a measurement campaign are determined and IP addresses get mapped to AS numbers.
Based on the sources, targets and parameters, entered by the user, campaigns

are scheduled over RIPE Atlas and archived into a database. The results are processed to create various graphics. While the system provides the user with information about the network graph, degree distribution, average neighbor degree, clustering coefficient and RTTs between countries, the data can also be acquired over a download feature and used by the user for his own purposes and analysis.
Another important factor are the differences between the IPv4 and IPv6 topology, displayed by the program. It allows us to see, that there are still significant differences in the state of both architectures in Latin America, that are worth to be monitored closely.
However, the full capacities of the monitoring system could not be reached, as it has to work with consideration for the limitations set by RIPE Atlas to the API user. These restrictions increase the time the system needs to complete measurement campaigns by a great margin and force the program design to fit its characteristics. This may lead to hindrances, depending on the desired use cases, as not all options could be included in the given development time.
Taking all this into account, the system can certainly be a great asset in helping organizations to continually monitor and analyze the state and development of the Internet topology in Latin America and the differences between the IPv4 and IPv6 architecture.

5.2 Outlook

The software and libraries used to implement the monitoring system offer a great array of different libraries, packages and extensions to add to the program. On this way it could be advanced into different directions. Examples include a realization of multiple users with own logins and preferences on the same server, entering requests for the database directly over the website and combining results from multiple measurement campaigns.
The biggest challenge, when evolving the system will be the scheduling of the measurements. As the platform to conduct the measurements is the property of RIPE and the limitations and credit costs can be very restrictive, a good communication with RIPE will be crucial in further exploring the Latin American Internet topology. However, the scripts could be re-purposed to use an other or own platform with little expense.
Also, the opportunities to work with the data collected are far from exhausted. There is a lot of additional information that can still be extracted from the data sets. Examples include an analyzing and visualizing the RTTs between two ASes instead of countries, the influence of different parameters, such as the protocol and usage of Paris traceroute, on the routing and getting information about the intra-domain network of ASes by further examining the hops of the traceroutes.
As the landscape of the Internet topology in Latin America will continue to evolve and the IPv6 network will continue to expand, a lot of opportunities to use and further develop the created monitoring system will arise.

References

[1] CAIDA Archipelago (Ark) Measurement Infrastructure. `http://www.caida.org/projects/ark/`. Accessed on 23th August, 2019.

[2] CAIDA Routing Datasets. `http://data.caida.org/datasets/routing/`. Accessed on 16th July, 2019.

[3] Complex Networks and Data Communication Group. `https://cnet.fi.uba.ar/en/`. Accessed on 23th August, 2019.

[4] GridFS. `https://docs.mongodb.com/manual/core/gridfs/`. Accessed on 24th August, 2019.

[5] ipaddress. `https://docs.python.org/3/library/ipaddress.html`. Accessed on 24th August, 2019.

[6] Latin American Network Information Center: Country Directory. `http://lanic.utexas.edu/subject/countries/`. Accessed on 16th July, 2019.

[7] RIPE Atlas API. `https://atlas.ripe.net/docs/api/v2/manual/`. Accessed on 16th July, 2019.

[8] RIPE Atlas Credit System. `https://atlas.ripe.net/docs/credits/`. Accessed on 16th July, 2019.

[9] Route Views Project. `http://www.routeviews.org/routeviews/`. Accessed on 16th July, 2019.

[10] WiredTiger Storage Engine. `https://docs.mongodb.com/manual/core/wiredtiger/`. Accessed on 24th August, 2019.

[11] Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied. `https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf`, February 2011. Accessed on 16th July, 2019.

[12] CAIDA AS Rank. `http://as-rank.caida.org/`, 03 2019.

[13] Ravindra K Ahuja, Kurt Mehlhorn, James Orlin, and Robert E Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990.

[14] Ignacio Alvarez-Hamelin, Luca DallâĂŹAsta, Alain Barrat, and Alessandro Vespignani. Lanet-vi in a nutshell, 2006.

[15] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 153–158, New York, NY, USA, 2006. ACM.

[16] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring multipath routing in the internet. *IEEE/ACM Trans. Netw.*, 19(3):830–840, June 2011.

[17] Jun Bi, Jianping Wu, and Xiaoxiang Leng. Ipv4/ipv6 transition technologies and univer6 architecture. *International Journal of Computer Science and Network Security*, 7(1):232–243, 2007.

[18] Josiah L Carlson. *Redis in action*. Manning Shelter Island, 2013.

[19] Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. " O'Reilly Media, Inc.", 2013.

[20] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, August 1999.

[21] David Flanagan. *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2006.

[22] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[23] David Groth and Ben Bergersen. *Network+ Study Guide*, pages 11–12. SYBEX Inc., Alameda, CA, USA, 2nd edition, 2001.

[24] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[25] Charles L Hedrick. Routing information protocol. 1988.

[26] Adrian Holovaty and Jacob Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.

[27] Kenneth Reitz. Requests: HTTP for Humans. `https://github.com/psf/requests`. Accessed on 24th August, 2019.

[28] Diego Kiedanski, Eduardo Grampín, and J. Ignacio Alvarez-Hamelin. The atlas vision of ipv6 in latin america: Topology and latency. In *Proceedings of the 10th Latin America Networking Conference*, LANC '18, pages 40–47, New York, NY, USA, 2018. ACM.

[29] Marek Kubica. Howto use python in the web, 2010.

[30] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*, pages 383–399. Pearson, 6th edition, 2012.

[31] Pietro Marchetta, Valerio Persico, Antonio Pescapé, and Ethan Katz-Bassett. Don't trust traceroute (completely). In *Proceedings of the 2013 workshop on Student workhop*, pages 5–8. ACM, 2013.

[32] Wes McKinney. pandas: a foundational python library for data analysis and statistics.

[33] Mike Dirolf. PyMongo. `https://github.com/mongodb/mongo-python-driver`. Accessed on 24th August, 2019.

[34] John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[35] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[36] Ioan Raicu and Sherali Zeadally. Evaluating ipv4 to ipv6 transition mechanisms. In *10th International Conference on Telecommunications, 2003. ICT 2003.*, volume 2, pages 1091–1098. IEEE, 2003.

[37] Yakov Rekhter and Tony Li. A border gateway protocol 4 (bgp-4). 1995.

[38] Leonard Richardson. Beautiful soup documentation. *April*, 2007.

[39] Alex Rodriguez. Restful web services: The basics. *IBM developerWorks*, 33:18, 2008.

[40] R Siamwalla, R Sharma, and S Keshav. Discovering internet topology. *Unpublished manuscript*, 1998.

[41] Jake Spurlock. *Bootstrap: responsive web development*. " O'Reilly Media, Inc.", 2013.

[42] RN Staff. Ripe atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3), 2015.

[43] Michael Stonebraker. Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11, 2010.

[44] Thomas Johansson, James Crasta. WTForms. `https://github.com/wtforms/wtforms`. Accessed on 24th August, 2019.

[45] Aaron Watters, James C Ahlstrom, and Guido Van Rossum. *Internet programming with Python*. Henry Holt and Co., Inc., 1996.