SR-App Analytics

Department of Computer Science
OST – University of Applied Sciences
Campus Rapperswil-Jona

Autumn Term 2020

Author(s):	Michel Bongard, Dominique Illi
Advisor:	Prof. Laurent Metzger
Project Partner:	Cisco Systems Belgium
External Co-Examiner:	Laurent Billas
Internal Co-Examiner:	Prof. Frank Koch

SR-App Analytics
A – Content

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

SR-App Analytics
B – Abstract

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

Abstract

SR-App Analytics

Author(s)	Michel Bongard, Dominique Illi
Advisor	Prof. Laurent Metzger
External Co-Examiner	Laurent Billas
Topic	Software Engineering, Segment Routing
Project Partner	Cisco Systems Belgium

Initial Situation:

With the emergence of the segment routing technology and the development of the "Jalapeño" data collection framework by Cisco, there are many opportunities for application development that offer a benefit to network engineers and network operators. By using the network data provided by the framework, a variety of different features and use cases may be implemented. The field of application of those so-called SR-Apps is huge and can be anything from basic monitoring functionality to in depth analytics of load distribution and simulation of network changes.

This thesis focuses on the development of an application in the field of analytics to provide information about the general network health state and link saturation in case of topology changes.

Approach / Technology:

It was decided to build the application using a monolithic architecture with ASP.Net Core, because this is what the development team is most familiar with. The programming languages chosen are C# for the backend and JavaScript for client-side functionality, along with HTML and SCSS for markup and styling.
For the client-server communication the web socket library SignalR (C#) was chosen.

To be able to display a map of a network in the web browser a graph visualization library was required. Different such libraries and toolkits were considered and compared before the library vis.js was chosen. It offers many features, its documentation is clear and easy to use and the community seems quite active.

After having implemented a rudimentary UI prototype that was capable of displaying the topology, the focus was switched to the business logic.
In order to calculate of the link saturation in case of topology changes, the traffic between any two routers had to be redistributed on the network. This is possible thanks to the traffic matrix provided by the SR protocol. The traffic matrix contains information on how much traffic flows between any two SR routers.
Since traffic in a SR network flows along the shortest paths, the Dijkstra algorithm was implemented to calculate them. Afterwards, a custom algorithm was implemented to redistribute the traffic along those paths and with that, calculate the saturation of each link in the topology.

The final step, now that the core business logic was implemented, was to improve the UI and adding some additional features.

Result:

The application built during this thesis covers all requested features and use cases. It is scalable for topologies consisting of up to one thousand routers. The response time for a topology change (until the UI displays the updated topology with the recalculated link saturation) is less than six seconds with the hardware at hand (Quad Core Multithreaded CPU at 4GHz on the server). A sample network is shown in Figure 1.

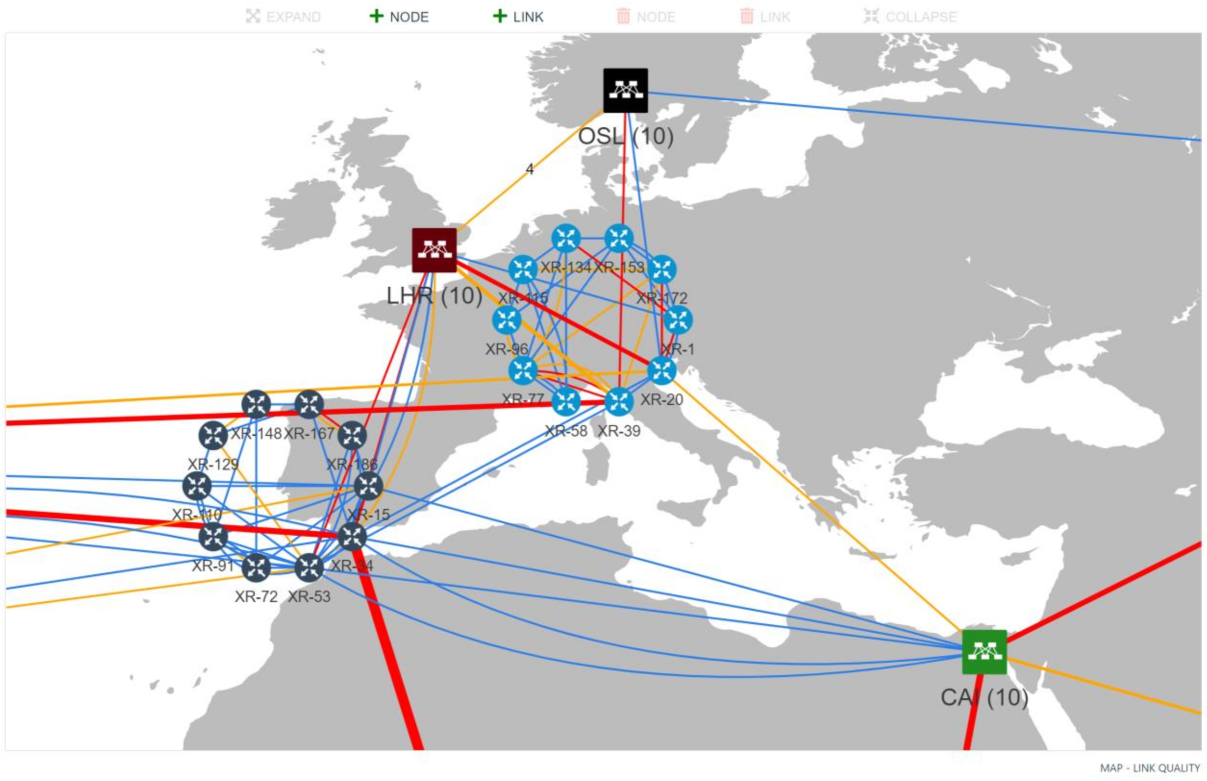


Figure 1: Sample network
Source: own creation

SR-App Analytics
C – Management Summary

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

Management Summary

SR-App Analytics

Author(s)	Michel Bongard, Dominique Illi
Advisor	Prof. Laurent Metzger
External Co-Examiner	Laurent Billas
Topic	Software Engineering, Segment Routing
Project Partner	Cisco Systems Belgium

Initial Situation:

Network infrastructures of service providers and big companies consist of thousands of devices which are strongly interconnected. This results in highly complex network designs with the need of knowing exactly how the health state of the network looks like. In this thesis a software was developed that shows crucial information about those connections through a visual web interface and that allows the network operators to simulate the consequences of device and connection failures.

Approach / Technology:

The software consists of three components, the frontend, the backend and the database. The backend and the database can run on private servers or in the cloud, while the frontend runs in the browser of the user.

The user can interact with the application using a web browser. He or she can view the devices and connections to get detailed information about their health state. The user can also simulate certain events through the browser, such as the loss of a connection or the adding of a new one. The quality of connections is displayed using different colors allowing an operator to quickly gain an overview of the health state of the connections.

To be able to react to changes in the network (such as a lost connection) and accurately predict the behavior of the changed network, certain algorithms were implemented. The results are shown in the user's browser. The calculations are all made on the server which provides much more resources to handle multiple requests from different users.

Result:

The application built during this thesis covers all requested features and use cases. It is scalable for networks consisting of up to one thousand routers. A user of the application can see all devices and connections such as their current health state and simulate changes in the network (see Figure 1).

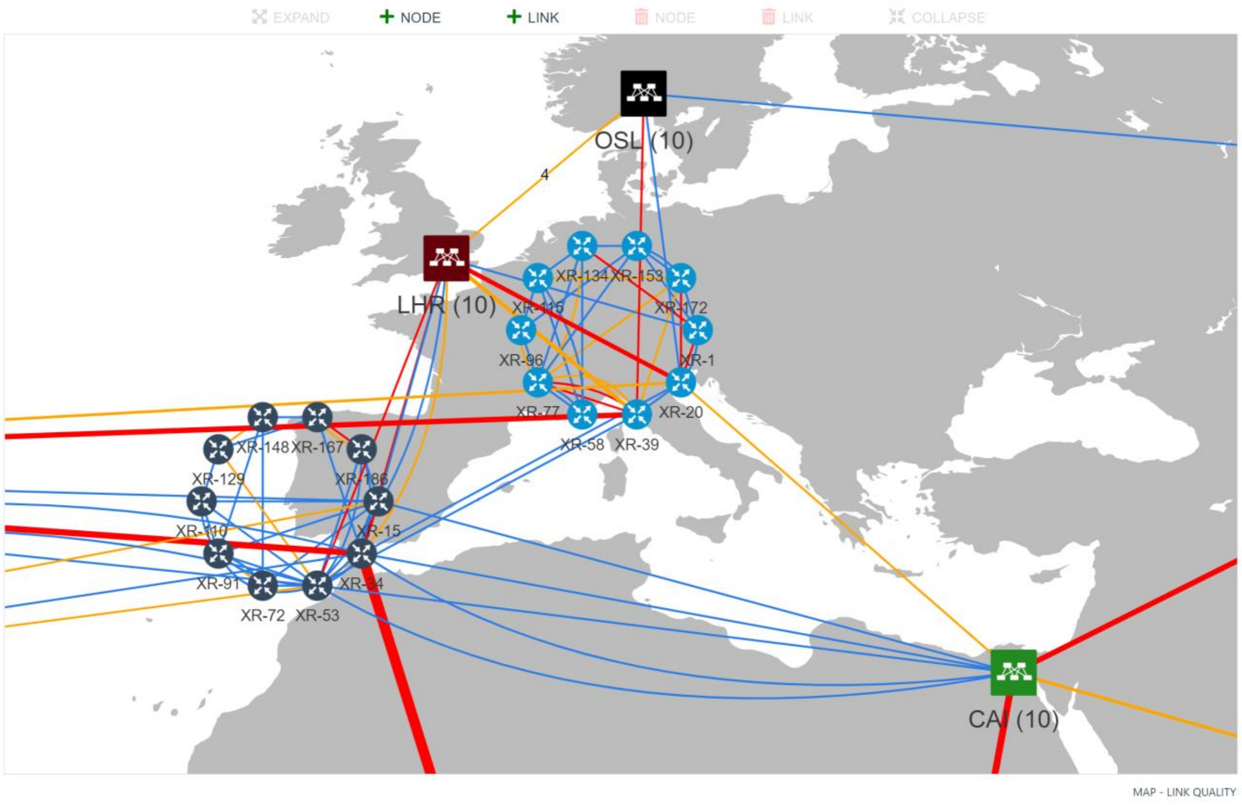


Figure 1: Sample network
Source: own creation

SR-App Analytics
D – Technical Report

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

1 Content

2 Introduction

2.1 Overview

This document presents the outcome of the thesis SR-App Analytics.
Chapters 4 to 9 discuss the key findings, such as the handling of the network visualization and approaches to the implementation of algorithms and their time and space complexity.
Chapters 10 and 11 show the results of this thesis.

2.2 Starting position and motivation

With the emergence of the segment routing technology and the development of the "Jalapeño" data collection framework by Cisco, there are many opportunities for application development that offer a benefit to network engineers and network operators. By using the network data provided by the framework, a variety of different features and use cases may be implemented. The field of application of those so-called SR-Apps is huge and can be anything from basic monitoring functionality to in depth analytics of load distribution and simulation of network changes.

Segment-routing is a new way of routing packets. Unlike other routing protocols segment-routing performs source-based routing. This simplifies traffic engineering and management across network domains. The destination of a packet is known at the ingress router and stored in the packet header.

"Jalapeño" is an application collecting data from a SR topology. The data collected reaches from link metrics like bandwidth, delay and packet loss to specific SR metrics displaying the amount of traffic sent between different SR enabled routers. All the data is made accessible in a graph database (ArangoDB) and queried by SR-Apps.

2.3 Problem Definition

During this thesis, a software application is to be developed providing monitoring and analytics functionality for a SR network. The application is to be able to show the health state of links and simulate effects of network changes in the topology. The health state of links shall be indicated by some sort of grading logic dividing the links in “Good”, “Moderate” and “Critical” links.

Further, the usage of links must be shown by counting the number of paths a link is used in. To simulate network traffic, an algorithm must be designed which can calculate the bandwidth utilization of each link upon network modifications such as the adding or removing of nodes and links. The results must be visualized allowing a quick identification of problem spots in the network.

Meeting these requirements presents the following main tasks and problems.

1. Find a suitable framework to visualize a network topology that can provide good performance and a high degree of customization.
2. Group nodes and links belonging to the same region to reduce the flood of information for the user.
3. Plan a software architecture which can handle thousands of nodes and is easily expandable and scalable.
4. Decide on a client-server communication model that is able to handle high traffic loads and that allows the pushing of data from the server to the client.
5. Define some sort of logic and metrics to grade links.

6. Implement an algorithm that can calculate the shortest paths between nodes providing high performance.
7. Implement an algorithm that can calculate the link saturation upon network changes.

The solutions to those problems and problems evolved during the project and are discussed and described in detail in Chapters 4 to 9.

3 Client-server communication

The SR-App is to be configured to poll new data from the database at certain intervals. To be able to inform the connected clients that new data is available, the server must be able to send messages to clients.
A simple REST API is therefore not sufficient for this application. To cover all use cases bidirectional communication is required, which is possible using web sockets.

3.1 REST vs WebSocket

REST is used for calls to an API with the client being the active part. The client initiates a call to the API (through a HTTP request) and waits for a response from the server.

Web sockets are used for bidirectional communication. Both the server and the client can invoke functions on their counterpart. This allows the server to push messages to a client.

3.2 ASP.NET Core SignalR

For this application, the WebSocket library SignalR is chosen, which provides the required functionality.

> *"ASP.NET Core SignalR is an open-source library that simplifies adding real-time web functionality to apps. Real-time web functionality enables server-side code to push content to clients instantly."*

- Cited from Microsoft Documentation [1]

4 Network Visualization Toolkit

To be able to display a map of a network in the web browser a graph visualization library is required. Elise Devaux compiled a large list of such libraries in a post on Medium [2]. From a cursory glance at each of these libraries, vis.js [3] seems to be the most promising one, due to its open-source nature and active community, as well as its comprehensive set of features.

Not listed in Elise Devaux' post is the toolkit called "NeXt UI" [4], probably because it does not appear to still be actively maintained. But because Cisco is listed as one of the companies using the NeXt UI framework as their topology visualization tool [5] (Cisco actually developed NeXt UI itself [6]) and because Cisco is a major Stakeholder in this project, this toolkit is also taken into consideration.

This chapter compares the two libraries vis.js and NeXt UI in more detail.

4.1 NeXt UI

Setting up a working example of NeXt UI is easy, thanks to the tutorials from Alexei Zverev [7], who used to work on the team that developed NeXt UI. Figure 1 shows what the graph looks like when some sample data is added.

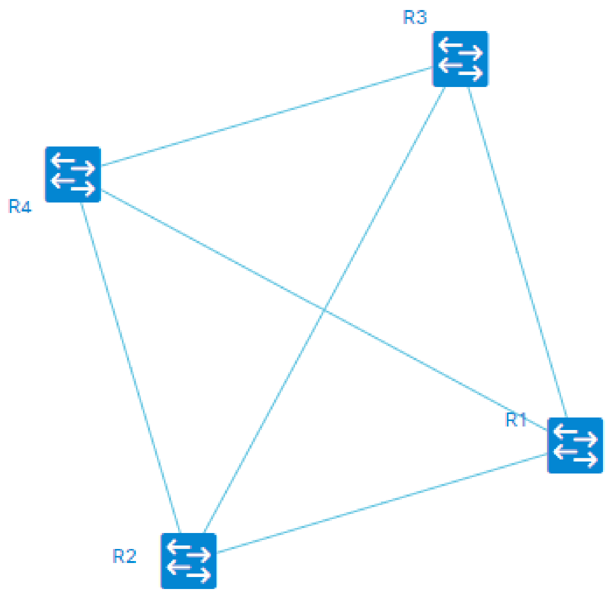
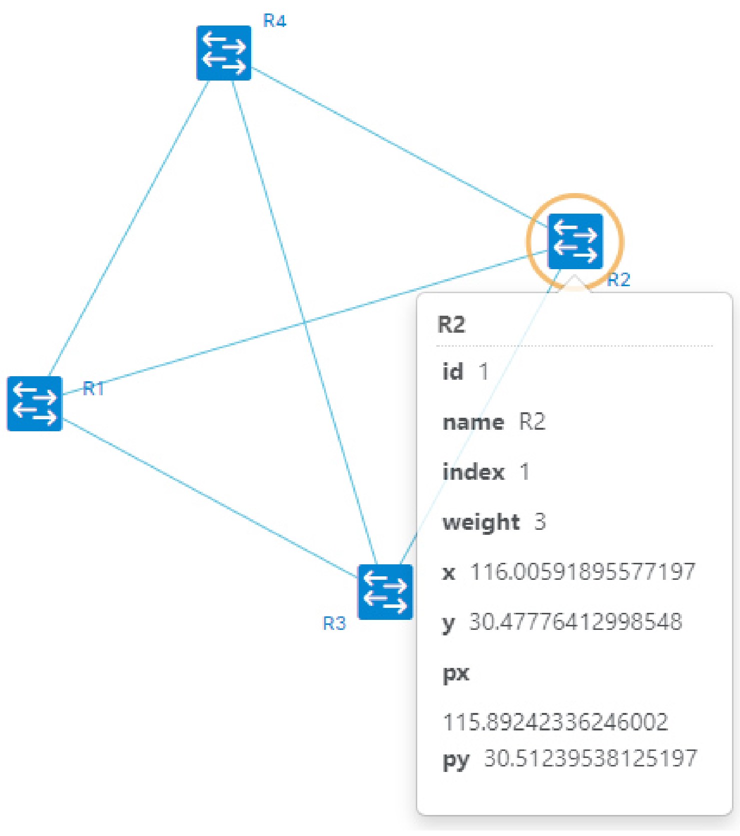


Figure 1: Prototype for graph visualization with NeXt UI
Source: own creation

The graph looks very good with default configurations and the popup windows to show detailed information on nodes and links (displayed in Figure 2) are very useful and also preconfigured.
The library offers useful features such as layering, path drawing and grouping (see the tutorials page [7]).



R2

id 1

name R2

index 1

weight 3

x 116.00591895577197

y 30.47776412998548

px

115.89242336246002

py 30.51239538125197

Figure 2: Popup window in NeXt UI
Source: own creation

Unfortunately, according to Alexei Zverev the NeXt UI is no longer under active development since 25.11.2019. Also, at a first glance, the NeXt UI community does not appear to be very active.

4.2 Vis.js

While NeXt UI offers functionality for only network graphs, vis.js supports network graphs, charts, timelines, 3D-graphs and more. Its community seems to be very active, the documentation [8] is clear and easy to understand and the library is still actively maintained (almost daily push to GitHub repository for vis-network [9] alone).

Setting up a working example for vis.js is easy by using the example on the GitHub page [9]. Figure 3 shows what the graph looks like, when populated with the same data as was the prototype for NeXt UI in Chapter 4.1.

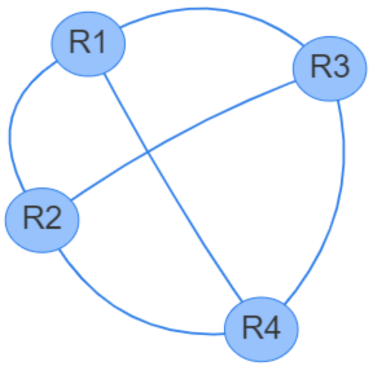


Figure 3: Prototype for graph visualization with vis.js
Source: own creation

It does not look quite as good as the default implementation of NeXt UI and there is no handy popup window to show detailed information on nodes or links, but this can be configured and implemented manually.

4.3 Comparison – NeXt UI vs. Vis.js

Table 1 summarizes the differences between NeXt UI and vis.js.

	NeXt UI	**vis.js**
Can display network graphs	Yes	Yes
Open source	Yes	Yes
Actively maintained	No	Yes
Documentation	Detailed [10]	Detailed and easy to use [8]
Community	Not very active	Quite active

Table 1: Comparison: NeXt UI vs. vis.js

The two libraries are actually very similar in how they are used and what they can do, though vis.js supports a larger variety of graphs. However, mainly because NeXt UI is no longer actively maintained, vis.js will be used for graph visualization for this project.

4.4 Testing UI scalability of vis.js

To test how well vis.js handles very large maps, different sets of random sample data is generated.

Vis.js has support for physics implemented [11], which handles the stabilization of the map, meaning it moves the nodes and edges around to show them more clearly. This calculation is done on the client side and can take long times for large maps. Vis.js allows physics to be disabled.

Each set of sample data is therefore tested with and without physics enabled. Each time measurement is repeated three times and then averaged. The results are shown in Table 2.

Test	Parameters	File size	Load time with Physics enabled	Load time with Physics disabled	Comments
#1	100 Nodes 150 Links	9KB	1,4 sec	0,8 sec	
#2	200 Nodes 300 Links	18KB	5,2 sec	0,8 sec	
#3	500 Nodes 750 Links	42KB	22,0 sec	1,0 sec	
#4	1000 Nodes 1500 Links	84KB	33,8 sec	1,5 sec	For test with larger data sets, physics is no longer enabled, since it would take impractically long.
#5	10'000 Nodes 15'000 Links	0,9MB	-	6,0 sec	Interactions (zooming in & out and on click behavior) are very responsive. Barley any delay is noticeable.
#6	100'000 Nodes 150'000 Links	9,5MB	-	83,0 sec	Interactions are very sluggish; it takes about 2 seconds for the UI to respond to each interaction.

Table 2: Testing of vis.js scaling

These tests show that, for an acceptable user experience the map should not exceed the parameters of test #2 with physics enabled or the parameters of test #5 with physics disabled. This meets the requirements of this project, since the number of nodes handled by SR-Apps does not exceed 10'000 nodes.

The longer load times for maps with 100'000 nodes and upwards can technically be rendered by vis.js and may still be acceptable for certain use cases but should be avoided, since each interaction becomes increasingly sluggish.

Such large maps are of limited use anyway, because there really cannot be any information gathered from them. While for test #5 the individual nodes can just be seen (Figure 4), test #6 just shows a blue circle (Figure 5), requiring the user to zoom in a lot before any nodes become visible.

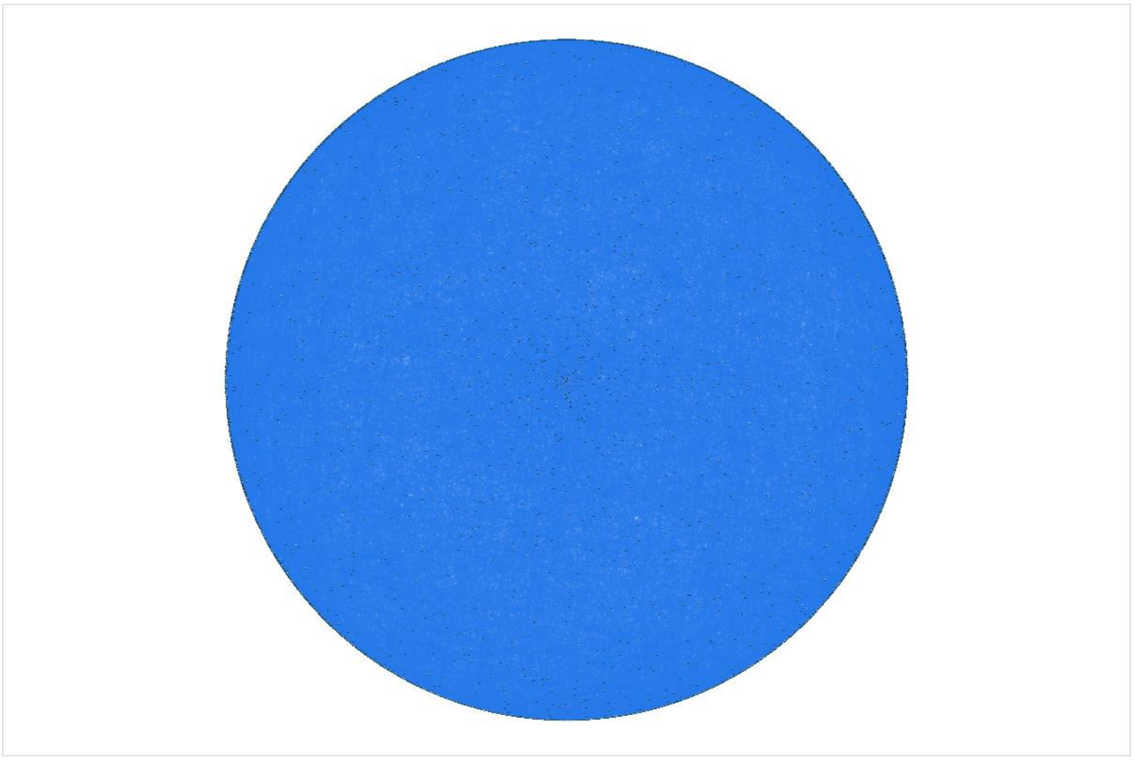


Figure 4: Test #5
Source: own creation

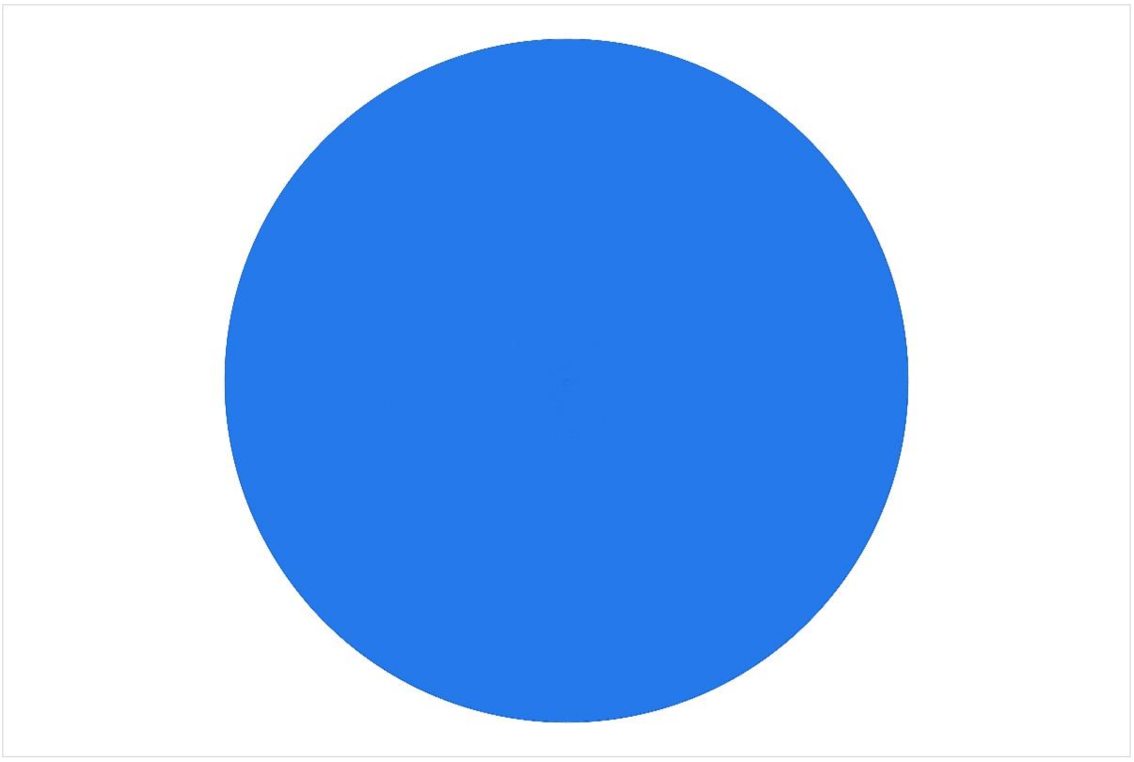


Figure 5: Test #6
Source: own creation

5 Issues with Vis.js

During the course of this project, some issue with vis.js came up. The more noticeable ones are described in this chapter.

5.1 Updating the Map

Vis.js offers methods that allow making changes to the map, such as updating nodes and edges or moving nodes to a new position. While these are great for making small changes, it was discovered that they run quite slow when making multiple changes at once, especially for large topologies. These methods seem to have an O(N) runtime.

Because of that, in most cases it is orders of magnitude faster to simply redraw the entire map instead of making incremental changes. For this reason, the entire map is redrawn to display changes.

5.2 Multiple edges between two nodes

By default, vis.js shows multiple edges between two nodes only when physics is enabled. The resulting graph is shown in Figure 6.

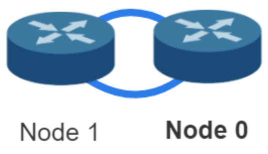


Figure 6: Two Edges, Physics enabled
Source: own creation

When physics are disabled however, vis.js overlaps the two edges, resulting in the graph in Figure 7.

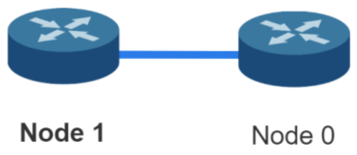
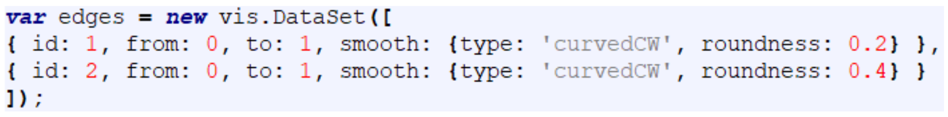


Figure 7: Two Edges, Physics disabled
Source: own creation

Unfortunately, enabling physics is not an option for this project, since this feature becomes unusable with networks that contain more than just a few hundred nodes (this is discussed in more detail in Chapter 4.4).

In 2017, GitHub user wimrijnders has proposed a workaround for this problem **Es ist eine ungültige Quelle angegeben.**. His suggestion is to use the roundness option on edges as shown in Figure 8 and to use a different roundness for each edge, which is a way to manually curve individual edges.



```
var edges = new vis.DataSet([
{ id: 1, from: 0, to: 1, smooth: {type: 'curvedCW', roundness: 0.2} },
{ id: 2, from: 0, to: 1, smooth: {type: 'curvedCW', roundness: 0.4} }
]);
```

Figure 8: Solution suggested by wimrijnders
Source: own creation

It is not a very neat solution, but it is the only way to show multiple edges between two nodes while leaving physics disabled.

6 Clustering

The networks that will ultimately be handled by SR-Apps can grow quite large with up to a thousand nodes. Displaying so many nodes in a single network map can be very overwhelming and confusing. Some form of mechanism must exist to filter irrelevant data and to show only information that is of interest to the user.

One way to do that is by clustering. Combining multiple nodes to larger clusters and omitting unimportant links, the complexity of the network map can be greatly reduced while retaining accuracy.

This chapter describes the multilevel clustering that was initially implemented for SR-Apps. The clustering has been simplified for the current version of SR-Apps, allowing only one level of clusters (regions) because it seemed to be sufficient for topologies up to one thousand nodes. Multilevel clustering may need to be revisited though, if the application needs to be able to handle even larger topologies.

Vis.js offers clustering natively, as can be gathered from their examples for vis-network [12] (see Figure 9).

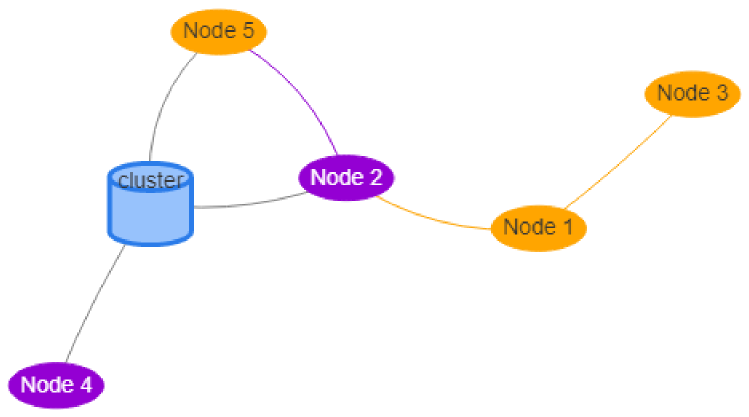


Figure 9: Vis.js clustering example
Source: visjs.github.io [13]

Unfortunately, the options for how the clusters are made are somewhat limited and not applicable to the requirements of this project. For the SR-Apps, the links are much more important than the nodes and vis.js clustering works by simply combining certain nodes (the links are secondary and simply connect the clusters).

For that reason, the clustering has to be implemented from scratch, to give more control over how nodes are combined, and which links remain visible.

6.1 Requirements of clustering

First the requirements of the clustering need to be defined. Table 3 describes the characteristics of the resulting clustering.

Requirement	Description
R1	There must be multiple levels of clusters, meaning the network map can for example be: Level 0: **Not clustered**, to show all nodes & links Level 1: **Clustered**, to show only important and very important nodes & links Level 2: **Strongly clustered**, to show only very important nodes & links
R2	Which nodes & links are considered important or very important is entirely based on the metrics of the links, because it is the links that are of importance to the user. How the importance of a link is calculated depends on the use case.
R3	Each cluster-node is aware of which nodes are in its cluster and can display detailed information on each of them if requested.
R4	The user can set the cluster level of the entire network map through a slider or via buttons or something similar.
R5	By clicking on a cluster-node the user can reduce the cluster level of the nodes in the selected cluster, essentially zooming into that cluster. All other clusters remain at their original cluster level.

Table 3: Clustering requirements

6.2 Approach

For orientation purposes Figure 10 shows the architecture of a possible implementation of the clustering described in this chapter and Figure 11 the accompanying sequence diagram.

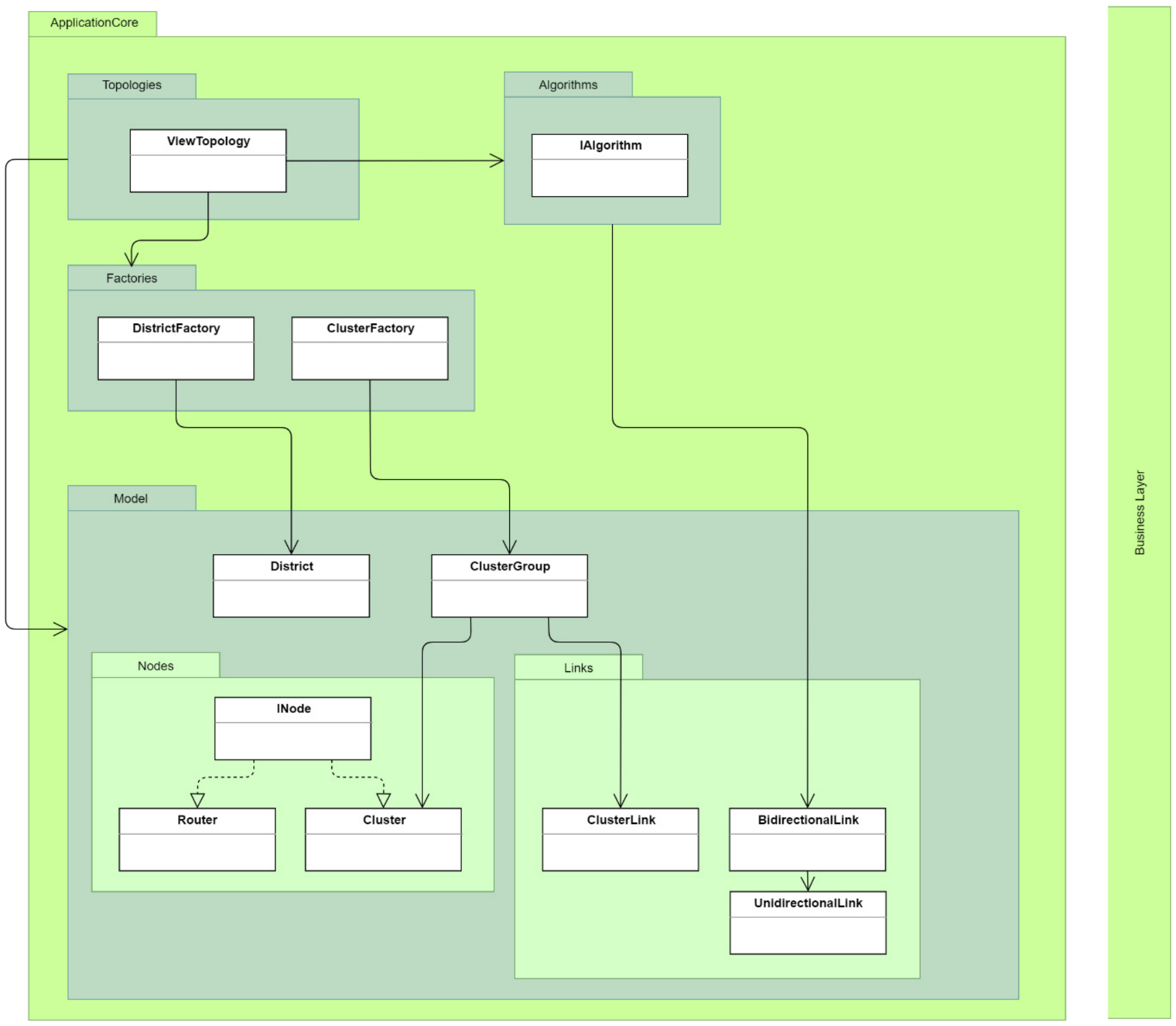


Figure 10: Simplified package diagram - clustering
Source: own creation

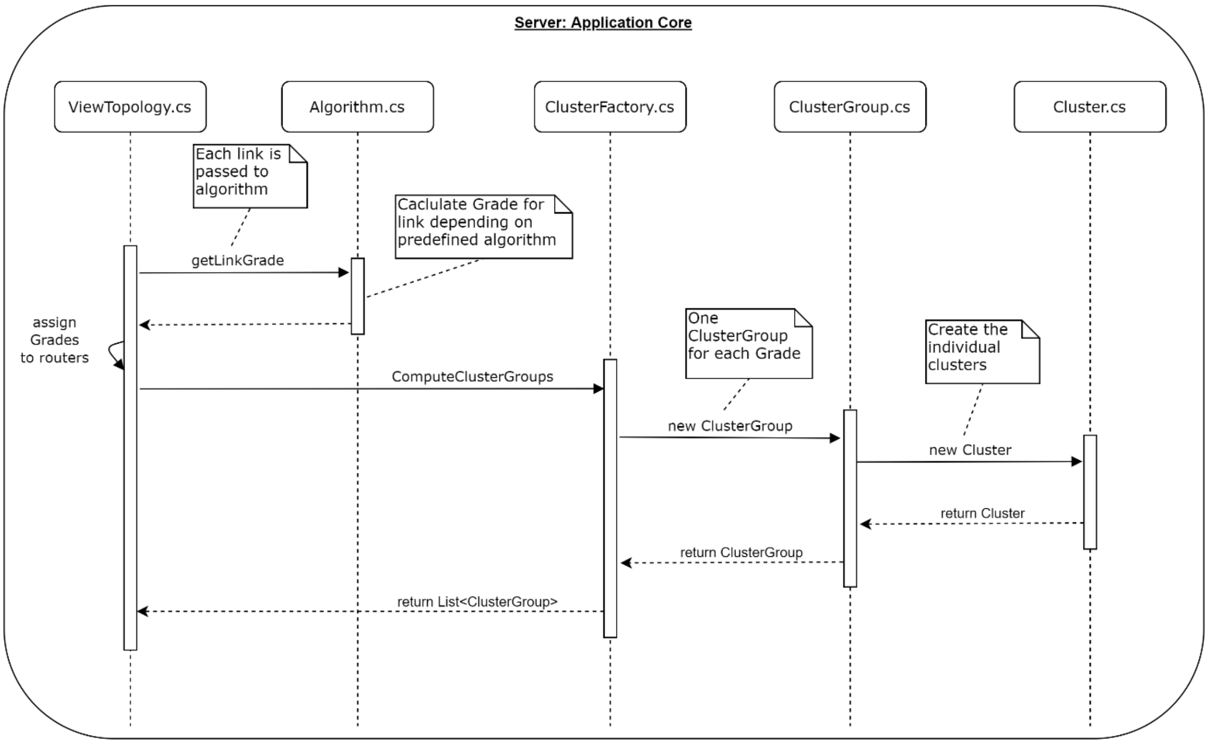


Figure 11: Sequence diagram - clustering
Source: own creation

6.2.1 Step 1 – Assign grades to links

When a new ViewTopology is created, it is passed an algorithm that determines how the grades of the links are calculated. A grade is a measure of how important the specific link is. The ViewTopology first passes each link to the algorithm which calculates grade based on predefined logic.

Example:

- **Grade 0**, if bandwidth utilization is below 50%.
- **Grade 1**, if bandwidth utilization is between 50% and 80%.
- **Grade 2**, if bandwidth utilization is between 80% and 95%.
- **Grade 3**, if bandwidth utilization is above 100%.

6.2.2 Step 2 – Assign grades to routers

Each router is assigned a grade based on its links. A router receives the highest grade of all its connected links.

Example:
As an example, consider Figure *12*. Node 3 has three links, where:

- Link A is grade 1
- Link B is grade 3
- Link C is grade 1
  ⇨ Therefore, Node 3 is considered to be of grade 3.

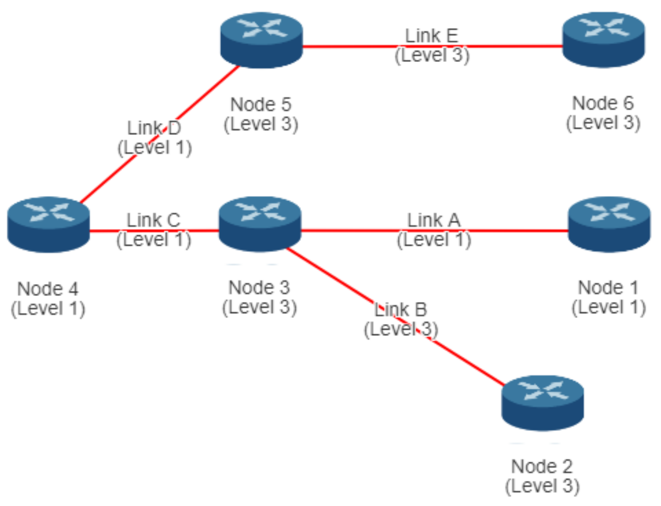


Figure 12: Example of router grade assignment
Source: own creation

6.2.3 Step 3 – Cluster Factory

Now that all grades have been calculated, the ViewTopology passes all its routers and links to the ClusterFactory.

The ClusterFactory creates several ClusterGroups, one per grade (aka zoom level). Each ClusterGroup takes the entire ViewTopology and creates all Clusters and ClusterLinks (links between two clusters) for their grade.

The concept of clustering is best explained on an example.

<u>Example of ClusterGroups:</u>
To clarify the concept, consider the example in Figure 13. The user can adjust the grade of the map through the slider. Figure 13 shows the map at grade 0, meaning it is not clustered. The entire topology is shown.

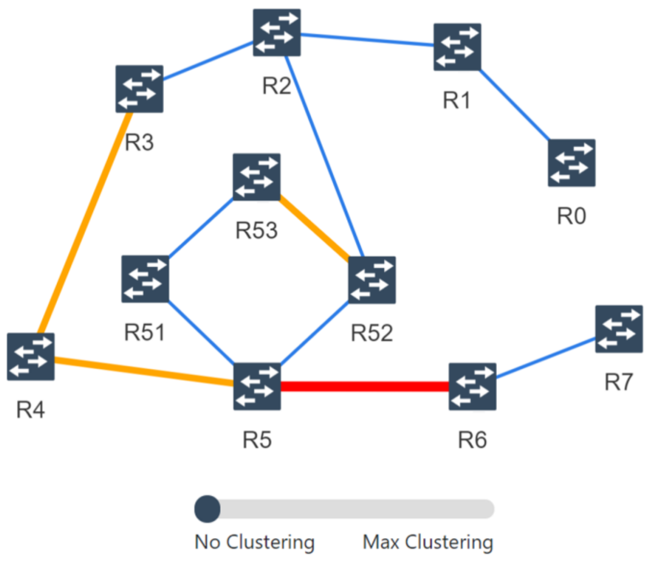


Figure 13: Sample Topology – Grade 0
Source: own creation

Because the grade directly references the metric of the link quality, it can be color coded accordingly. In this example, the links are color coded as follows:

- **Grade 0**: blue
- **Grade 1**: orange
- **Grade 2**: red
- **Grade 3**: **black** (no link of grade 3 exists in this topology)

If the user chooses to view the map at grade 1, as shown in Figure 14, ClusterGroup 1 is shown.

- ClusterGroup 1 shows only nodes and links that have a grade equal to or higher than 1.
- All routers with a grade lower than 1 were aggregated to its closest (number of hops) neighbor.
- Multiple links between two clusters are aggregated to ClusterLinks.
- A ClusterLink receives the grade (and therefor the color) of its link with the highest grade.

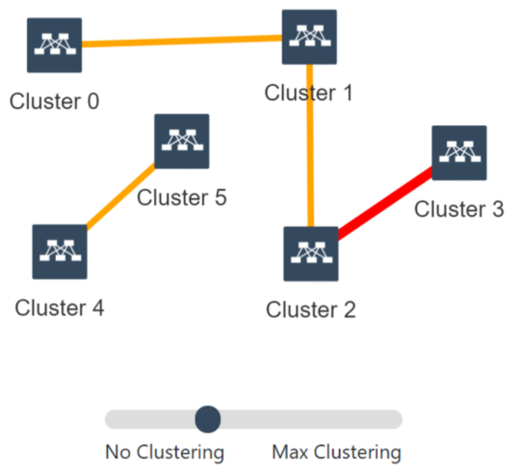


Figure 14: Sample Topology – Grade 1
Source: own creation

If the user chooses to view the map at grade 2, ClusterGroup 2 would be used to show only nodes and links with grade 2 or higher (see Figure 15).

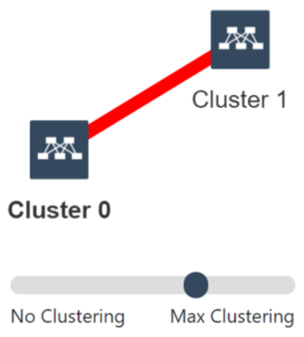


Figure 15: Sample Topology – Grade 2
Source: own creation

6.3 Disjointed networks

At this point clustering is fully implemented. However, there is a small issue, which can be seen in Figure 14 in Chapter 6.2.3. In most cases, the networks will appear disjointed, because they are connected only by links of lower grades.

To remedy that, links of lower grades are displayed in the map as dashed lines. For Figure 14 it would result in the new map shown in Figure 16.

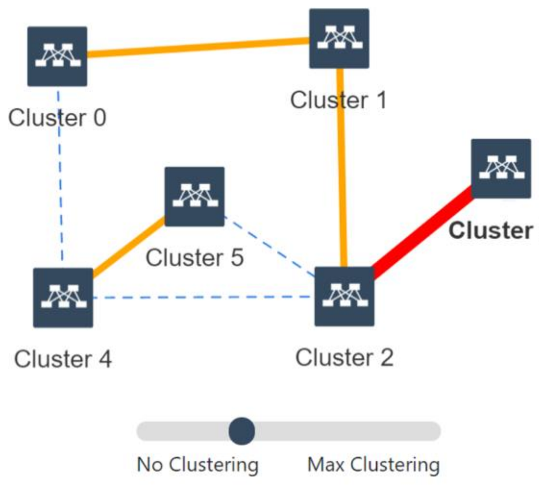


Figure 16: Sample Topology – Grade 1 - as a single network
Source: own creation

7 Regions

Besides clustering, SR-Apps offers another way to minimize the overwhelming amount of information of a full topology: Regions.
A user can choose to view only a specific region or even multiple regions and all their interregional links. Unfortunately, vis.js does not offer support for grouping nodes by region on a map. This means, that a map containing all nodes of the two regions “Zürich” and “Genf” (from a mocked and completely randomized topology) would look like a complete mess (see Figure 17). Vis.js simply places the nodes as best it can, without concern for region grouping.

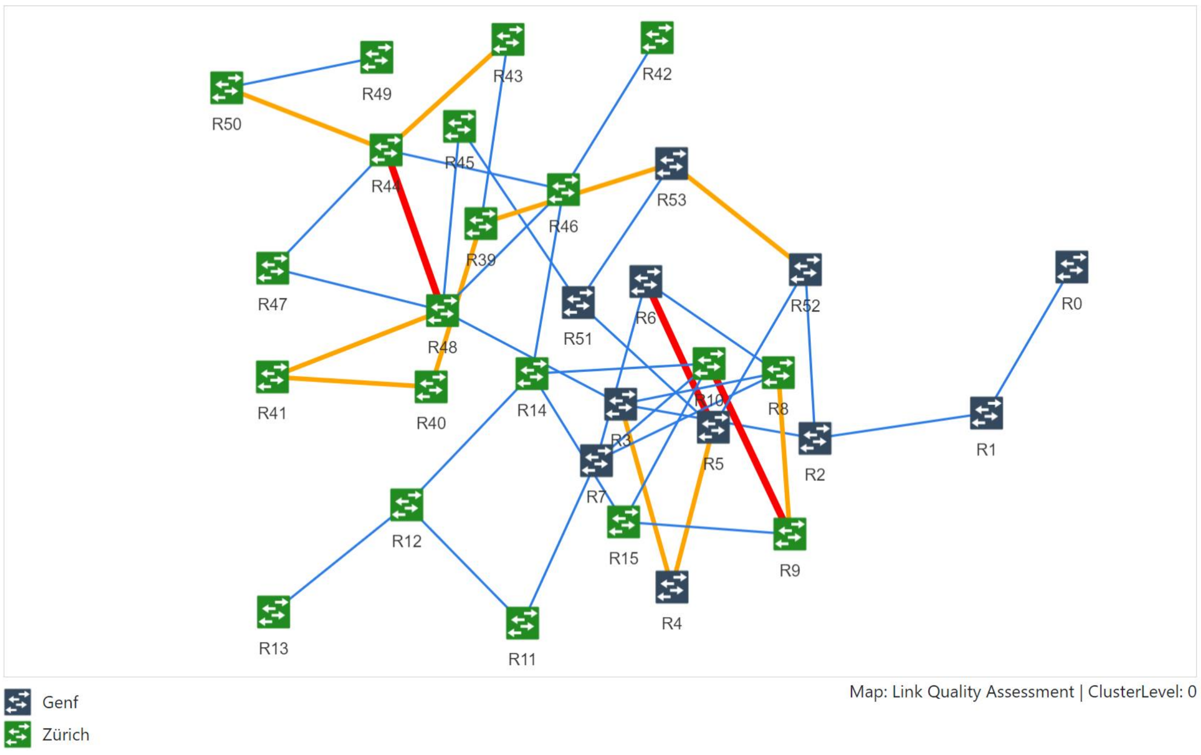


Figure 17: Topology Genf - Zürich (ungrouped)
Source: own creation

Ideally, all nodes of one region would be on the left and all nodes of the other region would be on the right. Such a feature has been discussed on GitHub before [14], and user Muntaner has even implemented a solution for this and was kind enough to make it publicly available on GitHub [15]. Muntaners solution, implemented in JavaScript, looks like the map displayed in Figure 18.

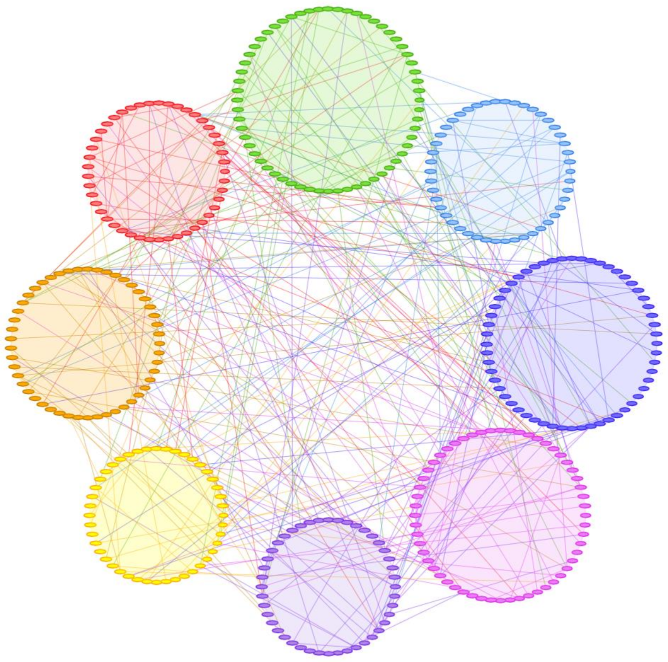


Figure 18: Muntaners implementation of districts
Source: screenshot of Muntaners sample application *[15]*

Muntaners algorithm works by manually calculating the exact coordinates of each node, which can then be passed to vis.js as shown in Figure 19.

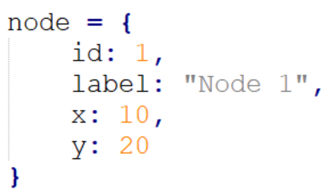

```
node = {
    id: 1,
    label: "Node 1",
    x: 10,
    y: 20
}
```

Figure 19: Passing coordinates to vis.js
Source: own creation

The algorithm first spreads out the regions evenly on a circle around the center of the map. Then for each district it spreads out all nodes from this district evenly on a circle around its center.

Arranging nodes in a circle is not always the best option, but it is a simple solution that is quick to implement.
For a more evenly distributed calculation of the coordinates a closer look would have to be taken at more complex algorithms. Vis.js for example uses the KamadaKawai algorithm to pre-calculate the coordinates, as is evident from their implementation [16], which is a force-directed graph drawing algorithm [17]. Such an algorithm would have to be implemented in C# and modified so regions may be grouped together.

For this project however, the simpler approach of arranging the regions in circles is chosen. After implementing Muntaners algorithm in C#, the resulting graph (Figure 20) looks already a lot better than the original one from Figure 17. The C# implementation is a modified and improved version of

Muntaners algorithm. It dynamically matches the spacings between nodes and regions more evenly for any size topologies, whereas before it used hardcoded values that would either result in regions overlapping (for large topologies) or regions being too far apart (for small topologies).

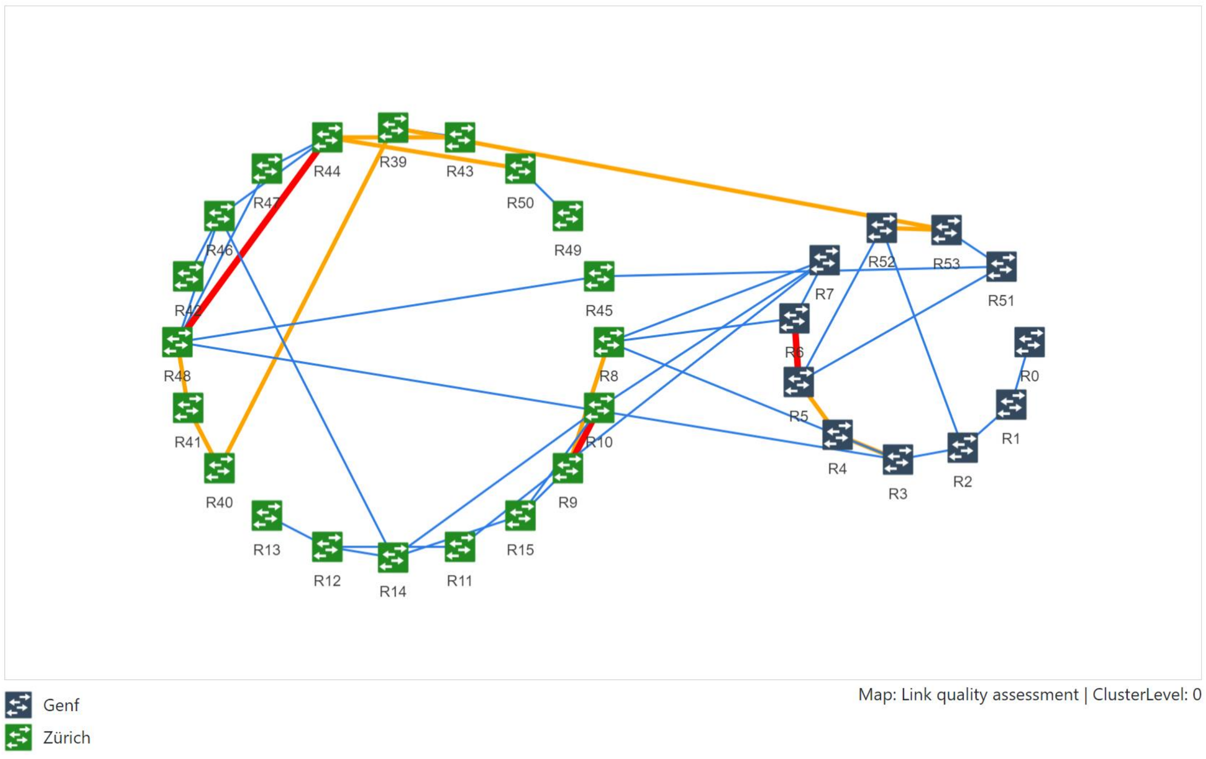


Figure 20: Topology Genf - Zürich (grouped by regions)
Source: own creation

8 Calculation of Shortest Paths

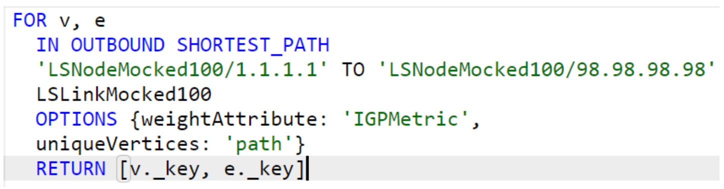
To get the number of shortest paths going through a specific link, all shortest paths of the entire network need to be calculated first.
ArangoDB [18] is specifically designed for graphs and provides its own query language, allowing the user to directly query for shortest paths between two nodes. Since ArangoDB is optimized for graph calculations, this seems to be the easiest and most promising approach.

8.1 ArangoDB

ArangoDB provides two methods to receive shortest paths. One is called **shortest-path** [19] and returns exactly one shortest path between two nodes. The other one is called **k-shortest-paths** [20] and returns all paths between two nodes, sorted in ascending order by their weight.

8.1.1 Shortest-Path

This method returns exactly one shortest path between two given nodes [19]. Figure 21 shows an example of a query for **shortest-path** from node '1.1.1.1' to node '98.98.98.98'.



```
FOR v, e
  IN OUTBOUND SHORTEST_PATH
  'LSNodeMocked100/1.1.1.1' TO 'LSNodeMocked100/98.98.98.98'
  LSLinkMocked100
  OPTIONS {weightAttribute: 'IGPMetric',
  uniqueVertices: 'path'}
  RETURN [v._key, e._key]
```

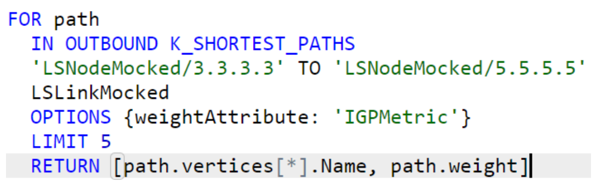
Figure 21: Example query shortest-path
Source: own creation

By default, the shortest path is found by using a depth-first search algorithm. Optionally, a breadth-first search algorithm can be used instead. The website clicage.com provides an excellent PDF on ArangoDB and its shortest path calculation which explains this in more detail [21]. Unfortunately, no information regarding the time complexity about the algorithms was found. However, regardless of how well it performs, this method is not an option for this project, because for the LQA use case the software needs to be able to handle multiple equal cost paths.

8.1.2 K-shortest-paths

The second option to calculate shortest paths is by using the **k-shortest-paths** method [20]. This method calculates all possible paths between two nodes. These paths are then returned in ascending order of their weight. It is highly recommended to use the **LIMIT** option, to limit the number of shortest paths returned, because otherwise it performs very poorly. If the option is set to **LIMIT 5** for example, the method will return only the five shortest paths between the two nodes.

Figure 22 shows an example of a query for k-shortest-paths from node '3.3.3.3' to node '5.5.5.5' with the limit set to 5.



```
FOR path
  IN OUTBOUND K_SHORTEST_PATHS
  'LSNodeMocked/3.3.3.3' TO 'LSNodeMocked/5.5.5.5'
  LSLinkMocked
  OPTIONS {weightAttribute: 'IGPMetric'}
  LIMIT 5
  RETURN [path.vertices[*].Name, path.weight]
```

Figure 22: Example query k-shortest-paths
Source: own creation

8.1.3 Time Complexity of K-Shortest-Paths

According to the GitHub page from ArangoDB [22] the k-shortest-paths method uses a slightly modified version of the Yen's k-shortest-paths algorithm. The Yen's algorithm has the following time complexity [23]:

$$O(kN * (M + N * \log(N)) \qquad (1)$$

- k = number of paths to return
- N = number of nodes
- M = number of edges

Since the number of equal cost shortest paths should generally be a rather small number in the scope of this project, k can be assumed to be constant.
Also, because in a realistic network the average number of edges per node does not depend on the size of the network, M can be expressed as **c * N**, where **c** is a constant. Therefore, the time complexity of k-shortest-paths can be expressed as follows:

$$O(N^2 * \log(N)) \qquad (2)$$

As a reminder, this is the time complexity for the calculation of all shortest paths from only one node to one other. Because this project requires knowing all shortest paths from each node to each node, the k-shortest-paths algorithm would have to be called N^2 times, resulting in the following final time complexity:

$$O(N^4 * \log(N)) \qquad (3)$$

8.1.4 Performance Analysis

To see how the k-shortest-paths algorithm performs and how well it scales, a few test queries were run, now shown in Table 4. As expected, with a time complexity of $O(N^4 * \log(N))$, k-shortest-paths does not scale well at all.

Number of nodes	Number of edges	K	Duration for a single execution of k-shortest-paths	Estimated duration for the entire network (N^2)
100	300	10	15ms	10'000 * 15ms = 2.5min
1’000	3000	10	80ms	1'000'000 * 80ms = 22.2h

Table 4: Performance of k-shortest-paths algorithm

Even though the server running ArangoDB only has two single threaded CPU cores running at 2.2GHz each and has just 2GB of RAM, these results are very poor. Even with better hardware, anything upwards of 1’000 Nodes would take too long to calculate.

8.1.5 Why does k-shortest-paths perform so poorly?

The reason why Yen’s k-shortest-paths algorithm performs so poorly for this use case is because it is designed to find all possible paths and to do so in ascending order.

The first thing the Yen’s algorithm does is calculate Dijkstra for the start node. Dijkstra already provides the necessary information to determine all shortest paths to all other nodes for this start node. Yen’s algorithm however ignores most of that and only stores one shortest path to a single target node.

Because SR-Apps requires to do this N^2-times, it results in a lot of redundant calculation.

8.2 C# Implementation

To remedy the issues discussed in Chapter 8.1.5 a custom implementation of the shortest path calculation is necessary. Using the fact that Dijkstra already provides the necessary information for all equal cost shortest paths, the performance can be greatly increased.

8.2.1 Time Complexity

When implemented with a priority queue based on a Fibonacci heap, the Dijkstra algorithm has the following time complexity:

$$O(M + N * \log(N)) \tag{4}$$

As discussed in Chapter 8.1.2, M can be expressed as **c * N**, where c is a constant. Therefore, the time complexity can be expressed as follows:

$$O(N * \log(N)) \tag{5}$$

The Dijkstra algorithm must be calculated N times to get all shortest paths of the entire network, so the overall time complexity would be the following:

$$O(N^2 * \log(N)) \tag{6}$$

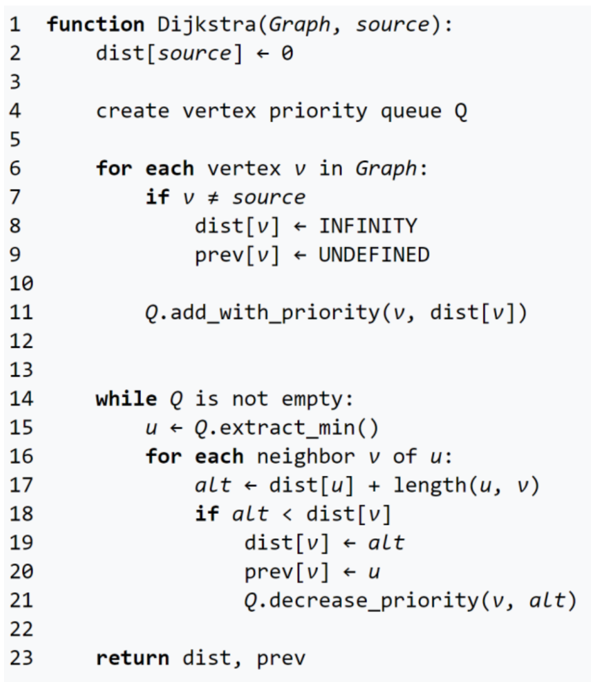
This will still not scale very well, but it is orders of magnitude better than the $O(N^4 * \log(N))$ that the solution with ArangoDB would result in.

Unfortunately, the solution with $O(N^4 * \log(N))$ cannot be improved upon any further, for two simple reasons.

1. The N^2 is unavoidable, because to know N paths for N nodes, N^2 steps are required just to store all the paths, regardless of how efficient the calculation is.
2. The log(N) is also unavoidable. It comes from adding and removing from the heap-based priority queue, which is the most efficient data structure for the Dijkstra algorithm.

8.2.2 Approach

The typical use of the Dijkstra algorithm is to get only one shortest path, as the pseudo code in Figure 23 shows.



```
1   function Dijkstra(Graph, source):
2       dist[source] ← 0
3
4       create vertex priority queue Q
5
6       for each vertex v in Graph:
7           if v ≠ source
8               dist[v] ← INFINITY
9               prev[v] ← UNDEFINED
10
11          Q.add_with_priority(v, dist[v])
12
13
14      while Q is not empty:
15          u ← Q.extract_min()
16          for each neighbor v of u:
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist, prev
```

Figure 23: Pseudo code of the Dijkstra algorithm
Source: Wikipedia [24]

The code on line 18 checks if the new distance **alt** is smaller than the old distance **dist[v]** and replaces it, if it is the case. To capture all shortest paths, one must simply handle the case that they are equal, so if **alt == dist[v]**.

If this pseudo code is implemented, the paths do not actually get stored, only the distances do. To be able to recreate the shortest paths afterwards, each node must store its parent node. Figure 24 shows an example where Dijkstra has been calculated for node A. The shortest path to node E can be reconstructed by starting at node E, and then going backwards up the tree, visiting each parent until node A is reached.

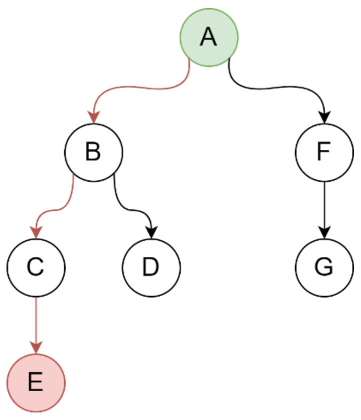


Figure 24: Example - Shortest path from A to E
Source: own creation

How the shortest path reconstruction works is discussed in more detail in Chapter 8.2.4.

8.2.3 Space Complexity

Storing only the parent for each node instead of the entire path saves a huge amount of space. If only the parent is stored for each node, the space complexity is as follows:

$$O(N) \tag{7}$$

Of course, this is only the space complexity for a single Dijkstra, so for the entire network the space complexity will be the following:

$$O(N^2) \tag{8}$$

If on the other hand the entire paths were stored, the space complexity would depend on how balanced the resulting tree from the Dijkstra calculation is.
In the best case, when the starting node has a direct link to every other node, the space complexity would still be O(N).
In the worst case, when the tree is completely unbalanced and basically just a long chain, the space the paths would require would be as follows:

$$1 + 2 + 3 + \cdots + N = \frac{n(n+1)}{2} \tag{9}$$

This means the space complexity for a single Dijkstra that stores all complete paths is O(N^2). Doing this for the entire network would result in the following time complexity:

$$O(N^3) \tag{10}$$

Therefore, only the parents are stored for each node, which lowers the space complexity to O(N^2). How these memory requirements can be managed is discussed in Chapter 9.

8.2.4 Shortest Path reconstruction

To be able to reconstruct the shortest paths, a target router needs to know its parents through which any given start router can reach it.

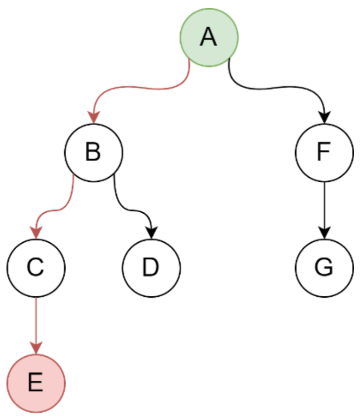
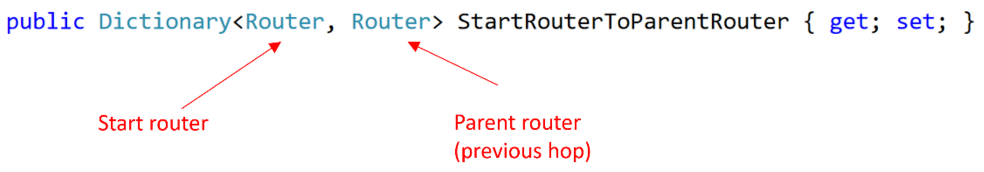


Figure 25: Example - Shortest path from A to E
Source: own creation

Consider the example in Figure 25. At this point Dijkstra has been calculated for start router "A" and each router knows its parent router that leads back to "A".
This information is stored in hash tables. Each router has a hash table, where the key is the start router, and the value is the parent router (in other words the "previous hop router"). Figure 26 shows the implementation in C# using a dictionary. Every router has a "StartRouterToParentRouter"-mapping.

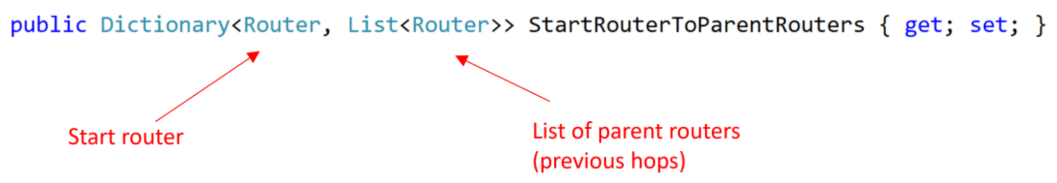


```
public Dictionary<Router, Router> StartRouterToParentRouter { get; set; }
```

Figure 26: C# Implementation of hash map for shortest path reconstruction
Source: own creation

To reconstruct the shortest path from the start router “A” to the target router “E”, the reconstruction algorithm will begin at target router “E”. Router “E” can look up the desired start router “A” in its hash table and will find that its parent router (previous hop) is router “C”. Router “C” in turn can look up the start router “A” in its own hash table and find that its parent router is router “B”. This process repeats until the start router “A” has been reached, at which point the shortest path is reconstructed.

Because for this project multiple equal cost shortest paths are possible and all of them need to be considered, there are multiple possible parent routers for each step. This means that the value of the hash table is not a single parent router but rather a list of parent routers. Figure 27 shows the updated implementation in C#.



```
public Dictionary<Router, List<Router>> StartRouterToParentRouters { get; set; }
```

Figure 27: Updated implementation to support multiple equal cost shortest paths
Source: own creation

8.2.5 Priority Queue

C# does not have a native implementation of a heap-based priority queue. However, there are plenty of implementations available online.

The one chosen for this project is the **FastPriorityQueue** from the GitHub repository of BlueRaja [25]. They provide several priority queue implementations for C#. The **FastPriorityQueue** [26] is designed to be as fast as possible, especially for pathfinding, which means it is perfectly design for this project.

8.2.6 Performance Analysis

To test the custom implementation of the shortest path calculation, several mock topologies were created. Because these tests run on a more powerful machine than the ArangoDB, a direct comparison to the results in Chapter 8.1.4 cannot be made. Still, it is obvious that this solution scales a lot better.

The tests were run on a computer with a CPU with 4 cores and 8 threads, each core running at 4GHz, and 32GB of RAM. Since the implementation is parallelized, it was able to take full advantage of the 8 threads, utilizing 100% of the CPU. The results are shown in Table 5.

Nodes	Edges (Bidirectional)	Duration
500	1’500	0.10 seconds
1’000	3’000	0.52 seconds
1’000	10’000	1.10 seconds
3’000	9’000	8.98 seconds

Table 5: Results of testing custom implementation for shortest path calculation

These results look a lot more promising than those from the ArangoDB.

The reconstruction of the shortest paths between two specific nodes can be done very quickly, as discussed in Chapter 8.2.4. Reconstructing all shortest paths of the entire network however (in case this is necessary) would take a lot longer with the following time complexity.

$$O(N^2 * P_{\emptyset-length}) \tag{11}$$

$P_{\emptyset\text{-length}}$ = The average length of a path

This is because there are at least N^2 shortest paths (assuming no equal cost shortest paths exist) and for each path its entire length needs to be traversed. Reconstructing all shortest paths of the topologies from Table 5 result in the measurements shown in Table 6.

Nodes	Edges (Bidirectional)	Duration
1’000	3’000	0.66 seconds
1’000	10’000	0.56 seconds
3’000	9’000	4.59 seconds
3’000	27’000	5.76 seconds
10’000	30’000	60.0 seconds

Table 6: Reconstructing all shortest paths

8.2.7 Estimating the duration for any N number of nodes

Estimating the duration for N number of nodes may be useful for larger N, when testing it would take too much time. This is possible with some mathematics.

Since the time complexity of the algorithm directly correlates to the duration, the duration can be expressed as follows:

$$p_N = N^2 * \log(N) \tag{12}$$

To find the duration p_{kN} for any k, the formula looks like this:

$$p_{kN} = (kN)^2 * \log(kN) \tag{13}$$

This can also be written as follows:

$$p_{kN} = k^2N^2 * \log(kN) * \frac{\log(N)}{\log(N)} \tag{14}$$

$$p_{kN} = k^2 * \frac{\log(kN)}{\log(N)} * N^2 * \log(N) \tag{15}$$

Replacing N^2*log(N) with p_N results in the final formula:

$$p_{kN} = k^2 * \frac{\log(kN)}{\log(N)} * p_N \tag{16}$$

This formula holds true if the number of edges relative to the number of nodes stays consistent. So, to calculate the duration for 100'000 Nodes and 300'000 Edges, the result from 10'000 Nodes and 30'000 Edges can be taken as a baseline. The calculation is as follows:

$$p_N = 62.7s \mid N = 10'000 \mid k = 10$$
$$p_{kN} = k^2 * \frac{\log(k * N)}{\log(N)} * p_N = 7'837.5s = 2.18h \tag{17}$$

This means with the same hardware used for the performance analysis in Chapter 8.2.6, it would take the algorithm roughly 2.18 hours to complete for 100'000 Nodes and 300'000 Edges. To speed up this computation, more powerful hardware would be required.

8.2.8 Conclusion

The fact that ArangoDB has a time complexity of O(N^4*log(N)) and the custom implementation only O(N^2*log(N)) makes it clear that ArangoDB is not a viable option, so the custom C# implementation will be used for this project. The test results presented in Chapters 8.1.4 and 8.2.6 support that decision.

For reasons discussed in Chapter 8.2.1, the C# implementation cannot be significantly improved upon anymore, so this is as fast as it is going to get.

9 Managing space requirements

During the course this project it became apparent, that just like the time complexity, the space complexity might pose an issue. Requiring O(N^2) memory becomes unfeasible for large N.

This thesis only requires the SR-Apps application to handle up to 1’000 nodes. Since the implementations of the shortest path calculation and the link saturation both run very fast, no data (besides the Dijkstra parent mapping) currently needs to be stored, since everything is simply recalculated when needed.
Nevertheless, this chapter discusses the space requirements and how it can be managed in case of larger topologies.

There are two types of information that need to be stored for this project. The first are the parent lists for each router, as discussed in Chapter 8.2.4 for the shortest path reconstruction.

The other information is associated to the links. The use case Link Saturation Prediction requires the app to allow the user to remove a link and to update the modified network. With a link removed, the shortest paths that previously passed through it need to be recalculated.
This means that in addition to knowing how many shortest paths pass through it, a link must also be able to identify these paths. This can be done by storing a mapping of start routers and target routers on each link.

The space requirements of storing these two types of information are discussed in the subsequent chapters.

9.1 Why is it necessary to store anything?

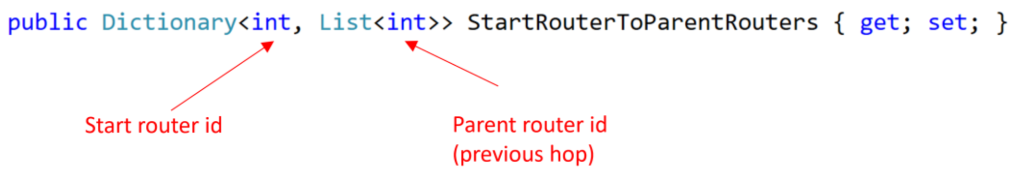
One of the two reasons to why the shortest paths calculation is done at all is to determine the importance of a link. The more shortest paths pass through a link, the more important it is. For this, no other information needs to be stored than a simple 32-bit counter per link that represents the number of shortest paths passing through it.

The other reason why the shortest paths calculation is done is to allow the user to simulate a link failure. If the user wishes to know how the network would react if a specific link went down, the modified topology needs to recalculate all shortest paths. This in itself also does not require storing anything more than the 32-bit counter per link.
However, if a link goes down, its traffic is redirected through other links. To calculate this the shortest path calculation of the modified network needs to be compared to the original one. For this purpose, the shortest path calculation of the original network needs to be stored.

9.2 Storing parent lists

As discussed in Chapter 8.2.3, storing the parent routers to be able to reconstruct the shortest paths presents a space complexity of O(N^2). As shown in Figure 27 in Chapter 8.2.4, the data structure required to store the information needed for path reconstruction is a hash table with a router as key and a list of routers as value.

Because the space requirements may become an issue, it is more space efficient to only store the router ids (which are 32 bit integer values) instead of the router references (which are 32 bit long on a 32-bit machine, but are 64 bit long on a 64-bit machine). This means that the smallest possible data structure to store the necessary information is a hash table with 32-bit integers as keys and a list of 32-bit integers as values. The C# implementation of this is shown in Figure 28.

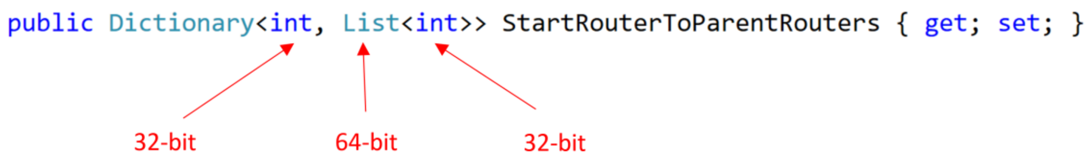


```
public Dictionary<int, List<int>> StartRouterToParentRouters { get; set; }
```

Figure 28: Most space efficient implementation of parent storing
Source: own creation

Since each router needs such a dictionary there will be N dictionaries, and because each router needs to know the shortest paths to every other there will be N entries in each dictionary. The number of entries in the parent list depends on the number of shortest paths. Chapter 9.4 shows the required space for different network sizes. For these examples it is assumed that these lists only contain one parent (so the network in question does not happen to have any multiple equal cost shortest paths). It is also assumed that the software is running on a 64-bit machine.

Under these assumptions a single entry would have a size of 128 bit (see Figure 29).



```
public Dictionary<int, List<int>> StartRouterToParentRouters { get; set; }
```

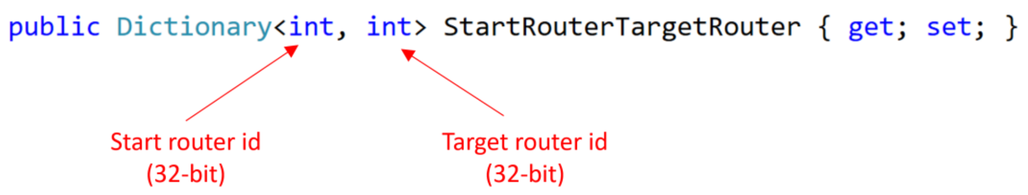
Figure 29: Space requirement per entry
Source: own creation

In generalized form, this results in the following space requirement for an entire network with N nodes:

$$Required\ space\ for\ parent\ lists = 128bit * N^2 \qquad (18)$$

9.3 Storing path identifiers on links

To be able to know which shortest paths pass through any given link, each link needs to have a hash table with the start router as key and the target router as value. With this information all paths passing through this link can be reconstructed. Figure 30 shows the implementation in C#, using a dictionary.



```
public Dictionary<int, int> StartRouterTargetRouter { get; set; }
```

Figure 30: Dictionary storing path identifiers - one per link
Source: own creation

This results in M_U dictionaries (M_U being the number of links) with 64 bit per entry. The number of entries (henceforth represented by the variable "k") depends on the number of shortest paths going through a particular link. The average number of entries in the dictionaries is represented by the variable "$k_{average}$".
In generalized form this results in the following space requirement for an entire network with M_U edges:

$$Required\ space\ for\ path\ identifiers = 64bit * M_U * k_{average} \tag{19}$$

The value for $k_{average}$ varies greatly and directly correlates to the size of the network. Unfortunately, the average k of a network not only depends on the size of the network but also varies depending on how exactly the topology looks like, which means there is no definitive way to calculate it in generalized form. However, Chapter 9.3.1 goes into more detail of what $k_{average}$ is and how it can be estimated.
The values chosen for $k_{average}$ for the space calculations done in Chapter 9.4 are empirical values based on mocked topologies with random routers and links.

9.3.1 Estimating $k_{average}$

As discussed in Chapter 9.3, $k_{average}$ is the average number of shortest paths going through a particular link. This can be expressed with the following equation.

$$k_{average} = \frac{K_{total}}{M_U} \tag{20}$$

K_{total} = Sum of the k-values of all unidirectional links
M_U = Number of unidirectional links

K_{total} can be expressed in the following way.

$$K_{total} = P_{\emptyset-length} * P_{total} \tag{21}$$

$P_{\emptyset\text{-}length}$ = Average length of a path (number of links visited)
P_{total} = Total number of (shortest) paths in the network

Combining Equations (20) and (21) results in the following formula for $k_{average}$.

$$k_{average} = \frac{P_{\emptyset-length} * P_{total}}{M_U} \tag{22}$$

Combining Equations (19) and (22) results in the following updated formula for the calculation of the required space for path identifiers.

$$Required\ space\ for\ path\ identifiers = 64bit * P_{\emptyset-length} * P_{total} \tag{23}$$

The total number of paths P_{total} has a minimum value of N^2 (for the case that no multiple equal cost shortest paths exist) whereas the average path length $P_{\emptyset\text{-length}}$ may be guessed through experience or existing data.

9.4 Space requirements depending on network size

To get a feeling of how much space is required for specific network sizes, Table 7 shows some sample calculations. The topologies analyzed are the same ones as those chosen for the performance analysis in Chapter 8.2.6. The values for $k_{average}$ are empirical. They were gathered by simply adding up the k-values of each link and then dividing by M.
For the calculations the Equations (18) and (19) from Chapters 9.2 and 9.3 are applied, which are as follows:

$$Required\ space\ for\ parent\ lists = 128bit * N^2 \tag{18}$$

$$Required\ space\ for\ path\ identifiers = 64bit * M_U * k_{average} \tag{19}$$

N (Nodes)	M_U (U-Dir-Links)	$k_{average}$ (empirical)	Space required for parent lists	Space required for path identifiers	Total space required
1’000	6’000	1’850	15.26 MB	84.69 MB	89.95 MB
1’000	20’000	416	15.26 MB	63.48 MB	78.74 MB
3’000	18’000	6’309	137.3 MB	866.4 MB	1.004 GB
3’000	54’000	2’596	137.3 MB	1.044 GB	1.178 GB
10’000	60’000	23’388	1.490 GB	10.43 GB	11.92 GB

Table 7: Examples for space requirement

9.5 Estimating space requirements for larger networks

Estimating the space requirements of larger networks can be done based on existing sample data. Analyzing the topologies used in Table 7 in more detail presents the information detailed in Table 8.

P_{ratio} is a value that expresses the number of all paths P_{total} in relationship to the minimum number of paths P_{min} in a network, which is always N^2 (see Equation (18)). P_{ratio} is a measure of what percentage of P_{total} are equal cost shortest paths.

$$P_{ratio} = \frac{P_{total}}{P_{min}} \tag{18}$$

P_{total} = Total number of shortest paths in the network
P_{min} = N^2 (minimum number of shortest paths, if no equal cost shortest paths exist)

N	M_U	P_{total}	$P_{Ø\text{-}length}$	P_{ratio}
1’000	6’000	1’467’735	7.56	1.47
1’000	20’000	1’899’111	4.38	1.90
3’000	18’000	13’026’867	8.72	1.45
3’000	54’000	20’695’717	6.77	2.30
10’000	60’000	135’977’501	10.3	1.36

Table 8: Detailed information on sample topologies

Because these topologies were created randomly, these results may not properly reflect reality. But because no real data for such information is available, the randomized data is used to estimate space requirements for larger networks.

Comparing the results in Table 8 implies the following:

- Increasing the number of links while keeping the number of nodes the same results in a lower average path length $P_{Ø\text{-}length}$.
- When increasing the number of nodes, $P_{Ø\text{-}length}$ increases only very slowly.
- With a constant ratio of $N:M_U$ the ratio P_{ratio} also seems to remain constant at roughly 1.43 for a $N:M_U$ ratio of 1:6.

Using this information allows making predictions on space requirements for larger networks. Table 9 shows possible values for such topologies assuming a $N:M_U$ ratio of 1:6 with P_{ratio} = 1.43. P_{total} is calculated by multiplying P_{ratio} and N^2.

N	M_U	P_{total} (calculated)	$P_{Ø\text{-}length}$	P_{ratio}
50’000	300’000	3’575’000’000	12	1.43
100’000	600’000	14’300’000’000	15	1.43

Table 9: Estimated values for large topologies

With Equation (22) from Chapter 9.3.1 the average k-value can be calculated.

$$k_{average} = \frac{P_{\emptyset-length} * P_{total}}{M_U} \tag{22}$$

Table 10 shows the estimated space requirements for the larger topologies based on previous guesswork.

N (Nodes)	M_U (U-Dir-Links)	$k_{average}$ (estimated)	Space required for parent lists	Space required for path identifiers	Total space required
50’000	300’000	143’000	37.25 GB	319.6 GB	356.9 GB
100’000	600’000	357’500	149.0 GB	1.567 TB	1.706 TB

Table 10: Estimated space requirements for large topologies

9.6 Conclusion

For larger topologies it should be considered splitting the topology into several smaller ones, because the amount of data required for such large networks becomes quickly very impractical to handle, reaching approximately 1.7 TB for 100’000 nodes.

An alternative would be to store neither the parent lists nor the path identifiers, and then recalculating the shortest paths when needed. This would pretty much eliminate any space requirements for the handling of shortest paths but would essentially double the time required to do the calculations.

10 Results

This chapter lists the results of the thesis by comparing the features, use cases and non-functional requirements defined in the requirements analysis with the final state of the application.

10.1 Feature coverage

The covered features are shown in Table 11.

Feature	Covered	Comment
Link Grading	Fully	The grading of links uses three different link metrics (link delay, packet loss and link saturation). The user can switch between different maps showing the links graded by these metrics. Adding new maps based on different link metrics is very quick and easy to implement. The importance of a link is calculated by counting the number of shortest paths going through the link. The count is used to display the thickness of the drawn link inside the UI (the more shortest paths go through a link, the thicker it is drawn).
Link Saturation Prediction	Fully	The link saturation is calculated for the whole topology upon topology changes. Topology modifications (adding or removing nodes and links) can be done by the user. The recalculated topology with potentially different link gradings.

Table 11: Feature Coverage

10.2 Use Case Coverage

Table 12 shows the use case coverage.

Use Case	Status		Comment
01 – Load Map	Fully implemented	✓	The user is able to load different maps showing the link grading based on three different metrics.
02 – Zoom In	Fully implemented	✓	The user can either use the "Expand" button or a double click on a region to view all underlying nodes and links.
03 – Zoom Out	Fully implemented	✓	The user can either use the "Collapse" button or double click a router to collapse them into a region.
04 – Show Details	Fully implemented	✓	The user can click on a node or link to view details. The details list the most

			important information like name, ip-addresses and link metrics.
05 – Remove Link	Fully implemented	✓	The user can remove a link from the topology and see the effect it has on the topology.
06 – Add Link	Fully implemented	✓	The user can add a link to the topology and see the changes in terms of link grading and thickness.
07 – Remove Node	Fully implemented	✓	The user can remove a node from the topology and see the effect it has on the topology.
08 – Add Node	Fully implemented	✓	The user can add a node to the topology and see the effect it has on the topology.
09 – Change Link Attributes	Fully implemented	✓	The uer can select a link and edit ist properties. This results in potentially new graded links and modified link thickness.
10 – Load Latest Topology	Fully implemented	✓	The user updates the view inside the UI to display the latest data the server polled from the database.
11 – Manually Poll Topology	Fully implemented	✓	The user can initiate polling new data from the database and view the adapted topology.

Table 12: Use Case Coverage

10.3 Non-functional requirements

Table 13 show which non-functional requirements were fulfilled.

NFR (response measure)	Fulfilled	Comment
All UI components are fully displayed when the application is accessed with Google Chrome 83.0 or higher or Firefox 80.0 or higher. All use cases can still be executed.	Yes	Use cases were tested with both versions and are fully functional.
The application must be scalable for topologies with up to 1'000 enabled routers.	Yes	The architecture, client-server communication and calculation inside the business logic works very well for 1’000 nodes.
The response time in the UI of the application must not exceed 10 seconds in 90% of all interactions that involve server calls. In the remaining 10% the response is within 15 seconds.	Yes	The response time for all use case walkthroughs is never > 10 seconds.

The operator can switch between the different maps with two clicks and gets additional information about nodes and links with one click.	Yes	By opening a dropdown menu (one click) and selecting the desired map (one click) a new map can be displayed. By clicking on a node or link (one click) the details about the component are shown.
Only one software component needs to be adjusted to implement a new map algorithm or change an existing one. The changes can be made in less than 4 hours.	Yes	Due to a high level of abstraction a new map algorithm can be easily implemented and requires adjusting the DefaultMapConfigurator component.

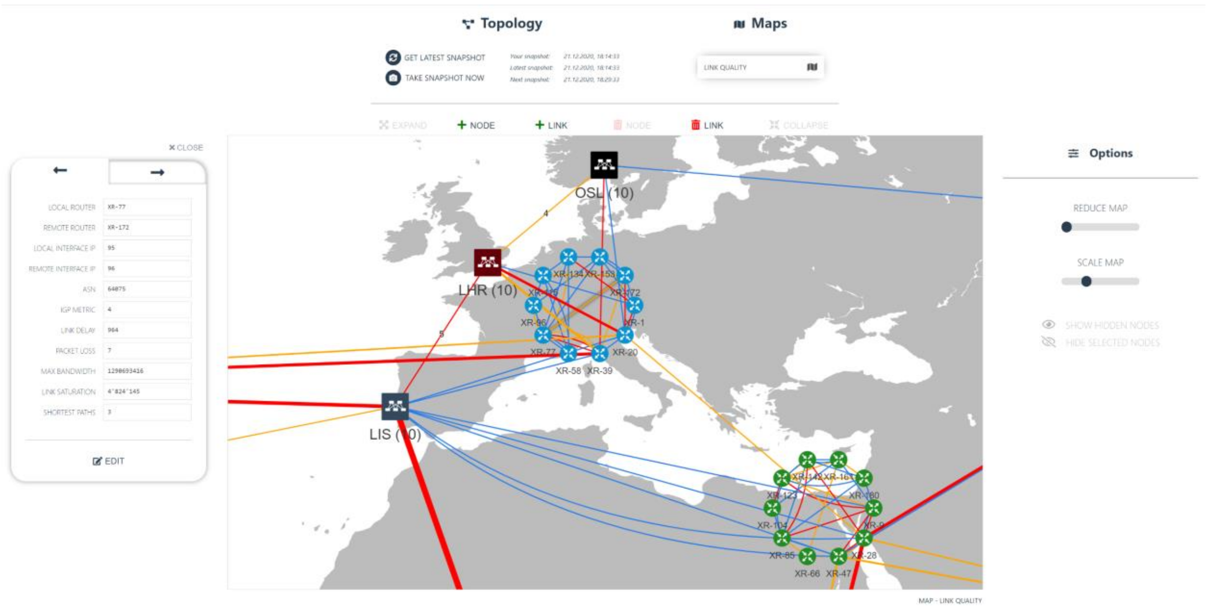
Table 13: Non-functional requirements

11 Conclusion

During this thesis, a fully functioning application was built covering all defined features and use cases. A user can monitor the link state of SR topologies and simulate network changes to see their effects on the network.
Different maps are used to display the link quality based on different link metrics. The more often a link is used the thicker it is drawn in the UI. The calculation of link saturation (which is performed when the user makes changes to the topology) is very performant allowing fast simulations of potential link failures.

The application can be deployed using docker-compose. This allows the deployment on most target systems.

The SR-App is a tool to easily identify weak points in a network. Figure 31 - Figure 34 show the UI of the application.

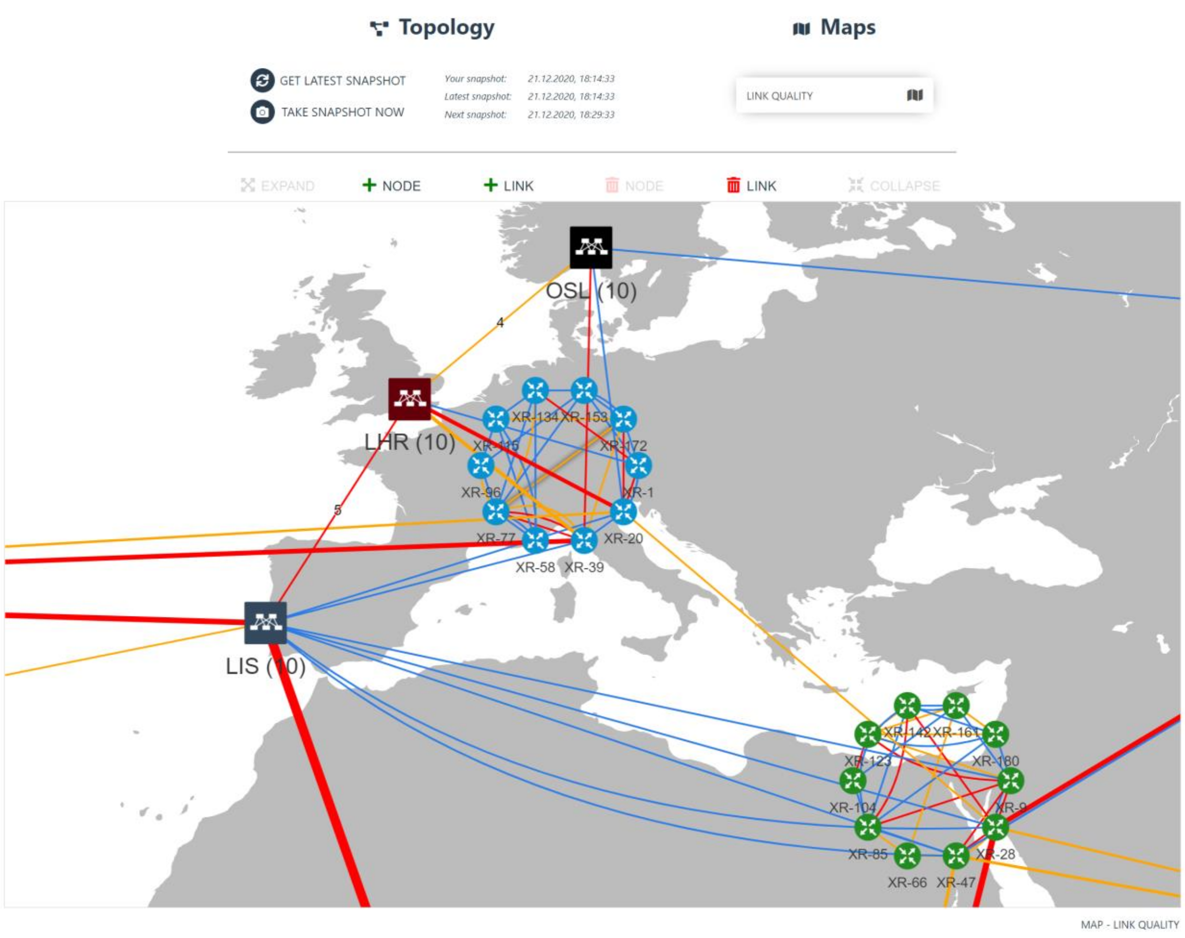


Your snapshot: 21.12.2020, 18:14:33
Latest snapshot: 21.12.2020, 18:14:33
Next snapshot: 21.12.2020, 18:29:33

LINK QUALITY

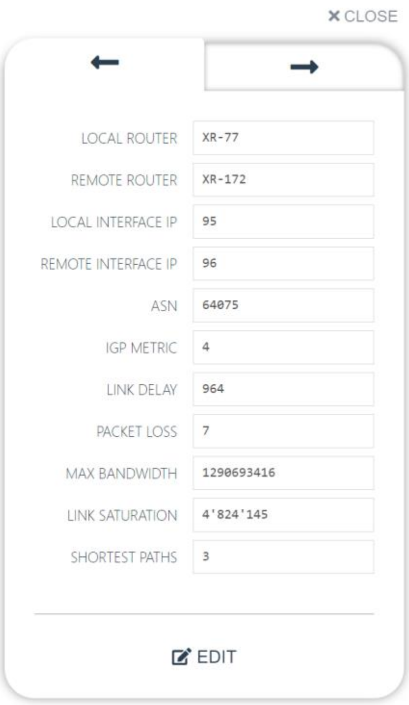
LOCAL ROUTER	XR-77
REMOTE ROUTER	XR-172
LOCAL INTERFACE IP	95
REMOTE INTERFACE IP	96
ASN	64075
IGP METRIC	4
LINK DELAY	904
PACKET LOSS	7
MAX BANDWIDTH	1298693416
LINK SATURATION	4'824'145
SHORTEST PATHS	3

Figure 31: Final version SR-App
Source: own creation



Your snapshot: 21.12.2020, 18:14:33
Latest snapshot: 21.12.2020, 18:14:33
Next snapshot: 21.12.2020, 18:29:33

Figure 32: Map in main view
Source: own creation



LOCAL ROUTER	XR-77
REMOTE ROUTER	XR-172
LOCAL INTERFACE IP	95
REMOTE INTERFACE IP	96
ASN	64075
IGP METRIC	4
LINK DELAY	964
PACKET LOSS	7
MAX BANDWIDTH	1290693416
LINK SATURATION	4'824'145
SHORTEST PATHS	3

Figure 33: Details of a (unidirectional) link
Source: own creation

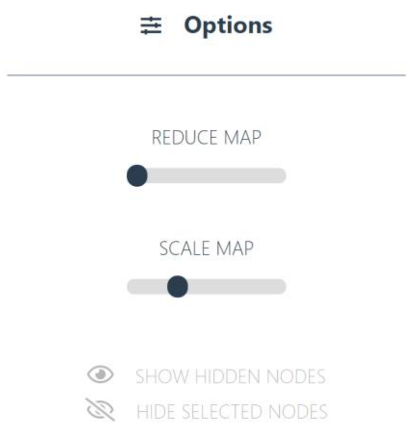


Figure 34: Map options
Source: own creation

11.1 Outlook

While the current version of the application covers all predefined features and use cases, there is still a lot more that can be done to improve it. This chapter discusses possible additional features that may be of interest.

11.1.1 Support Layer 3 EtherChannel

Some networks make use of network-layer EtherChannels, which are logical links consisting of multiple physical links. The reason for this is it to provide redundancy to a connection between two routers. The IP-address is assigned to the logical link, which ensures connectivity even if some physical connections in the aggregated link fail.
To support this kind of link, the data model in the business logic and frontend would have to be updated. Further, it would have to be evaluated what kind of impact this change has on the business logic.

11.1.2 Allow the user to adjust the link grading dynamically

In the current version of the application the thresholds to grade links are fixed values. To give the user more control over the grading system the UI could contain controls to change these thresholds. This would result in the user having control for which threshold a link is shown as orange or as red.

The implementation of this feature would have only a minimal impact on the business logic.

11.1.3 Interface providing region data

Region data is the information of which router belongs to which region. This information might come from various sources such as the name of the router, external systems or provided by the user in textual form. External systems may even contain exact coordinates of the routers.

To properly process and use this information the application would have to be extended accordingly.

12 Glossary

Term	Description
API	Application Programming Interface
CPU	Central Processing Unit
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
KB	Kilobyte
MB	Megabyte
RAM	Random Access Memory
REST	Representational State Transfer
SR	Segment Routing
TB	Terabyte

Table 14: Glossary

13 Illustration index

14 Sources

[1] "Microsoft Documentation - SignalR," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0. [Accessed 22 12 2020].

[2] "Medium - List of graph visualization libraries," [Online]. Available: https://medium.com/@Elise_Deux/the-list-of-graph-visualization-libraries-7a7b89aab6a6. [Accessed 24 09 2020].

[3] "VisJS Homepage," [Online]. Available: https://visjs.org/. [Accessed 24 09 2020].

[4] "NeXt UI GitHub page," [Online]. Available: https://github.com/NeXt-UI. [Accessed 24 09 2020].

[5] "NeXt UI - GitHub - Next," [Online]. Available: https://github.com/NeXt-UI/next. [Accessed 24 09 2020].

[6] "Vinllen NeXT UI Tutorial - Who developed NexT UI," [Online]. Available: http://vinllen.com/next-ui-tutorial-supplement/. [Accessed 24 09 2020].

[7] "NeXt UI Tutorials," [Online]. Available: https://github.com/NeXt-UI/next-tutorials. [Accessed 24 09 2020].

[8] "Vis.js documentation for network graphs," [Online]. Available: https://visjs.github.io/vis-network/docs/network. [Accessed 24 09 2020].

[9] "VisJs Network - GitHub," [Online]. Available: https://github.com/visjs/vis-network. [Accessed 24 09 2020].

[10] "NeXt UI API Reference," [Online]. Available: https://developer.cisco.com/codeexchange/github/repo/NeXt-UI/next-tutorials/. [Accessed 13 12 2020].

[11] "Vis.js documentation on physics," [Online]. Available: https://visjs.github.io/vis-network/docs/network/physics.html. [Accessed 28 09 2020].

[12] "Vis-Network Examples," [Online]. Available: https://visjs.github.io/vis-network/examples/. [Accessed 24 09 2020].

[13] "Vis.js Examples - Changing clustered edges and nodes," [Online]. Available: https://visjs.github.io/vis-network/examples/network/other/changingClusteredEdgesNodes.html. [Accessed 24 09 2020].

[14] "GitHub Vis.js - Discussion on grouping," [Online]. Available: https://github.com/visjs/vis-network/issues/203. [Accessed 17 10 2020].

[15] "GitHub - Muntaner visjs districts implementation," [Online]. Available: https://github.com/Muntaner/visjs_districts/tree/master/js. [Accessed 17 10 2020].

[16] "GitHub - vis.js KamadaKawai implementation," [Online]. Available: https://github.com/visjs/vis-network/blob/master/lib/network/modules/KamadaKawai.js. [Accessed 17 10 2020].

[17] "Wikipedia - Force-directed graph drawing," [Online]. Available: https://en.wikipedia.org/wiki/Force-directed_graph_drawing. [Accessed 17 10 2020].

[18] "ArangoDB Homepage," [Online]. Available: https://www.arangodb.com/. [Accessed 17 10 2020].

[19] "ArangoDB - Shortest-Path," [Online]. Available: https://www.arangodb.com/docs/stable/aql/graphs-shortest-path.html. [Accessed 17 10 2020].

[20] "ArangoDB - K-Shortest-Path," [Online]. Available: https://www.arangodb.com/docs/stable/aql/graphs-kshortest-paths.html. [Accessed 17 10 2020].

[21] "Clicage - ArangoDB PDF," [Online]. Available: http://www.clicage.com/ilyatoo/objets/oeuvres/ARANGODB.pdf. [Accessed 17 10 2020].

[22] "ArangoDB GitHub," [Online]. Available: https://github.com/arangodb/arangodb/pull/8715. [Accessed 17 10 2020].

[23] "Wikipedia - Yen's algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Yen%27s_algorithm. [Accessed 17 10 2020].

[24] "Wikipedia - Dijkstra Algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. [Accessed 17 10 2020].

[25] "GitHub - BlueRaja PriorityQueue," [Online]. Available: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp. [Accessed 17 10 2020].

[26] "GitHub - BlueRaja FastPriorityQueue," [Online]. Available: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp/wiki/Using-the-FastPriorityQueue. [Accessed 17 10 2020].

[27] "Microsoft SignalR documentation," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-5.0. [Accessed 17 12 2020].

[28] "Juniper - What is segment routing?," [Online]. Available: https://www.juniper.net/us/en/products-services/what-is/segment-routing/. [Accessed 20 09 2020].

[29] "Jalapeño GitHub repository," [Online]. Available: https://github.com/cisco-ie/jalapeno. [Accessed 17 12 2020].

[30] "GitHub BlueRaja - High Speed Priority Queue," [Online]. Available: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp. [Accessed 14 12 2020].

SR-App Analytics
E – Attachments

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

SR-App Analytics

E-1 – Requirements Analysis

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

1 Content

2 General description

2.1 Product perspective

The SR-App allows a user to monitor an SR topology and to simulate changes in the network, such as adding or removing links and nodes. The application displays a simple and understandable view of an existing segment routing topology to visualize and cover the features Link Saturation Prediction and Link Quality Assessment which are defined in this document.

Possible applications for this software are the following:

- Monitoring of the health state of an existing SR network.
- Predicting the consequences of a link or node failure by simulating it through the UI.
- Better predicting the benefits of adding a specific new link or node to the network by simulating it through the UI. This is especially useful to decide which investment would improve the network the most.

2.2 Product functionality

The product should provide the following functionality:

- Display an existing SR topology with all its nodes and links.
- Grade links based on certain attributes such as packet loss, link delay and link saturation and display this information visually.
- Allow the user to simulate network changes in case of link and node failures.
- Allow the user to simulate network changes in case of adding links and nodes.

2.3 User characteristics

The primary users of the application are network operators and network engineers.

2.4 External Dependencies

This application depends on the Jalapeño software system, which is provided by the industry partner Cisco.

Jalapeño collects data of an SR topology. The data is sent by the routers via BMP and streaming telemetry to Jalapeño and processed there further before it is stored in a database. The SR-App directly queries this database.

3 Features

The required features are derived from the problem definition provided by the advisor and the project partner. The following chapters describe two features called Link Saturation Prediction and Link Quality Assessment. The use cases discussed in Chapter 4 are derived from these features.

3.1 Feature 1 – Link Quality Assessment

The Link Quality Assessment feature grades all links in a topology based on predefined metrics. The grading is shown to the user using different color codes indicating the quality of a link. This allows the user to quickly gain an overview of the topologies health state and to identify critical links.

3.2 Feature 2 – Link Saturation Prediction

The Link Saturation (bandwidth utilization) Prediction feature allows the SR-Operator to simulate topology changes. Topology changes can be the removal or adding of links and nodes.
Because any such alteration to a network would in most cases result in changed shortest paths, the saturation of the individual links would change as well. Therefore, to accurately predict the saturation of links, the path of all traffic in the topology must be recalculated so the traffic can be redistributed.

4 Use Cases

This chapter describes all use cases derived from the requested features. Figure 1 shows all use cases in a use case diagram. Optional use cases are displayed in yellow color.

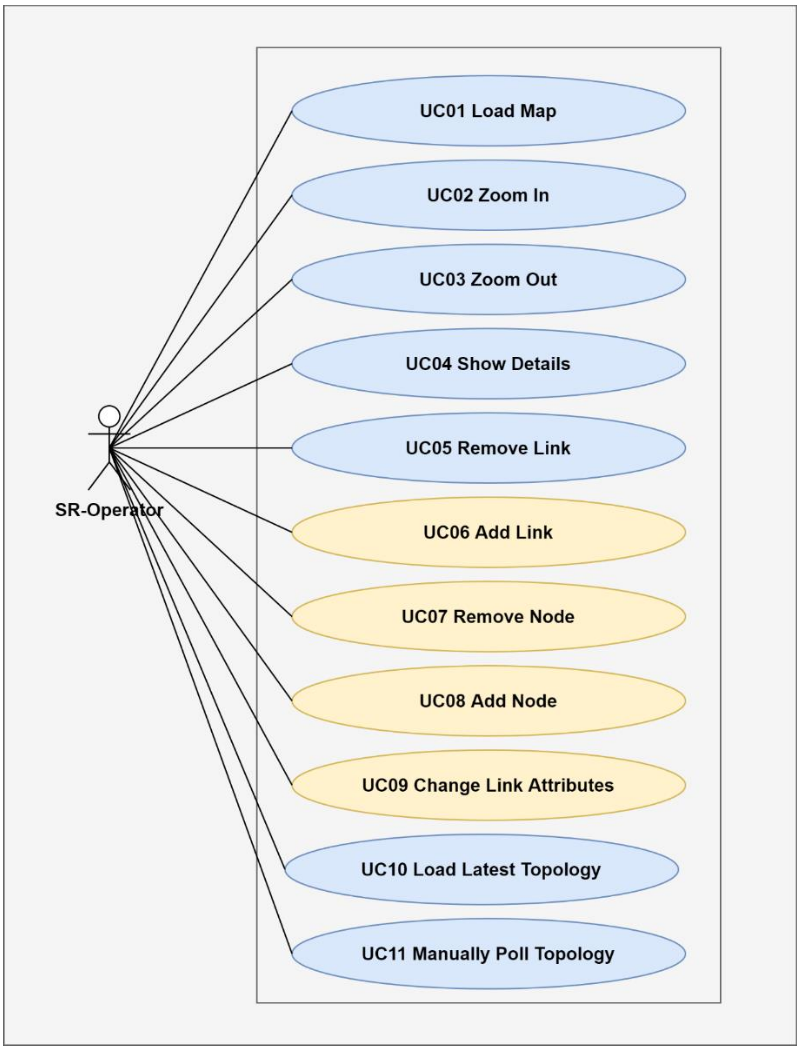


Figure 1: Use case diagram
Source: own creation

4.1 Actors & stakeholders

The only actor is the user of the application (SR-Operator). The interaction between the actor and the application is limited to the interaction needed to perform the use cases.

4.2 Description (fully dressed)

4.2.1 Use case 01 – Load Map

The user can switch between different maps of a topology. The different maps (heat maps) are highlighted by either packet loss, link delay, link saturation, or a combination of them.

Use Case ID	**01**
Use Case name	Load Map
Actors	SR-Operator
Description	The operator loads a specific map (Packet Loss, Link Delay, Link Quality, Link Saturation) in the UI.
Preconditions	SR-App is fully loaded, and the default map (Link Quality) is displayed.
Postconditions	The requested map of the topology is shown to the SR-Operator.
Normal Flow	• The SR-Operator loads the SR-App website. • The website displays the default map (Link Quality). • The SR-Operator would like to see another map and selects the desired map in the drop-down menu. • The topology is loaded from the server and displayed in the browser
Alternative Flow	-
Exceptions	• Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	Connection to the database needs to be established
Assumptions	-
Notes and Issues	-

Table 1: Use case 01

4.2.2 Use case 02 – Zoom In

By default, only regions (clusters of routers) are shown. To view the routers in a particular region, the user can zoom into it (expanding the cluster).

Use Case ID	**02**
Use Case name	Zoom In
Actors	SR-Operator
Description	The operator zooms into a specific cluster (region)
Preconditions	The user can see the topology in the browser. At least one cluster is visible.
Postconditions	The cluster is no longer visible, but all its routers are shown instead.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator double-clicks a specific cluster or selects a cluster and clicks the "Expand" button. • The expanded cluster is shown in more detail with all its routers and links.
Alternative Flow	-
Exceptions	• Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 2: Use case 02

4.2.3 Use case 03 – Zoom out

To zoom out of a region, the user can collapse the cluster.

Use Case ID	**03**
Use Case name	Zoom Out
Actors	SR-Operator
Description	The operator zooms out of a cluster (region) to collapse it.
Preconditions	The user can see the topology in the browser. At least one cluster is expanded.
Postconditions	The routers of the region are no longer visible, but all its cluster is shown instead.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator expands a region. • The SR-Operator double-clicks a specific router or selects a router and clicks the "Collapse" button. • The collapsed cluster is shown and all its routers and links are removed.
Alternative Flow	-
Exceptions	• Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 3: Use case 03

4.2.4 Use case 04 – Show Details

The user can view the details of a node or link by selecting it in the UI.

Use Case ID	**04**
Use Case name	Show Details
Actors	SR-Operator
Description	The user selects a node or link, and its details are displayed in the browser.
Preconditions	The user can see the topology in the browser.
Postconditions	The details about the selected component are shown to the user.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator expands a region. • The SR-Operator selects a link or node. • The browser displays its details.
Alternative Flow	-
Exceptions	• Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 4: Use case 04

4.2.5 Use case 05 – Remove Link

The user can remove a link from the topology.

Use Case ID	**05**
Use Case name	Remove Link
Actors	SR-Operator
Description	The user removes a link from the topology.
Preconditions	The user can see the topology in the browser. At least one cluster is expanded, so a link is shown on the map.
Postconditions	The SR-Operator sees the map without the deleted link and the saturation of all links has been updated to reflect the change in the topology.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator expands a region. • The SR-Operator selects a link. • The SR-Operator clicks the "Remove Link" button. • The browser displays a loading screen. • The updated map is displayed.
Alternative Flow	-
Exceptions	• Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 5: Use case 05

4.2.6 Use case 06 – Add Link

The user can add a link to the topology.

Use Case ID	**06**
Use Case name	Add Link
Actors	SR-Operator
Description	The user adds a link to the topology.
Preconditions	The user can see the topology in the browser. The clusters are expanded, such that at least two routers are visible on the map.
Postconditions	The SR-Operator sees the map with the newly created link and the saturation of all links has been updated to reflect the change in the topology.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator expands a region. • The SR-Operator clicks the "Add Link" button. • The SR-Operator draws a link between two nodes. • The browser displays a loading screen. • The updated map is displayed.
Alternative Flow	-
Exceptions	• Connectivity to the database is lost -> Error Message "Connection to database lost" • The SR-Operator tries to draw a link between two clusters or between a cluster and a router. This is not allowed, and the user will be shown an error message. • The SR-Operator tries to draw a loopback link on a router. This is not allowed, and the user will be shown an error message.
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 6: Use case 06

4.2.7 Use case 07 – Remove Node

The user can delete a router from the topology.

Use Case ID	**07**
Use Case name	Remove Node
Actors	SR-Operator
Description	The user deletes a router from the topology.
Preconditions	The user can see the topology in the browser. The clusters are expanded, such that at least one router is visible on the map.
Postconditions	The SR-Operator sees the map without the deleted router and the saturation of all links has been updated to reflect the change in the topology.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator expands a region. • The SR-Operator selects a router. • The SR-Operator clicks the "Remove Node" button. • The updated map is displayed.
Alternative Flow	-
Exceptions	Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 7: Use case 07

4.2.8 Use case 08 – Add Node

The user can add a router to the topology.

Use Case ID	**08**
Use Case name	Add Node
Actors	SR-Operator
Description	The user adds a router to the topology.
Preconditions	The user can see the topology in the browser.
Postconditions	The SR-Operator sees the map with the newly created router and the saturation of all links has been updated to reflect the change in the topology.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator clicks on the “Add Node” button. • The updated map is displayed, with the newly created router placed on the center of the users field of view.
Alternative Flow	-
Exceptions	Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 8: Use case 08

4.2.9 Use case 09 – Change Link Attributes

The user can change certain attributes of links.

Use Case ID	**09**
Use Case name	Change Link Attributes
Actors	SR-Operator
Description	The user changes link attributes.
Preconditions	The user can see the topology in the browser. The clusters are expanded, such that at least one link is visible on the map.
Postconditions	The SR-Operator sees the updated map which reflects the changes made to the link. The saturation of all links has been updated to reflect the change in the topology.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator expands a region. • The SR-Operator selects a link and clicks the "Edit" button. • The SR-Operator changes the attribute values of the link. • The updated map is displayed.
Alternative Flow	-
Exceptions	Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 9: Use case 09

4.2.10 Use case 10 – Load Latest Topology

The user can update his or her snapshot of the topology to the latest version available on the server.

Use Case ID	**10**
Use Case name	Load Latest Topology
Actors	SR-Operator
Description	The user updates his or her snapshot of the topology to the latest version available on the server.
Preconditions	The user can see the topology in the browser.
Postconditions	The SR-Operator sees the latest topology available on the server.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator has been using the application for several minutes or more. • The SR-Operator would like to see the latest topology snapshot. • The SR-Operator clicks the "Get Latest Snapshot" button. • The same map is shown with the latest data available on the server.
Alternative Flow	-
Exceptions	Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 10: Use case 10

Use case 11 – Manually Poll Topology

The user can trigger a fresh poll from the database on the server.

Use Case ID	**11**
Use Case name	Manually Poll Topology
Actors	SR-Operator
Description	The user triggers a fresh poll from the database on the server.
Preconditions	The user can see the topology in the browser.
Postconditions	The SR-Operator sees the latest topology available on the database.
Normal Flow	• The SR-Operator has loaded a map. • The SR-Operator has been using the application for several minutes or more. • The SR-Operator would like to see the latest data available on the database. • The SR-Operator clicks the "Take Snapshot Now" button • The same map is shown with the latest data available in the database.
Alternative Flow	-
Exceptions	Connectivity to the database is lost -> Error Message "Connection to database lost"
Priority	High
Frequency of Use	High
Special Requirements	-
Assumptions	-
Notes and Issues	-

Table 11: Use case 11

5 Non-functional Requirements

5.1 NFR01 - Compatibility

NFR	Compatibility
Business objective	Use of the application on different browsers
Stimulus	Operator accesses the application on different browsers.
Response	The operator gets the same results on each supported browser as soon the page is loaded.
Response Measure	All UI components are fully displayed when the application is accessed with Google Chrome 83.0 or higher or Firefox 80.0 or higher. All use cases can still be executed.

Table 12: NFR01

5.2 NFR02 - Scalability

NFR	Scalability
Business objective	The operator can use the application for small to medium sized topologies.
Stimulus	The operator accesses the application.
Response	The operator sees the correct topology with all information with the performance corresponding to what is defined in Chapter 5.3 for small and medium sized topologies.
Response Measure	The application must be scalable for topologies with up to 1'000 enabled routers.

Table 13: NFR02

5.3 NFR03 - Efficiency

NFR	Efficiency
Business objective	Good efficiency and real advantage for the operator.
Stimulus	Operator opens the browser and navigates to the sites displaying topology information.
Response	The operator sees the topology and all associated data within the set deadline.
Response Measure	The response time in the UI of the application must not exceed 10 seconds in 90% of all interactions that involve API calls. In the remaining 10% the response is within 15 seconds.

Table 14: NFR03

5.4 NFR04 - Usability

NFR	Usability
Business objective	Easy to use and intuitive application.
Stimulus	The operator navigating the web application. During browsing he or she wants to see another topology.
Response	The operator can switch between maps easily.
Response Measure	The operator can switch between the different maps with two clicks and gets additional information about nodes and links with one click.

Table 15: NFR04

5.5 NFR05 - Changeability

NFR	Changeability
Business objective	Changes and improvements of the business logic can be made fast and easily.
Stimulus	Developer wants to change or implement a new map algorithm for link grading to be able to add additional heat maps to the application.
Response	There are no side effects on other system components.
Response Measure	Only one software component needs to be adjusted to implement a new map algorithm or change an existing one. The changes can be made in less than 4 hours.

Table 16: NFR05

6 Glossary

Term	Description
BMP	BGP (Border Gateway Protocol) Monitoring Protocol
SR	Segment Routing
UI	User Interface

Table 17: Glossary

SR-App Analytics
E-2 – Project Plan

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

1 Content

2 Introduction

2.1 Purpose

This document contains all necessary documentation for the thesis SR-App Analytics. It contains the planning of this project and acts as a guideline to comprehend the methods used. This project plan contains a summary of the project and an overview of the project organization.

2.2 Validity scope

This document is valid as part of the thesis SR-App Analytics.

3 Project overview

3.1 Purpose and aim

> "Segment routing (SR) is a source-based routing technique that simplifies traffic engineering and management across network domains. It removes network state information from transit routers and nodes in the network and places the path state information into packet headers at an ingress node."
>
> - Cited from Juniper.net [1]

SR-App provides a user with monitoring functionality for a segment-routing topology. The App displays a simple and understandable view of an existing segment routing topology to visualize and cover the use cases Link Saturation Prediction and Link Quality Assessment which are defined in the document "Requirements Analysis".

For the user, the app is intended to create added value in monitoring the health of a segment-routing network as well as simulating changes in the topology.

3.2 Delivery

3.2.1 Results

- Source code
- Software architecture document
- Requirement analysis documentation

3.2.2 Project management

- Project plan
- Time tracking evaluation
- Project statistics

4 Project organization

The project will contain four phases: Inception, Elaboration, Construction and Transition. Within these phases scrum will be utilized in order to remain agile.

4.1 Organizational structure

Name	Position	Email	Responsibilities and Tasks
Dominique Illi	Developer	dominique.illi@ost.ch	- Development - Takes times
Michel Bongard	Developer	michel.bongard@ost.ch	- Development - Jira, Bamboo & GitHub admin

Table 1: Organizational structure

4.2 External persons

Name	Position	Email	Responsibilities and Tasks
Laurent Metzger	Supervisor	laurent.metzger@ost.ch	- Supervisor - Primary contact Person
Urs Baumann	Supervisor	urs.baumann@ost.ch	- Supervisor - Contact Person
Francois Clad	Supervisor	fclad@cisco.com	- Define Scope - Primary liaison to Cisco

Table 2: External Persons

5 Management procedures

5.1 Time management

The following timetable provides an overview of the project phases, iterations and milestones.

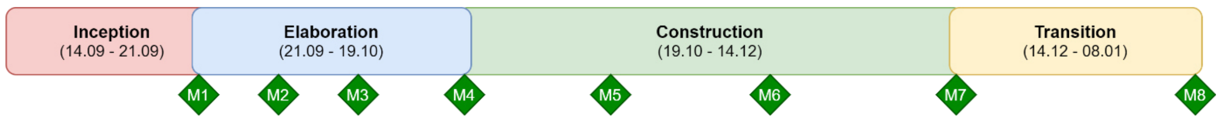


Figure 1: Project timetable
Source: own creation

5.2 Milestones

M#	Date	Description	Products
M1	21.09.2020	Inception	Project plan Requirement analysis Definition of scope Work environment setup - CI/CD pipeline setup
M2	05.10.2020	UI PoC	UI PoC - Definition of interfaces
M3	12.10.2020	Data access PoC	Data access PoC List of required data - Class diagram
M4	19.10.2020	Business Logic PoC	Business logic PoC Class diagram - SAD
M5	26.10.2020	App LQA	Implementation of LQA including tests - Packet Loss Map, Delay Map, Combined Map
M6	16.11.2020	App LSP Core Functionality	Implementation of LSP including tests - Link Saturation Map
M7	14.12.2020	App LSP	- Implementation of additional use cases
M8	08.01.2020	Delivery	- All delivery items uploaded to AVT

Table 3: Milestones

5.3 Phases / Iterations

The project will consist of the four phases called Inception, Elaboration, Construction and Transition. A phase consists of iterations. Each iteration lasts one week. At the end of each iteration the progress is reviewed at a meeting. It will be checked, if all planned tasks were completed and discuss the difficulties and problems that arose during the iteration. Subsequently, the next iteration is planned by deciding on which work packages need to be attended to next.

5.3.1 Inception

The inception phase lasts one week. In this phase the scope of the project will be defined, and the work environment will be set up.

5.3.2 Elaboration

The elaboration phase lasts four weeks. This phase contains the following:

- Determine how the required data from Jalapeño can be accessed and how it will be loaded into the SR-Apps.
- Create a prototype for the UI
- Create a prototype for the business logic and define all interfaces
- Create a class and package diagram to show the planned software architecture

5.3.3 Construction

During the construction phase, which will be taking eight weeks, the SR-Apps will be developed. All three Apps will be combined to a single application.

5.3.4 Transition

The last three weeks is reserved for the transition phase. This phase will contain final code clean up and the finalizing of all documentation.

5.4 Meetings

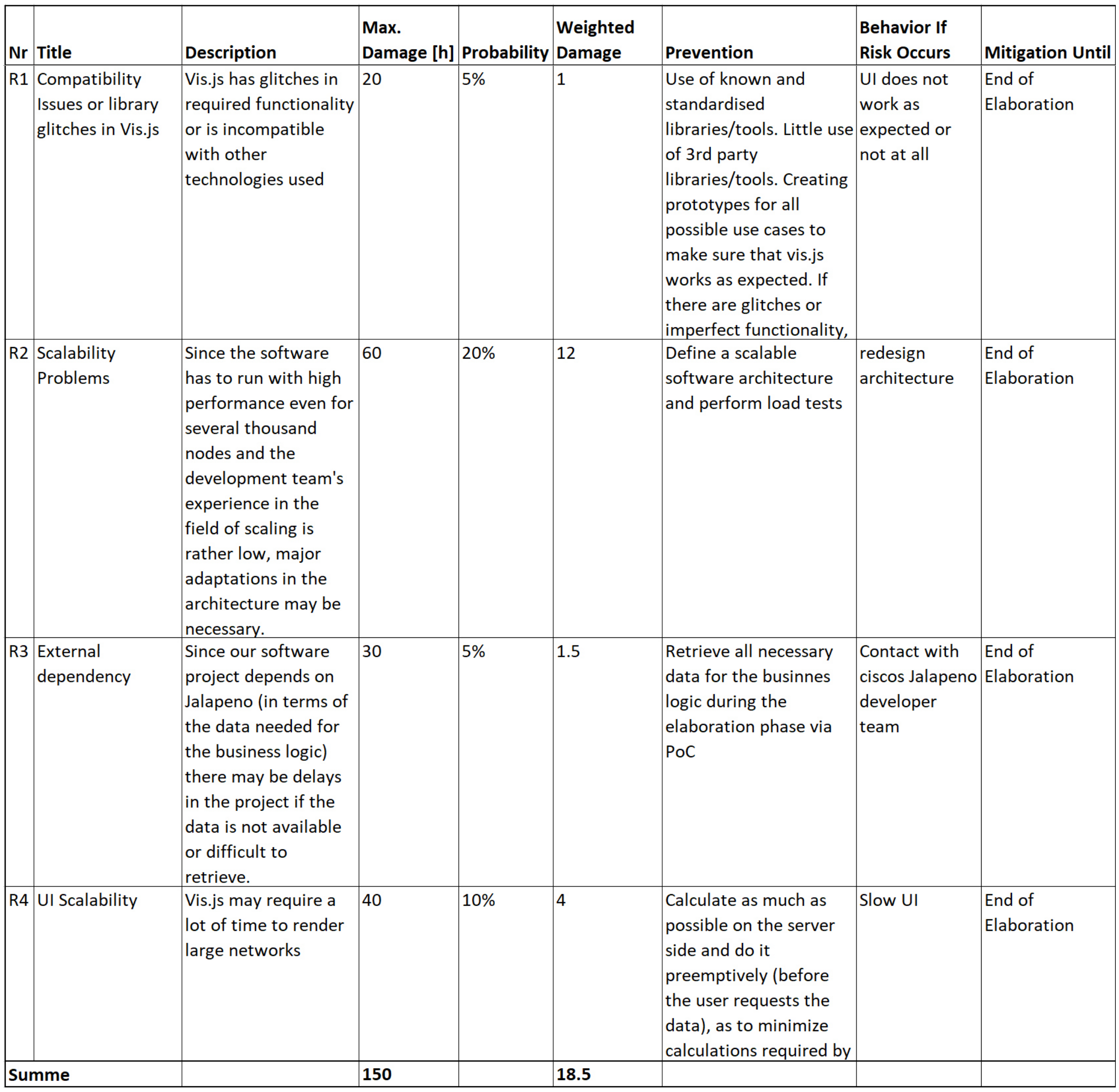
Day of Week	Time	Topic	Who	Where
Monday	09:00 – 10:00	Review and planning with supervisor	Dominique Illi Michel Bongard Laurent Metzger	HSR, Room 8.125
Monday	13:00 – 14:00	Sprint review, grooming & planning	Dominique Illi Michel Bongard	HSR, Room 1.258

Table 4: Meetings

6 Risk management

6.1 Risks

The risk assessment is shown in Figure 2.



Nr	Title	Description	Max. Damage [h]	Probability	Weighted Damage	Prevention	Behavior If Risk Occurs	Mitigation Until
R1	Compatibility Issues or library glitches in Vis.js	Vis.js has glitches in required functionality or is incompatible with other technologies used	20	5%	1	Use of known and standardised libraries/tools. Little use of 3rd party libraries/tools. Creating prototypes for all possible use cases to make sure that vis.js works as expected. If there are glitches or imperfect functionality,	UI does not work as expected or not at all	End of Elaboration
R2	Scalability Problems	Since the software has to run with high performance even for several thousand nodes and the development team's experience in the field of scaling is rather low, major adaptations in the architecture may be necessary.	60	20%	12	Define a scalable software architecture and perform load tests	redesign architecture	End of Elaboration
R3	External dependency	Since our software project depends on Jalapeno (in terms of the data needed for the business logic) there may be delays in the project if the data is not available or difficult to retrieve.	30	5%	1.5	Retrieve all necessary data for the businnes logic during the elaboration phase via PoC	Contact with ciscos Jalapeno developer team	End of Elaboration
R4	UI Scalability	Vis.js may require a lot of time to render large networks	40	10%	4	Calculate as much as possible on the server side and do it preemptively (before the user requests the data), as to minimize calculations required by	Slow UI	End of Elaboration
Summe			**150**		**18.5**			

Figure 2: Risk Assessment
Source: own creation

7 Work packages

All work packages are managed by Jira.

8 Infrastructure

8.1 General

Tool	Description
Computer	Every developer needs a computer for development.
Visual Studio	Every developer needs Visual Studio for development.
Atlassian Jira	A cloud-hosted instance of Jira is used to create and manage work packages and to log time.
Atlassian Bamboo	An instance of Bamboo is hosted on a private server to handle CI/CD.
GitHub	The repository of the source code as well as version control is managed by GitHub.
OneDrive	OneDrive will be used for collaborative editing of documents.

Table 5: General Infrastructure

8.2 CI/CD Pipeline

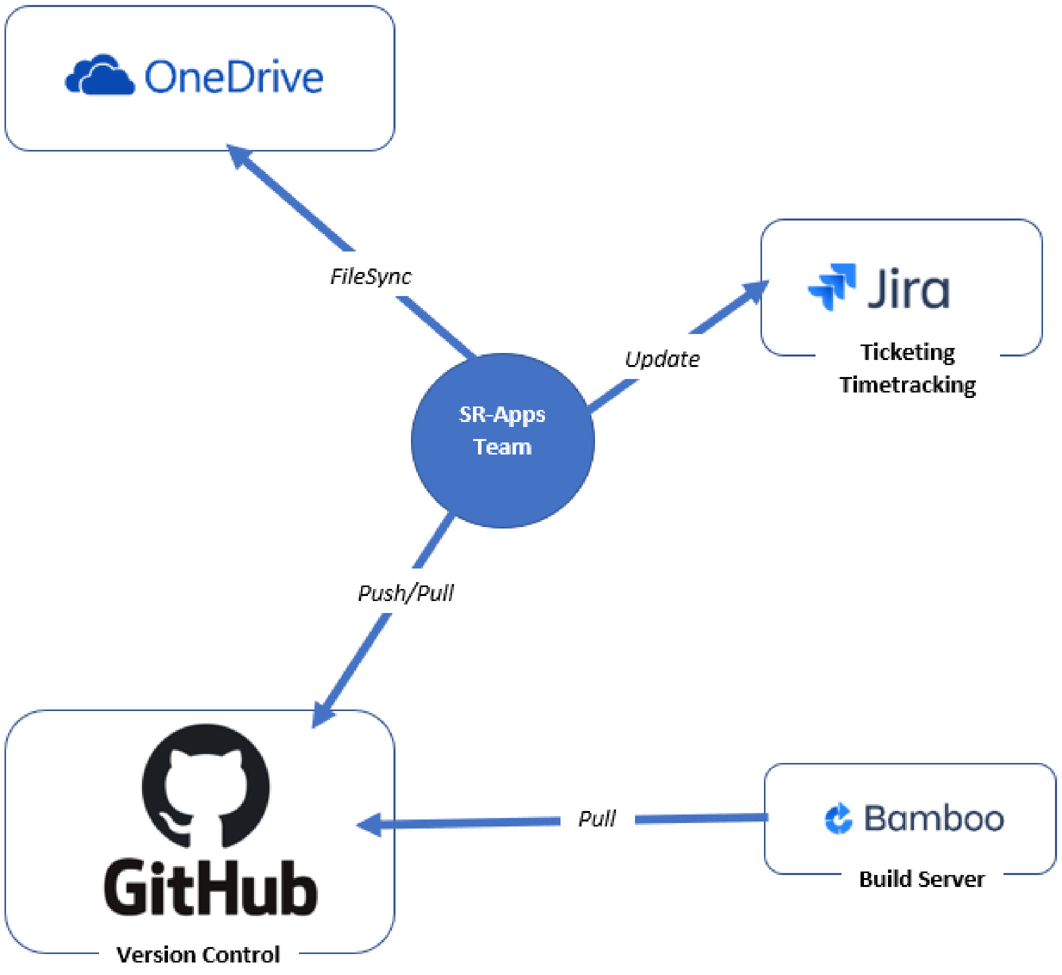


Figure 3: CI/CD Pipeline
Source: own creation

Figure 3 shows the CI/CD pipeline.

- **GitHub**: is used for version control. The source code is pushed to GitHub if all tests and the style checker are green.
- **Bamboo**: is the build server. The source code is built every night. With each build all tests are executed and reported.
- **Jira**: is used for time tracking and sprint planning.
- **OneDrive**: is used to exchange files and store the documentation.

9 Quality measures

9.1 Documentation

The project documentation is stored on OneDrive which allows for collaborative editing.

9.2 Project management

The work packages are created and managed on Jira. This tool is also used to log the time spent on each package. Every week a grooming meeting is held, where packages are defined, estimated and prioritized.

9.3 Development

The code is located on a repository on GitHub.

10 Illustration index

11 Glossary

Term	Description
CI/CD	Continuous Integration / Continuous Deployment
SR	Segment Routing

Table 6: Glossary

SR-App Analytics

E-3 – Software Architecture Document

Authors: Dominique Illi, Michel Bongard

Fall Term 2020

1 Content

2 Feature implementation

This chapter describes the scope of the implemented features.

2.1 Feature 1 - Link Grading

The feature Link Grading allows the user to view the quality and importance of links. Through UI controls, the user can choose between the four different views (maps) listed in Table 1.

Map	Description
Packet Loss	Highlights the map according to the packet loss values of the links. Higher is worse.
Link Delay	Highlights the map according to how frequently the delay values of the links change. The more frequently the delay changes, the worse.
Link Saturation	Highlights the map according to the saturation of the links (how much of the available bandwidth is used). The more saturated, the worse.
Link Quality	Combines the maps “Packet Loss” and “Link Delay”, highlighting the individual links according to the worse value of the two.

Table 1: Maps

The highlighting of the map is achieved by coloring the links according to the code shown in Figure 1.

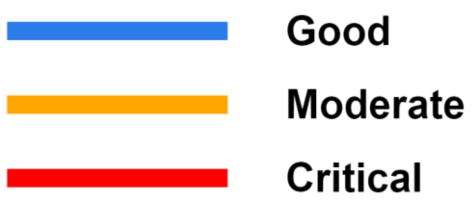


Figure 1: Map color codes
Source: own creation

The importance of a link is determined by how many shortest paths are using it. The more shortest paths use a specific link, the more important it is considered to be. This is because, if a link fails that is used by many shortest paths, its absence has a higher impact on the topology than one that is used by only a few shortest paths.

Figure 2 shows an example of a Packet Loss map. Figure 3 shows the same topology but with the Link Saturation map instead.

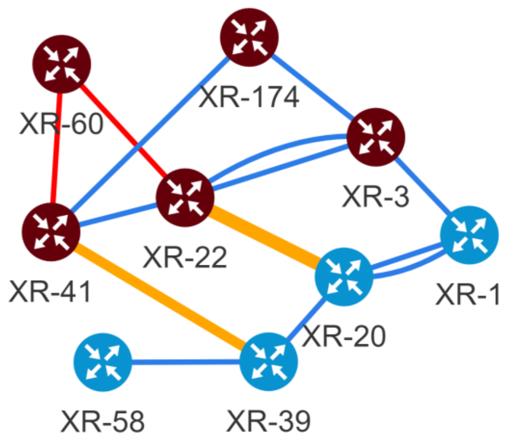


Figure 2: Packet Loss map
Source: own creation

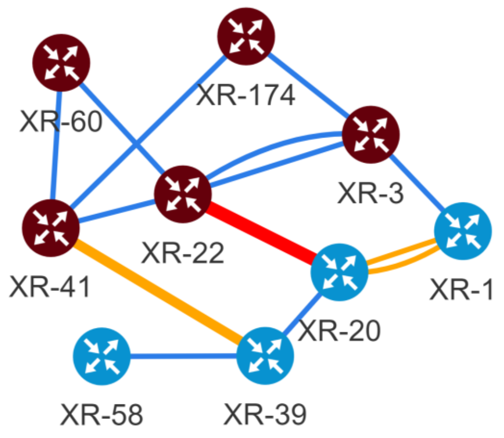


Figure 3: Link Saturation map
Source: own creation

2.2 Feature 2 – Link Saturation Prediction

The application allows the user to make changes to the topology through the UI, such as removing and adding links or nodes. This, of course, may impact certain shortest paths and therefore also change the distribution of traffic in the network.

To allow the user to simulate such changes to the topology, all shortest paths are recalculated after any alteration to the network and the traffic distribution is recalculated.
This is possible thanks to the traffic data provided by the SR protocol. Through its traffic matrix, the amount of traffic between any two nodes can be extracted.

For a more detailed description on how the traffic is redistributed, see Chapter 4.2.

3 Use Case Coverage

This chapter describes to what extent the use cases are covered (see Table 2) and which architectural components they use. Due to similarity in some use cases, not all are described in detail.

Use Case	Status	
01 – Load Map	Fully implemented	✓
02 – Zoom In	Fully implemented	✓
03 – Zoom Out	Fully implemented	✓
04 – Show Details	Fully implemented	✓
05 – Remove Link	Fully implemented	✓
06 – Add Link	Fully implemented	✓
07 – Remove Node	Fully implemented	✓
08 – Add Node	Fully implemented	✓
09 – Change Link Attributes	Fully implemented	✓
10 – Load Latest Topology	Fully implemented	✓
11 – Manually Poll Topology	Fully implemented	✓

Table 2: Use case coverage

3.1 Use case 01 – Load Map

All maps are created during the session initialization (when a new user connects to the server) and continuously updated when the user makes changes to the topology.

The use case is triggered by switching the map from the drop-down menu in the UI (see Figure 4).

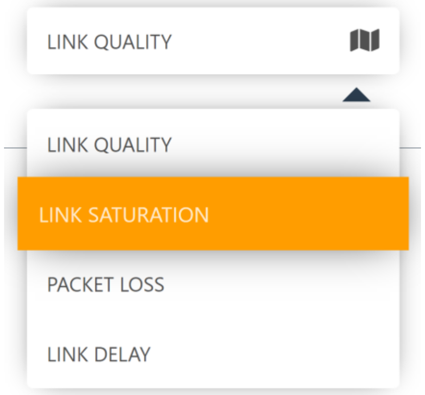


Figure 4: Map selector
Source: own creation

Figure 5 shows the sequence which is triggered by the user selection.

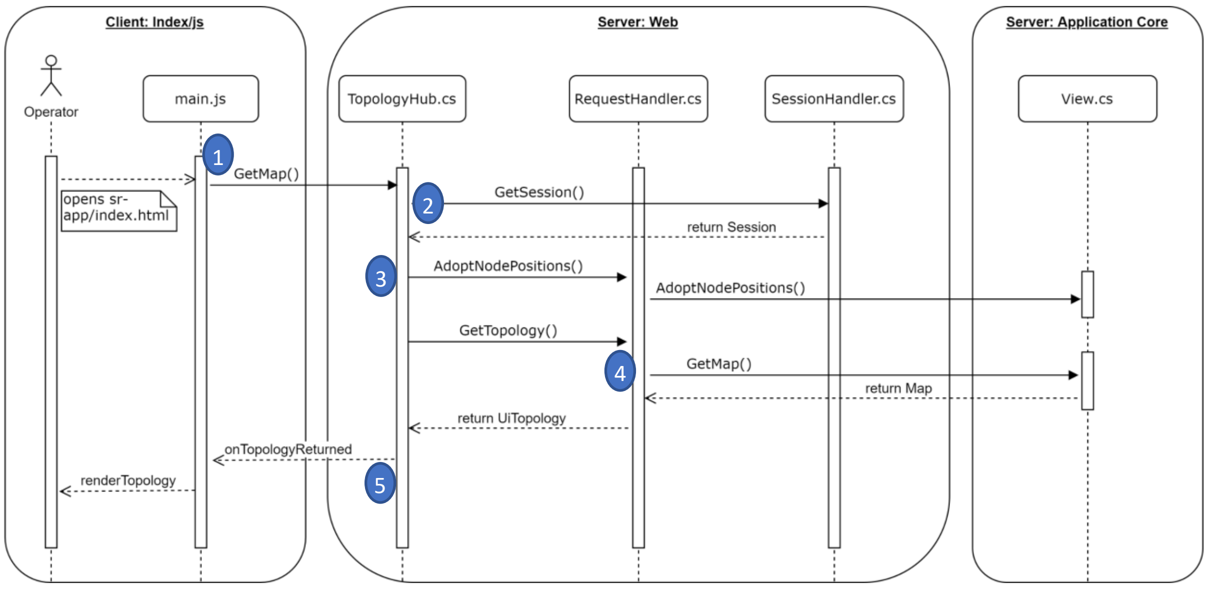


Figure 5: Sequence diagram – Use case 01
Source: own creation

1. Using SignalR [1], the client invokes the method GetMap() in the TopologyHub on the server.
2. The TopologyHub first retrieves the session of the user which contains the session specific topology.
3. Because the user can move nodes around on the map, the node coordinates may have changed. To prevent resetting the positions of the nodes after every API call, all node coordinates (such as they are positioned by the user) need to be sent along with the API call and adopted by the server.
4. Retrieves the requested Map and then converts the Map into a UiTopology which will be sent back to the client and rendered by Vis.js.
5. Using SignalR, the server invokes the method onTopologyReturned() on the client.

For more details on client-server communication see Chapter 6.2.

3.2 Use case 02 – Zoom in

By default, only regions (clusters) are shown (see Figure 6).

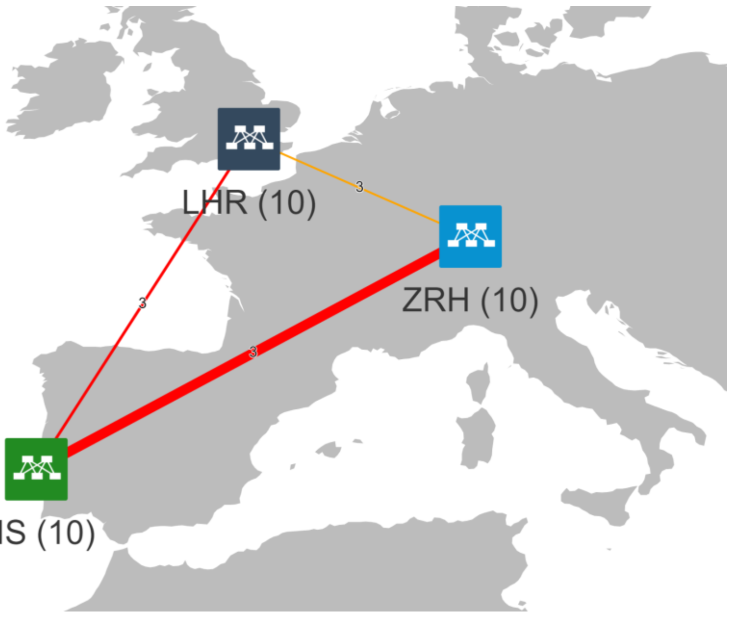


Figure 6: Default view - showing only regions
Source: own creation

To see the routers in a region the cluster can be expanded by simply double-clicking on a region. This triggers the sequence shown in Figure 8, which removes the cluster node from the map and instead places all routers in a circular fashion around the region center. This results in the map shown in Figure 7.

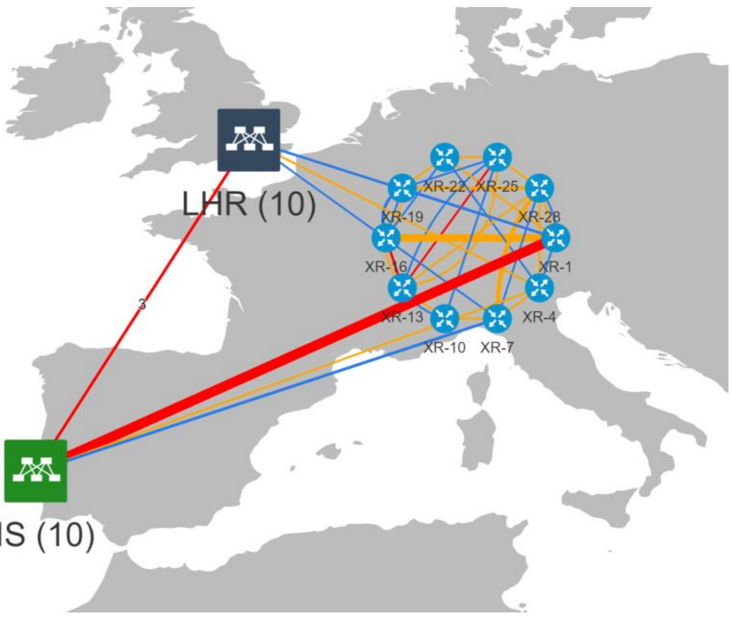


Figure 7: Expanded region ZRH
Source: own creation

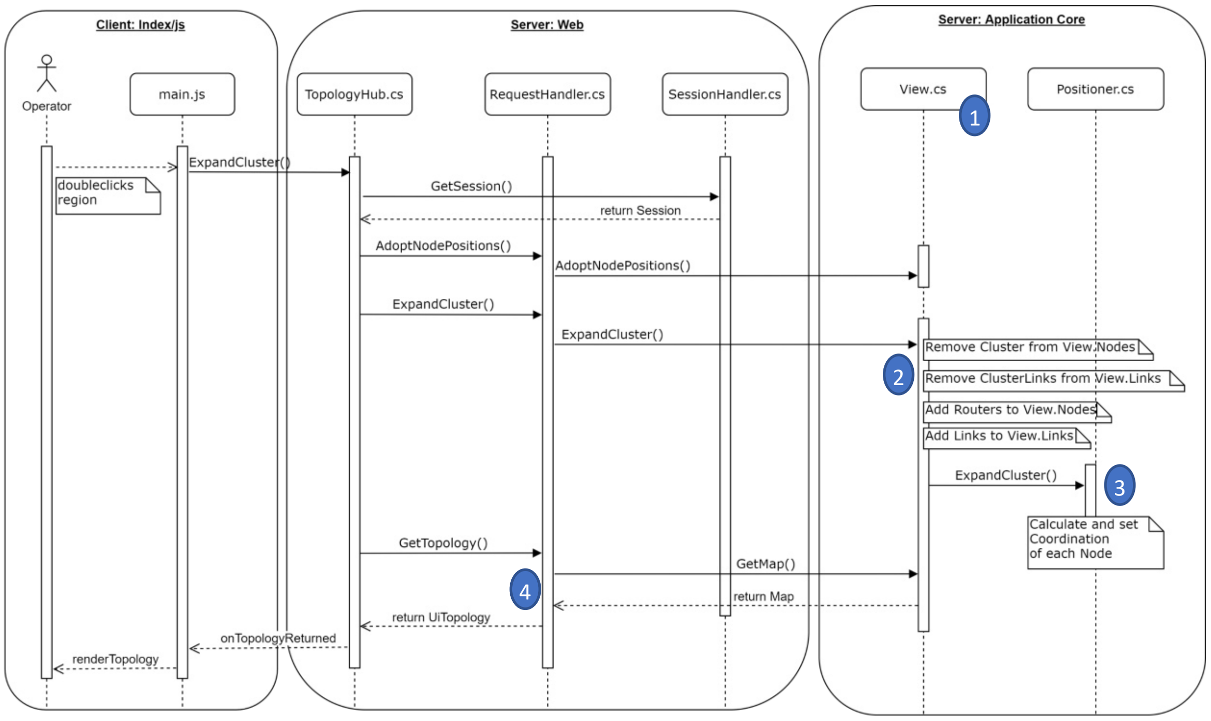


Figure 8: Sequence diagram – Use case 02
Source: own creation

The client-server communication and node adoption process is analog to the use case 01 in Chapter 3.1.

1. The View class keeps track of all nodes and links which are visible and get rendered by the UI.
2. The cluster and cluster links are removed from the View. In their stead, the routers and links from that cluster are added to the lists.
3. The Positioner is called to calculate the coordinates of the now visible routers.
4. Finally, the updated map is retrieved and sent back to the client as a UiTopology analog to Chapter 3.1.

3.3 Use case 03 – Zoom Out

The logic behind this use case is nearly identical to use case 02 (zoom in) in Chapter 3.2.
The user can collapse a region by double-clicking on a router. In this case the cluster and cluster links are added to the View and the actual routers and links are removed from the View. The view is then converted into a UI Topology and sent to the client (see Figure 9).

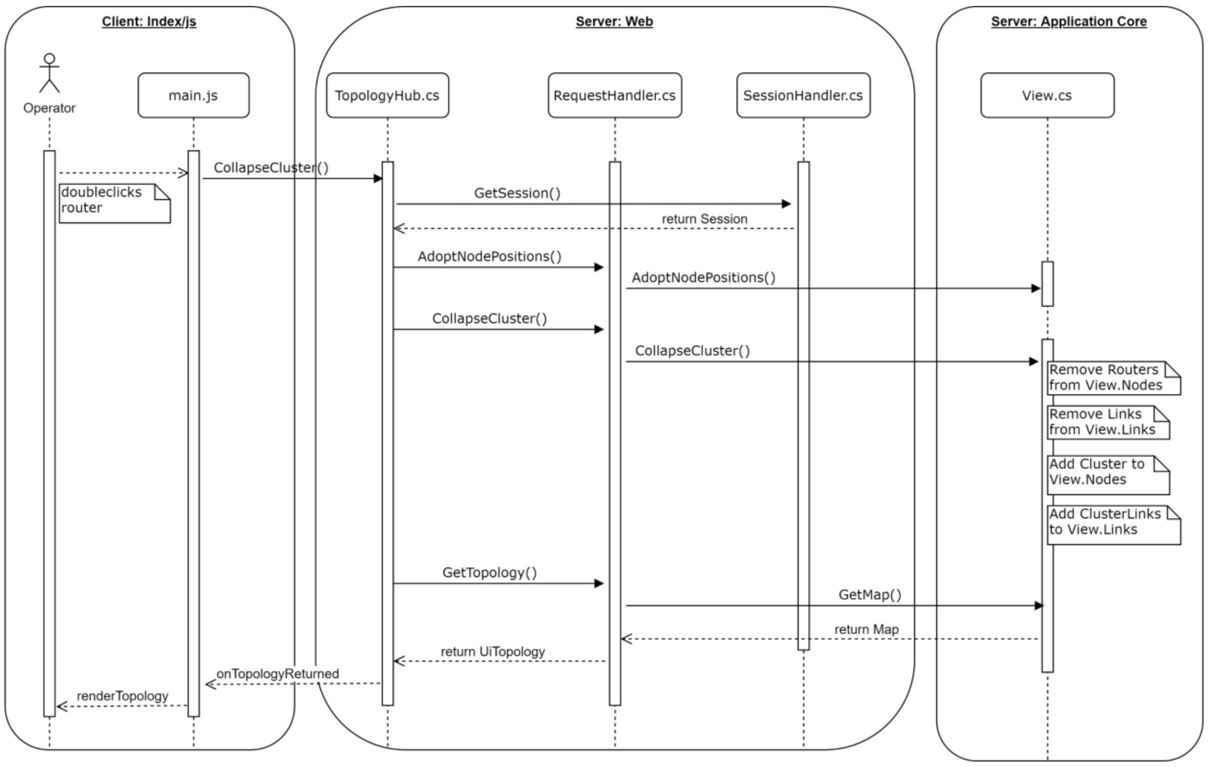


Figure 9: Sequence diagram – Use case 03
Source: own creation

3.4 Use case 05 – Remove Link

The user can remove a link by selecting it in the UI and then clicking the “Delete Link” button in the map header (see Figure 10).

EXPAND
NODE
LINK
NODE
LINK
COLLAPSE

Figure 10: Map header
Source: own creation

Figure 11 shows the Link Saturation map of the original topology and Figure 12 shows the Link Saturation map after the link between XR1 and XR2 (the red one) has been deleted. Because the traffic was redistributed after the change to the topology (see Chapter 4.2 for more information), the map is now highlighted differently, showing the link saturation prediction.

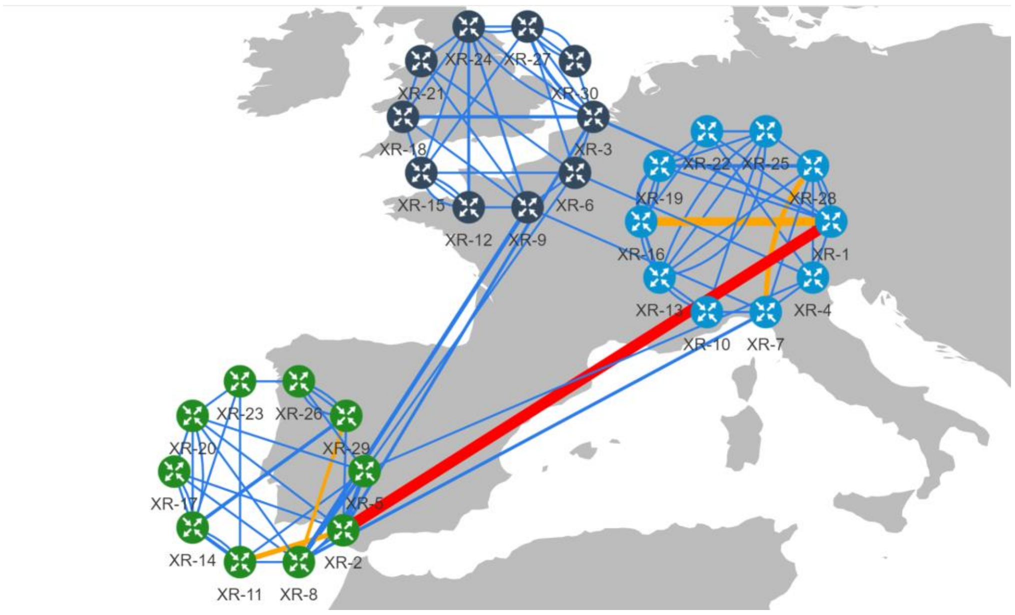


Figure 11: Original topology
Source: own creation

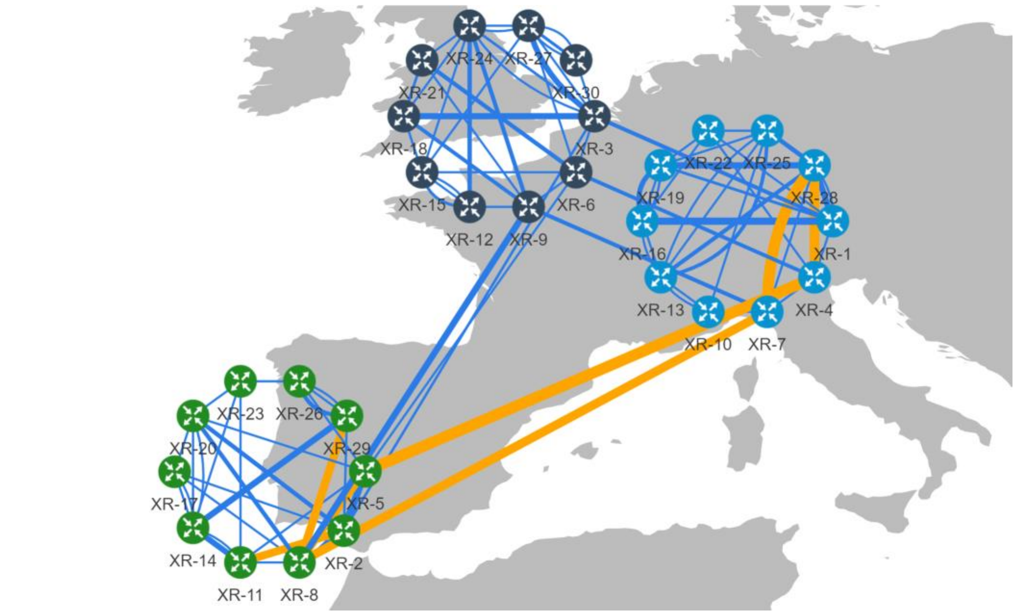


Figure 12: Topology after link XR1-XR2 is removed
Source: own creation

Figure 13 shows the sequence triggered when the user clicks the “Delete Link” button.

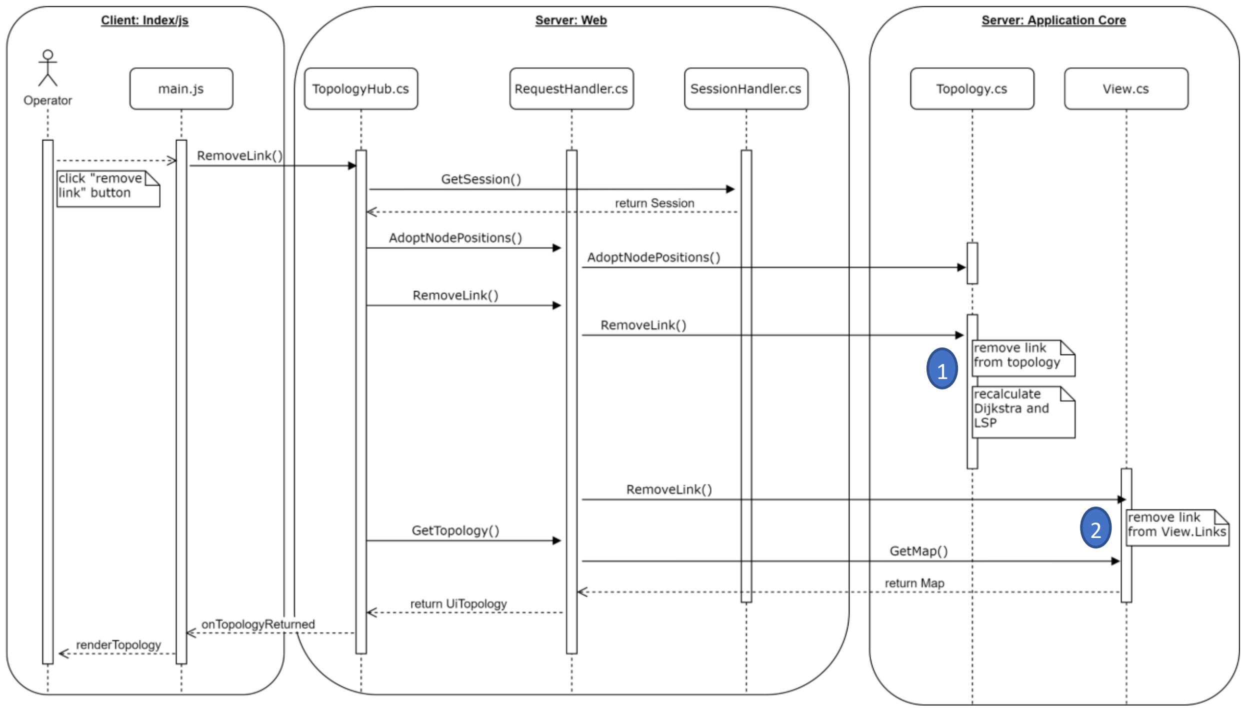


Figure 13: Sequence diagram – Use case 05
Source: own creation

1. The link is removed from the session specific topology and a recalculation of the shortest paths and the link saturation is triggered.
2. The link is removed from the View class and the grading of links and nodes on each map is recalculated to represent the changed topology.

3.5 Use case 06 – Add Link

The user can add a link to the network by clicking the “Add Link” button in the map header and then clicking and dragging the new link from the start to the target router. Adding a link triggers the sequence shown in Figure 14.

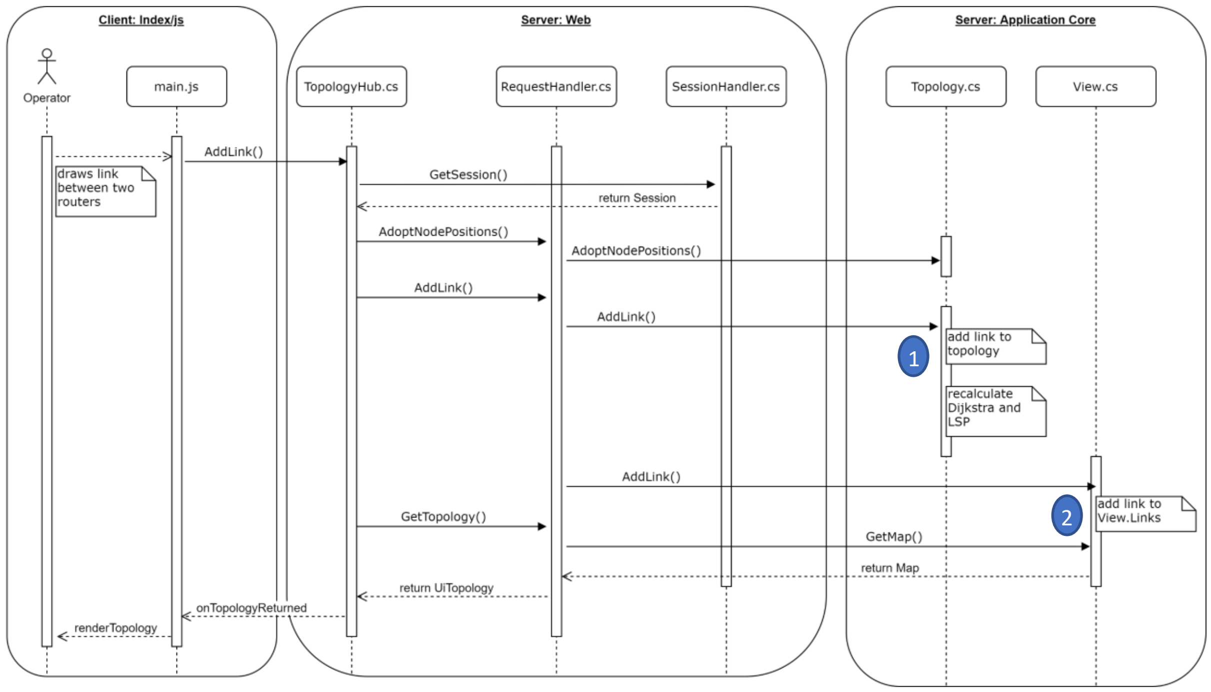


Figure 14: Sequence diagram – Use case 06
Source: own creation

1. The link is added to the session specific topology and a recalculation of the shortest paths and the link saturation is triggered.
2. The link is added to the View class and the grading of links and nodes on each map is recalculated to represent the changed topology.

3.6 Use case 07 – Remove Node

The user can delete a router from the network by selecting it first and then clicking the “Delete Node” button in the map header. Figure 15 shows the sequence that is triggered by this action.

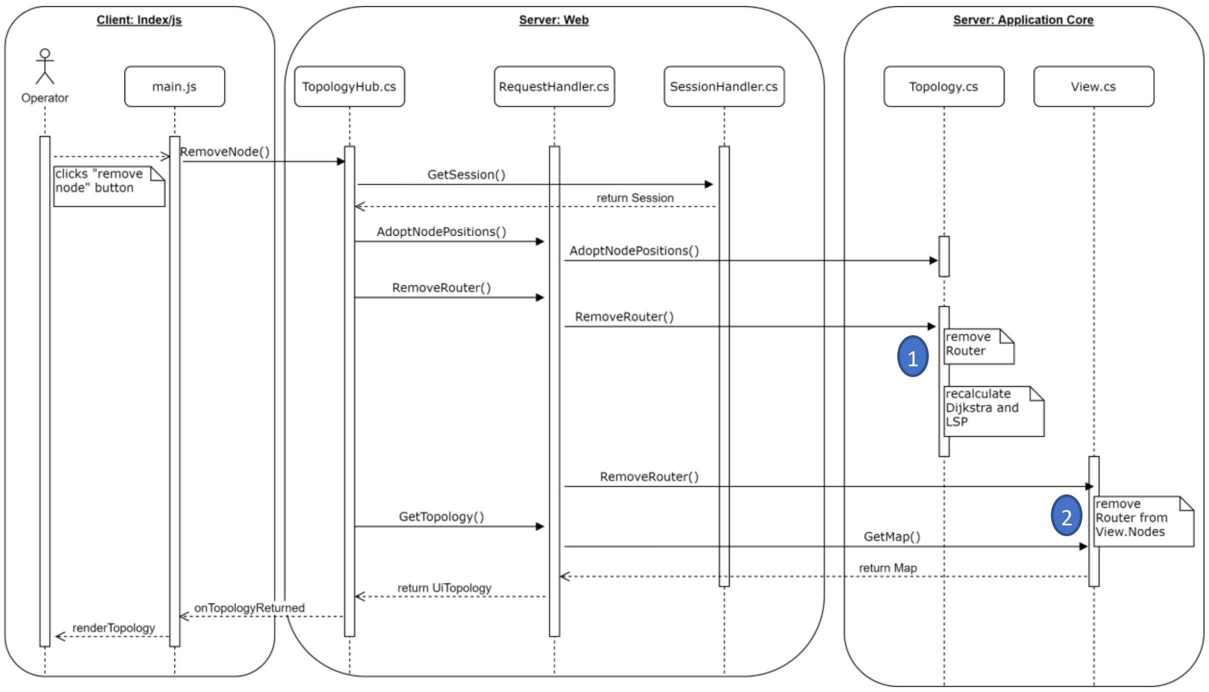
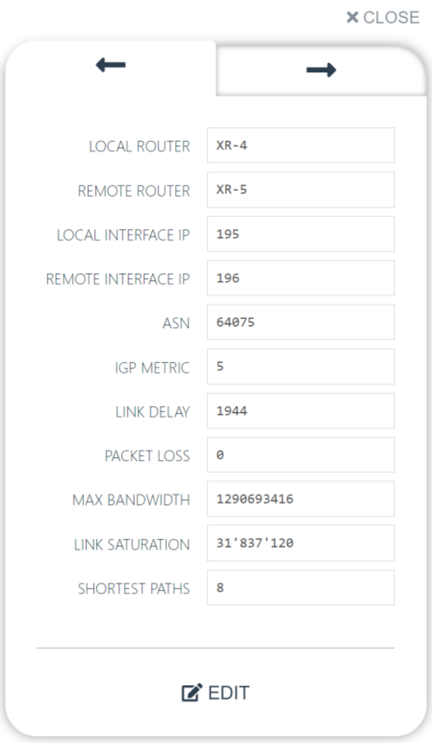


Figure 15: Sequence diagram – Use case 07
Source: own creation

1. The router is removed from the session specific topology and a recalculation of the shortest paths and the link saturation is triggered.
2. The router is removed from the View class and the grading of links and nodes on each map is recalculated to represent the changed topology.

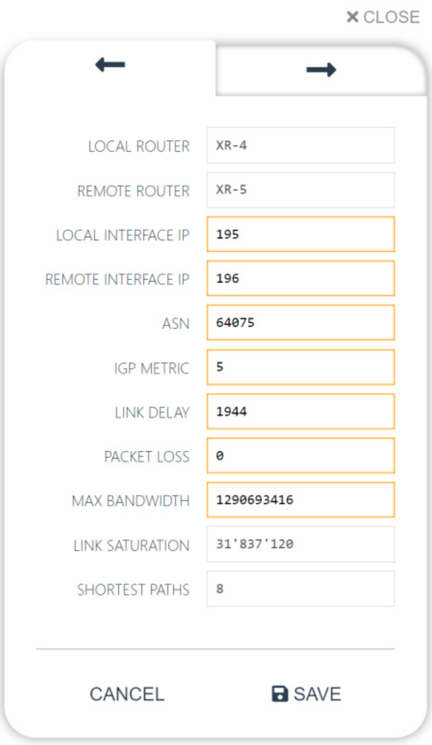
3.7 Use case 09 – Change Link Attributes

The user can change link attributes by selecting a link and then clicking the “Edit” button in the details view (see Figure 16) to enter the “Edit” mode (see Figure 17). Fields that are eligible for editing are highlighted with an orange border. The two tabs marked with arrows allow switching between the two unidirectional links that the link consists of.



LOCAL ROUTER	XR-4
REMOTE ROUTER	XR-5
LOCAL INTERFACE IP	195
REMOTE INTERFACE IP	196
ASN	64075
IGP METRIC	5
LINK DELAY	1944
PACKET LOSS	0
MAX BANDWIDTH	1290693416
LINK SATURATION	31'837'120
SHORTEST PATHS	8

Figure 16: Link details view
Source: own creation



LOCAL ROUTER	XR-4
REMOTE ROUTER	XR-5
LOCAL INTERFACE IP	195
REMOTE INTERFACE IP	196
ASN	64075
IGP METRIC	5
LINK DELAY	1944
PACKET LOSS	0
MAX BANDWIDTH	1290693416
LINK SATURATION	31'837'120
SHORTEST PATHS	8

Figure 17: Link details view in "Edit" mode
Source: own creation

Saving the changes triggers the sequence shown in Figure 18.

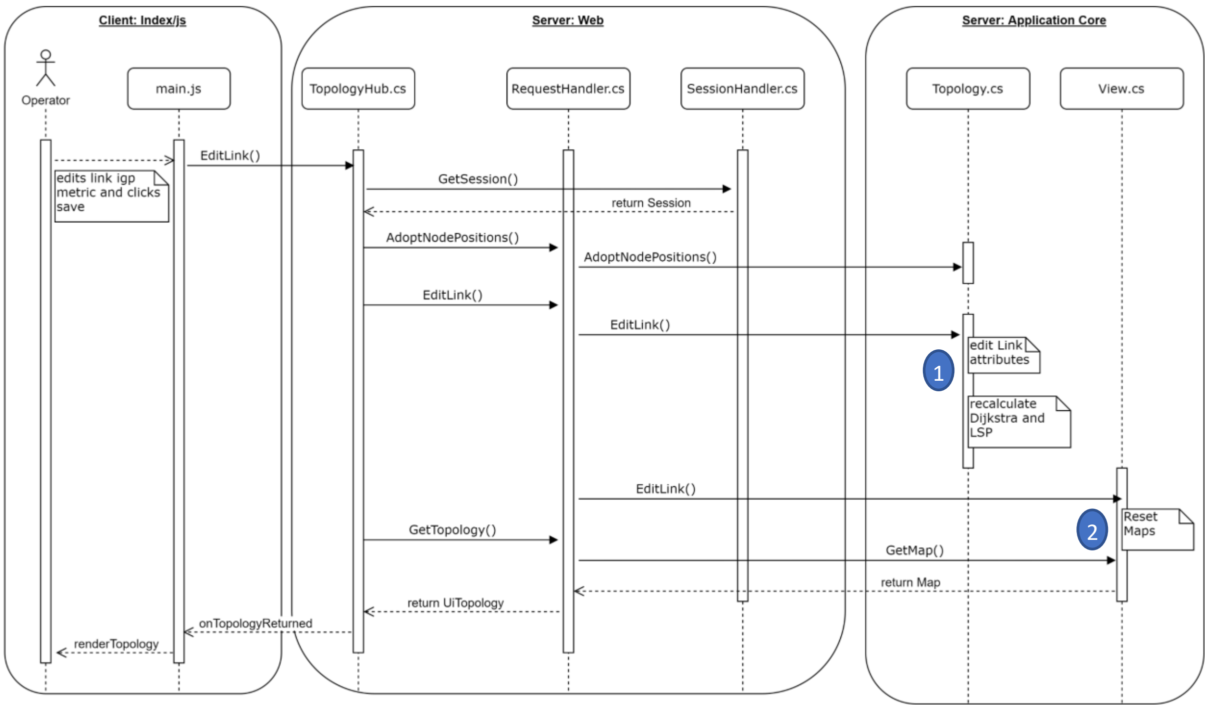
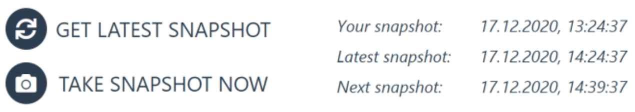


Figure 18: Sequence diagram – Use case 09
Source: own creation

1. The updated link values are transferred to the session specific topology and a recalculation of the shortest paths and the link saturation is triggered.
2. The View class triggers the re-grading of links and nodes on each map to represent the changed topology.

3.8 Use case 11 – Manually Poll Topology

The user can force the server to immediately poll the latest data from the database by clicking the "Take New Snapshot" button (see Figure 19). This resets the periodic polling task running in the background on the server, which periodically polls data from the database every 15 minutes.



Your snapshot: *17.12.2020, 13:24:37*
Latest snapshot: *17.12.2020, 14:24:37*
Next snapshot: *17.12.2020, 14:39:37*

Figure 19: Snapshot buttons
Source: own creation

Clicking this button triggers the sequence shown in Figure 20.

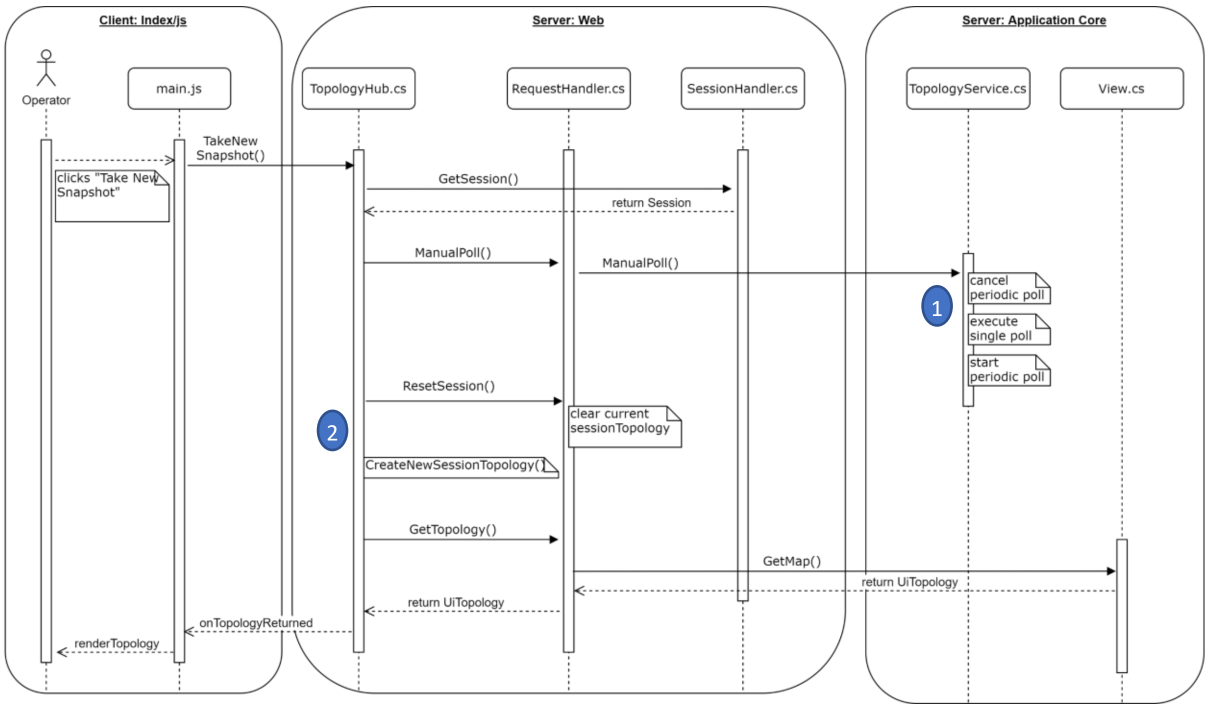


Figure 20: Sequence diagram – Use case 11
Source: own creation

1. The task which periodically polls data from the database every 15 minutes is cancelled. A single poll is then executed, after which the periodic polling task is restarted
2. The TopologyHub then resets the session, loading the latest topology snapshot to the session.

4 Core Logic

This chapter provides short descriptions of the implemented algorithms and concepts.

4.1 Shortest Path Calculation

4.1.1 Introduction

Both to be able to display the importance of links for the link quality assessment (Chapter 2.1) and to calculate the saturation of the links (Chapter 2.2) all shortest paths of the entire topology need to be calculated.

For this purpose, a modified version of Dijkstra is implemented, using the **FastPriorityQueue** from the GitHub user BlueRaja [2], which is a C# implementation of a heap-based priority queue that has been optimized for speed when used for path-finding algorithms.

The shortest path calculation is discussed in more detail in the document “Technical Report”.

4.1.2 Modifications

The two modifications to the classic Dijkstra algorithm are the following:

1. From the resulting tree structure calculated by Dijkstra, each node remembers its parent. This allows the reconstruction of the shortest paths.
2. The classic Dijkstra algorithm disregards equal cost shortest paths. Through an additional if-clause, these paths can be taken into consideration as well.

4.1.3 Time & Space Complexity

The calculation of all shortest paths of a topology has the following **time complexity**:

$$O(N^2 * \log(N))$$

This results in the real-life performance measurements shown in Table 3. These tests were run on an Intel i7-6700k (4GHz Quad-Core, Multithreaded) and 32GB of RAM.

Nodes	Edges (Bidirectional)	Duration (best of 5)
500	1’500	0.10 seconds
1’000	3’000	0.52 seconds
1’000	10’000	1.10 seconds
3’000	9’000	8.98 seconds

Table 3: Results of testing custom implementation for shortest path calculation

Storing all shortest paths of a topology has the following **space complexity**:

$$O(N^2)$$

Because in order to store the shortest paths, only references to the parents need to be saved, the space requirement of any topology is simply $N^2 * 64bit$ on a 64-bit system, which results in roughly 8MB for 1'000 Nodes.

4.2 Link Saturation Calculation

To accurately predict the behavior of a network in case of a change in the topology (such as a link failure), the saturation of every single link in the network needs to be recalculated.

For this purpose, SR-Apps uses the traffic data provided by the segment routing protocol, which indicate the amount of traffic flowing between any two SR-routers in the network.

4.2.1 Example

As an example, consider the topology in Figure 21. The numbers on the links represent the IGP metric used for the shortest path calculation.

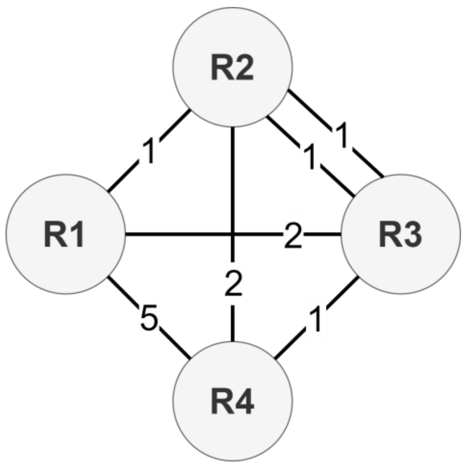


Figure 21: Sample Topology
Source: own creation

For this example, only the traffic from router R4 to router R1 is distributed. Figure 22 shows all four shortest paths from R4 to R1, which are highlighted as red, green, blue and orange.

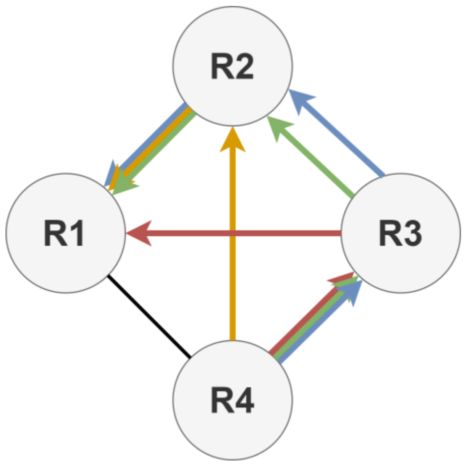


Figure 22: All shortest paths from R4 to R1
Source: own creation

Assuming that the traffic flowing from R4 to R1 is 1.2 Gbps, the traffic would be distributed as shown in Figure 23. The numbers on the links show the distributed traffic in Mbps.

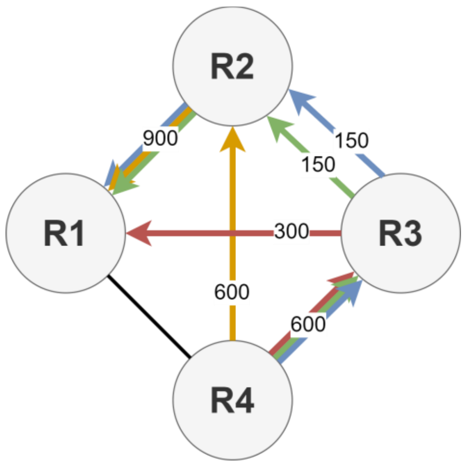


Figure 23: Distributed traffic from R4 to R1
Source: own creation

Traffic is always split evenly among all possible next hops. Therefore, the same amount of traffic flows from R4 to R2 as it does from R4 to R3, even though there are three shortest paths going from R4 to R3.

Traffic is also split evenly again among all equal cost links between two routers. Therefore, the 300 Mbps flowing from R3 to R2 is split in two, half flowing in one link, the other half in the other link.

To recalculate the link saturation of the entire topology, the process illustrated in this chapter needs to be repeated for every possible router combination, so it needs to be done N^2 times.

4.2.2 Time Complexity

The calculation of the link saturation of a topology has the following **time complexity**:

$$O(N^2 * \log(N))$$

This results in the real-life performance measurements shown in Table 4. These tests were run on an Intel i7-6700k (4GHz Quad-Core, Multithreaded) and 32GB of RAM.

Nodes	Edges (Bidirectional)	Duration (best of 5)
500	1’500	0.21 seconds
1’000	3’000	0.97 seconds
1’000	10’000	1.19 seconds
3’000	9’000	10.77 seconds

Table 4: Results of testing implementation for link saturation calculation

4.3 Clustering

4.3.1 Introduction

Because for large topologies the flood of information for the user can be overwhelming, a way to cluster nodes is implemented.
The current version of the application only implements a single level of clusters, which are the regions, since this has been determined to be sufficient for topologies up to 1’000 nodes. For larger topologies however, multilevel clustering may need to be considered. This is discussed in the document “Technical Report”.

4.3.2 Approach

The clusters are created by grouping the routers according to their regions (IATA airport codes).

The individual links between regions are combined to a single cluster link, which gets assigned the same grade (importance metric) as the link with the highest grade it contains.

4.4 Positioner

To provide better orientation for the user, the topology in the UI is drawn on top of a static image of a world map.

The positioner is responsible for placing the regions on the correct coordinates, which are predefined in advance. The regions then stay on a fixed place on the map, even when zooming in and out.
The positioner also calculates the coordinates of routers, when a region is expanded. The routers are positioned in a circular fashion around the center of the region.

5 System overview

The project consists of two software systems. One system is called "Jalapeño" and is provided by the industry partner. The second system, called SR-App, is developed for this bachelor thesis.

The SR-App system relies on Jalapeño to get data on the monitored SR topology. The network information is sent by the routers using BGP Monitoring Protocol (BMP) and Streaming Telemetry to Jalapeño. Jalapeño uses Kafka to aggregate the received data and stores it in two databases (InfluxDB for performance data and ArangoDB for topology data). The SR applications gets the required data directly from ArangoDB. The data can then be used to perform calculations and provide the results to the user. The data flow is shown in Figure 24.

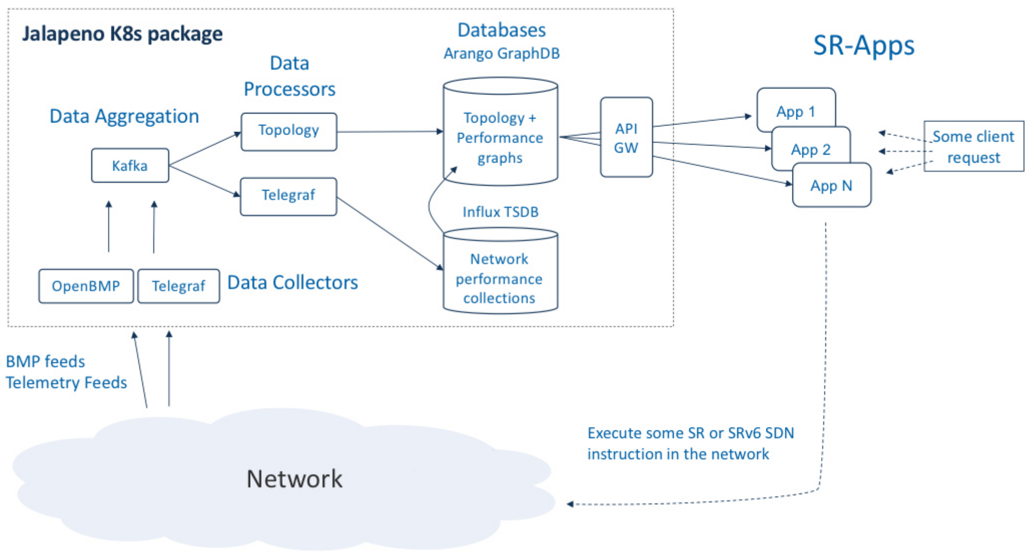


Figure 24: Data flow Jalapeño and SR-App
Source: Private GitHub repository of Jalapeño [3]

Because the current version of Jalapeño does not provide all information required to implement the defined use cases (such as the SR traffic matrix) and some minor problems with the router to Jalapeño communication occurred, no real data is used for the SR-App. Instead, all topology data is mocked and loaded to ArangoDB manually. By using the mocked database new data attributes can be easily added and topologies of any size and complexity can be generated.

The use of a mocked database has no impact on the software development and software architecture because the SR-App only communicates with the ArangoDB.

5.1 System Context Diagram

The System Context as shown in Figure 25 illustrates how the user (SR-Topology operator) and the software systems SR-App and Jalapeño are interconnected.

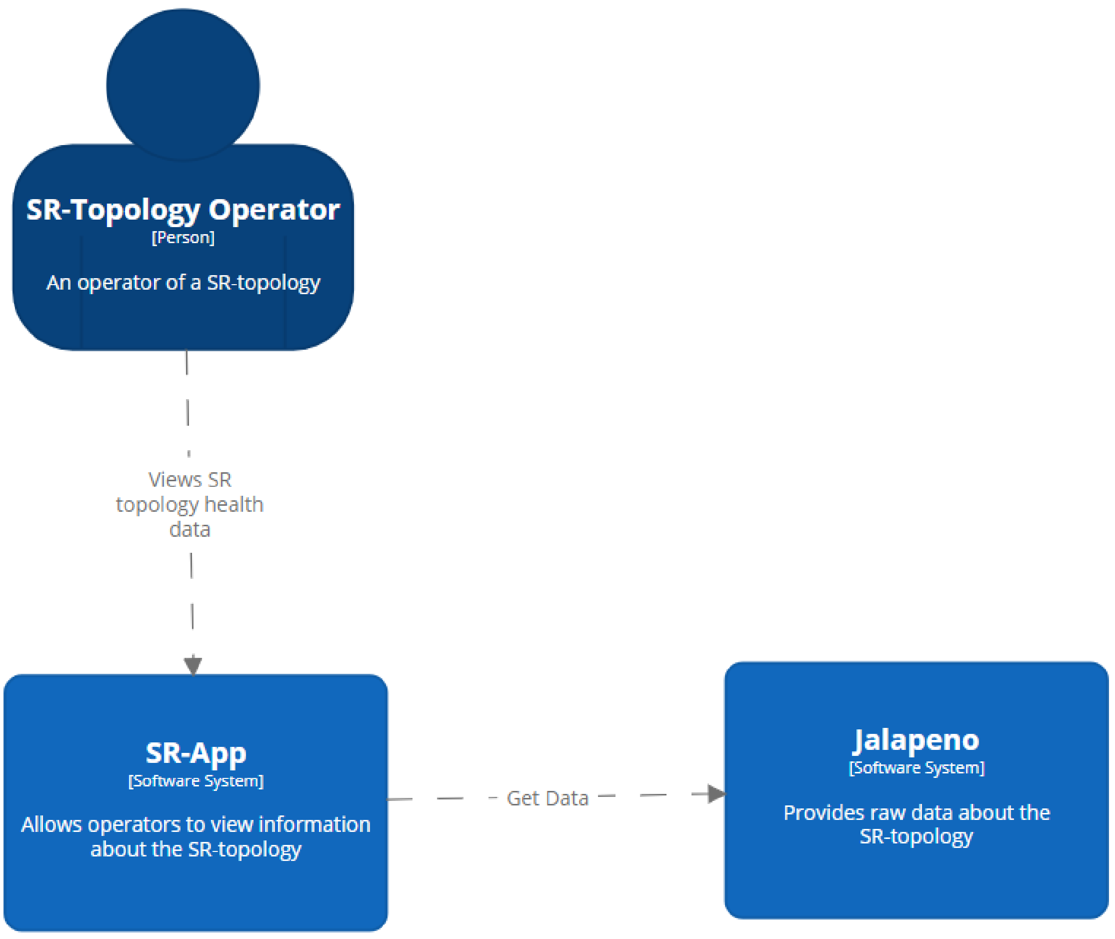

SR-Topology Operator
[Person]
An operator of a SR-topology
Views SR topology health data
SR-App
[Software System]
Allows operators to view information about the SR-topology
Get Data
Jalapeno
[Software System]
Provides raw data about the SR-topology

Figure 25: System Context Diagram
Source: own creation

5.2 Container Diagram

Figure 26 shows the container diagram of the application.

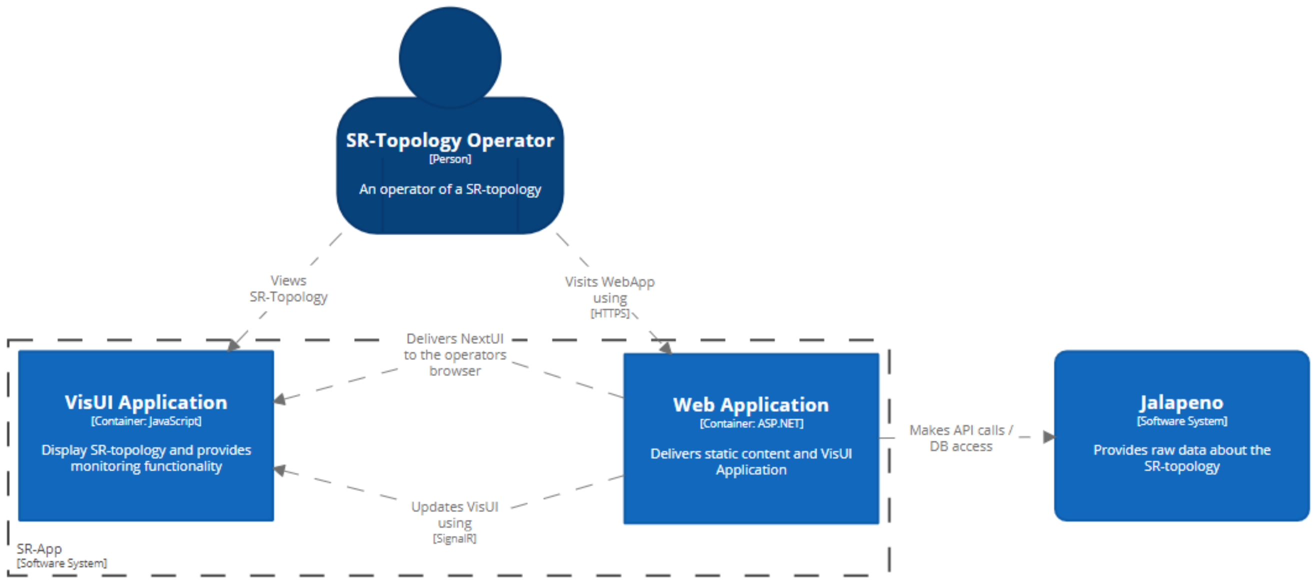

SR-Topology Operator
[Person]
An operator of a SR-topology
Views SR-Topology
Visits WebApp using [HTTPS]
VisUI Application
[Container: JavaScript]
Display SR-topology and provides monitoring functionality
Delivers NextUI to the operators browser
Updates VisUI using [SignalR]
Web Application
[Container: ASP.NET]
Delivers static content and VisUI Application
Makes API calls / DB access
Jalapeno
[Software System]
Provides raw data about the SR-topology
SR-App
[Software System]

Figure 26: Container Diagram
Source: own creation

6 Logical Architecture

This section documents the logical architecture and design decisions made during the project.

6.1 Core Architecture

The SR-App follows a classic client-server architecture. All calculations are made in the backend and published to the clients using a web socket. The client itself is only used to display the data but does not perform any calculations on its own, except for some visual tweaks through JavaScript. As foundation an ASP.NET Core 3.1 web application is used. The application follows a monolithic architecture. It was decided to use a monolithic architecture instead of a microservice architecture because it is a lot easier to test and debug and easier to deploy.

6.2 Client-Server Communication

The client-server communication is display in Figure 27. For communication, a web socket is used. It was decided not to use a REST API but a WebSocket, because there is much more data sent by the server to the client than the other way round. Data can be generated at any point during the lifecycle of the application and needs to be sent to the client without any prior requests made by the client. ASP.NET Core offers an open-source library called SignalR which was used to push content to clients instantly. SignalR supports real-time data transmission and requires JavaScript on the client to establish the WebSocket connection. This allows the client to call C# backend methods and the server to invoke JavaScript methods on the client.

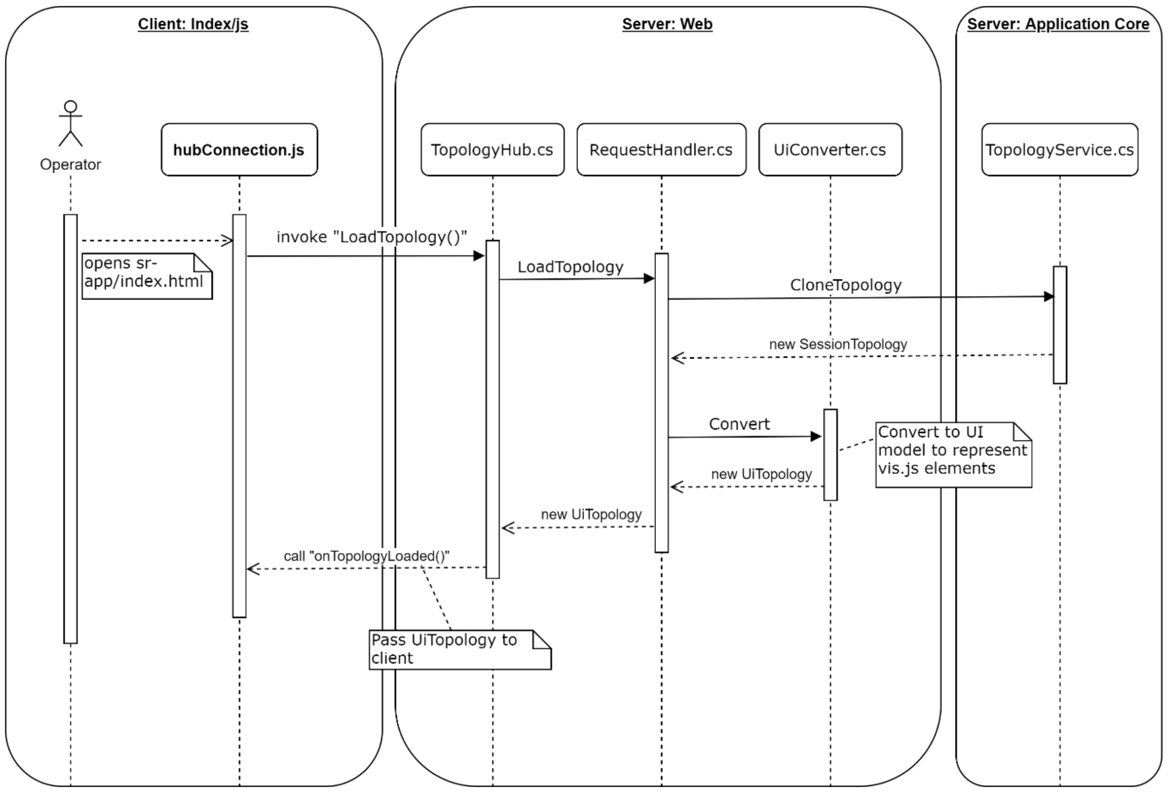


Figure 27: sequence diagram client-server communication
Source: own creation

6.3 Architectural overview

SR-App is structured into three layers (see Table 5) and multiple packages per layer.

Layer	Description
Web	For UI components and the communication to the client. Depends on the ApplicationCore layer.
ApplicationCore	Contains business logic and functionality. Depends on the Infrastructure layer.
Infrastructure	Performs data access to the ArangoDB.

Table 5: Application Layers

An overview of all layers and packages is given by Figure 28.

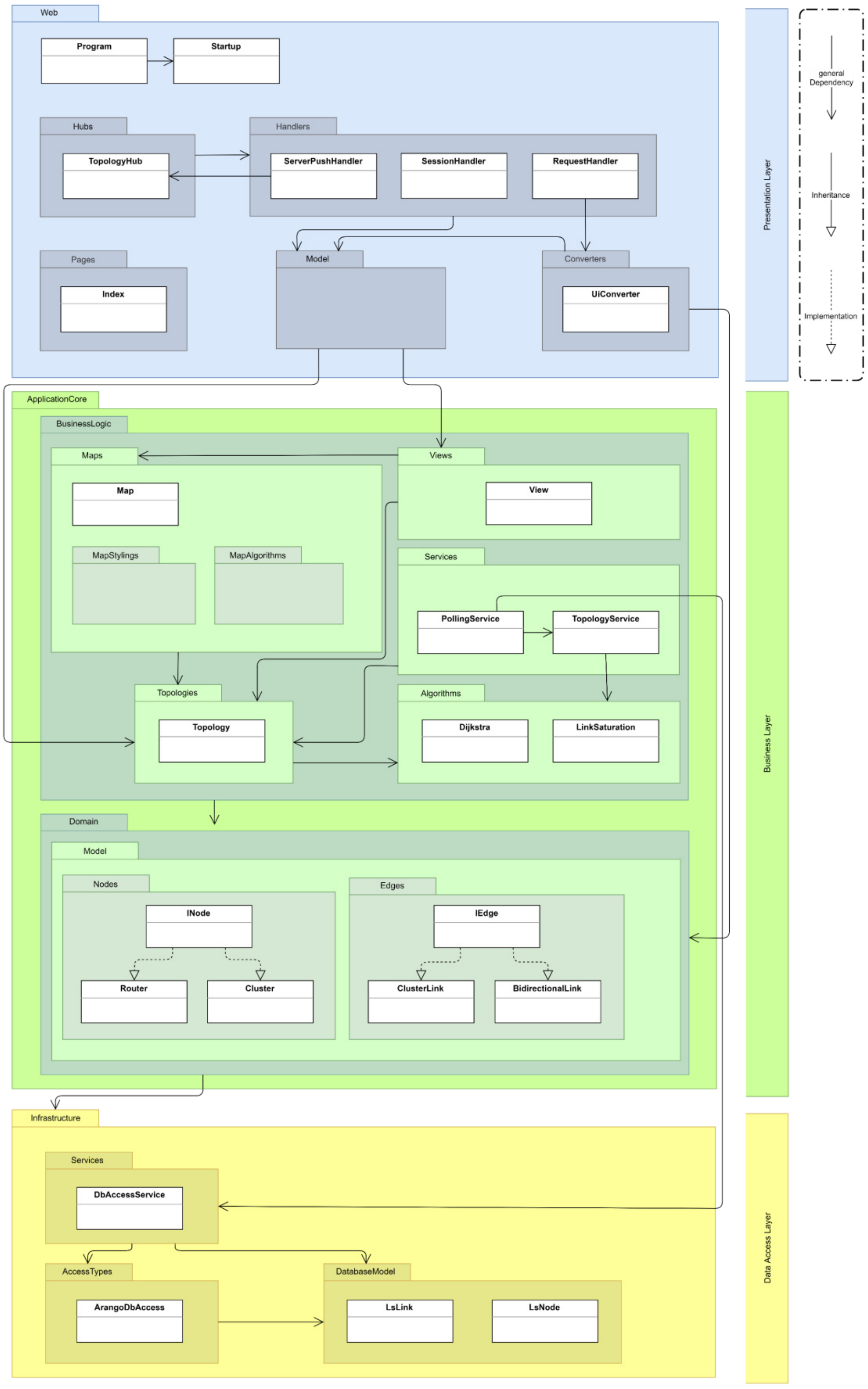


Figure 28: Package diagram
Source: own creation

7 Presentation Layer

The presentation layer is the starting point of the application. It contains the main method to startup all other components. Its package diagram is illustrated in Figure 29. It also contains a wwwroot folder which holds everything that is sent to the client, such as the markup code, style sheets and JavaScript files.

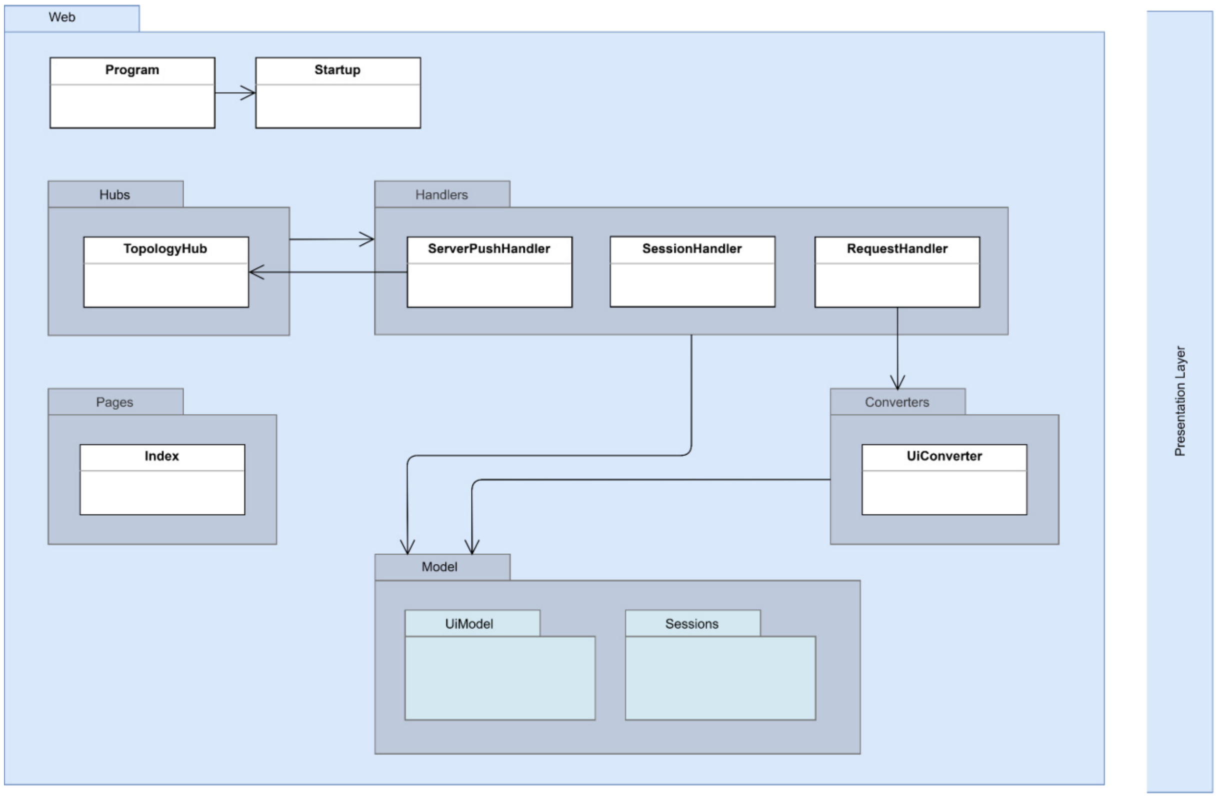


Figure 29: Package diagram – presentation layer
Source: own creation

7.1 Hubs

This package contains all hubs which are needed to communicate to the clients using SignalR (see Table 6). The hub allows bidirectional communication between clients and the server by invoking either C# methods through the client or JavaScript methods through the server.

Class	Description
TopologyHub.cs	Gets instantiated as soon the application starts up. The hub contains all methods a client can invoke and is also capable of invoking methods on the client side. The TopologyHub does not itself process the requests but passes them on to the RequestHandler.

Table 6: Hubs package - class description

7.2 Handlers

This package contains important classes to handle different type of operations (see Table 7). It handles the individual sessions and their requests, as well as updates that are to be pushed from the server to the clients, for example in case of database connectivity loss.

Class	Description
RequestHandler.cs	Instantiated as singleton. It is responsible of processing all client requests.
ServerPushHandler.cs	Instantiated as singleton. Provides callback methods to send notifications to clients.
SessionHandler.cs	Instantiated as singleton. Holds and manages all established sessions and saves session information.

Table 7: Handlers package - class description

7.3 Converters

This package holds classes which are responsible for the conversion between server-side and client-side objects (see Table 8).

Class	Description
UiConverters.cs	This class converts a server-side Topology to a client-side UiTopology.

Table 8: Converters package - class description

7.4 Model

This package contains all classes which are used to represent a topology in the UI, as well as the class to represent a Session (see Table 9). These models contain specific attributes which are needed by the frontend framework vis.js to draw the topology.

Class	Description
Session.cs	Holds all necessary information on a session. Each Session has a reference to its own copy of a topology.
UiTopology.cs	The UiTopology contains converted objects for nodes and edges, which directly represent the objects passed to the vis.js library.

Table 9: Model package - class description

8 Business Layer

The business layer is the heart of the application and contains the main logic needed for the predefined use cases. Its package diagram is illustrated in Figure 30.

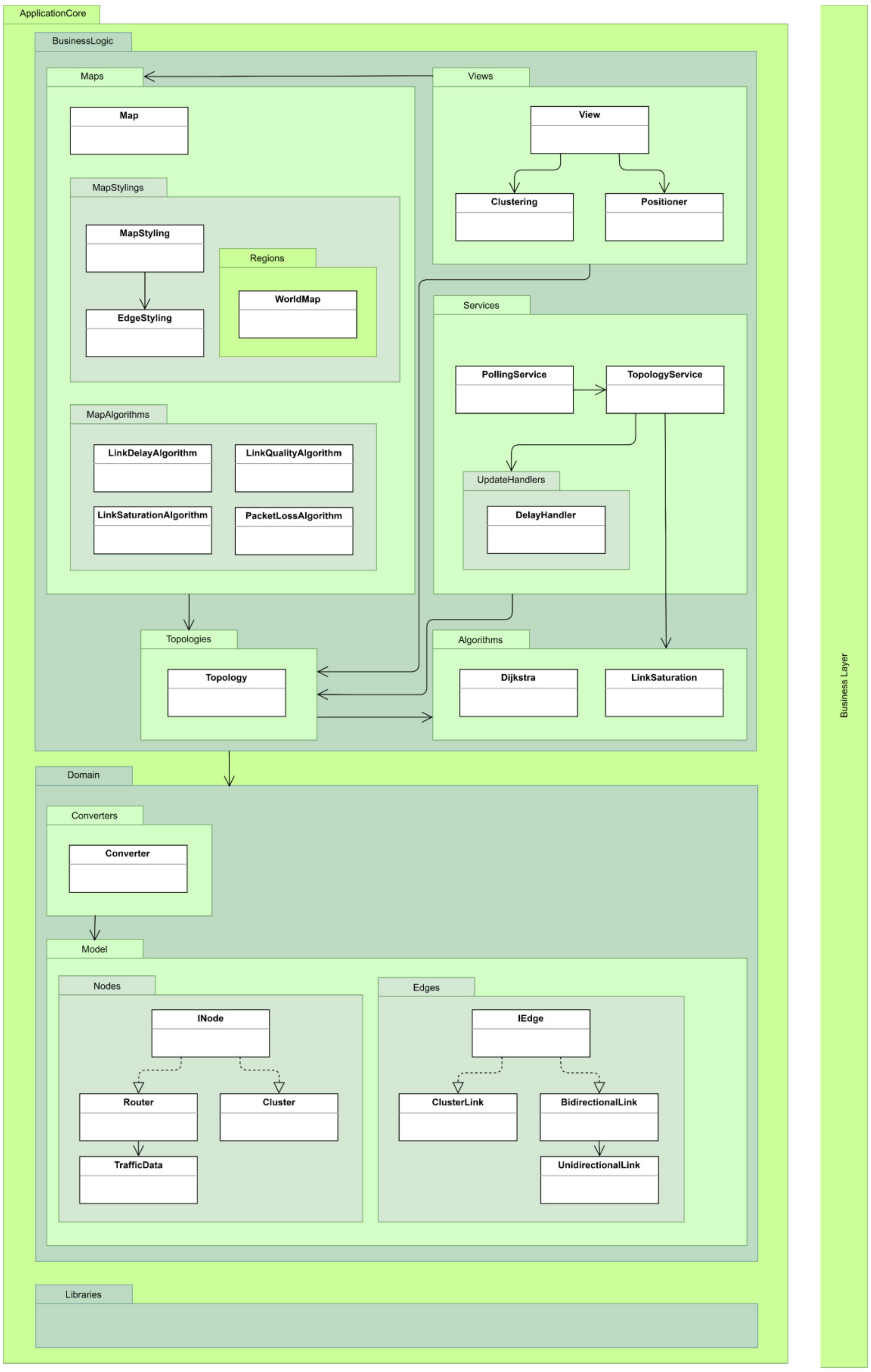


Figure 30: Package diagram – business layer
Source: own creation

8.1 Views

The Views package contains three classes (see Table 10), which are responsible for **what** the user sees in the UI.

Class	Description
View.cs	The View represents **what** the user sees in the UI, so which nodes and edges of the topology are displayed and where they are positioned. • Each session gets its own instance of a View. • The View contains all routers, clusters, links and cluster links from the current user selection. • A View contains several Maps, which describe **how** (styling) the nodes and edges are displayed.
Clustering.cs	This class handles all clustering operations. It is called by the View when the user makes any changes to the nodes of the topology.
Positioner.cs	This class calculates the positions (coordinates) of the nodes. It is called by the View whenever clusters are expanded or collapsed.

Table 10: Views package - class description

8.2 Maps

The Maps package contains everything required for **how** the nodes and edges are displayed in the UI, meaning the different heat maps. The most important classes are described in Table 11.

Class	Description
Map.cs	The Map class represents a specific heat map. • It is called by the View whenever the user makes changes to the topology. It then triggers the recalculation of the edge and node grading. • Each Map uses a specific Algorithm (such as LinkDelayAlgorithm) to grade the edges and nodes.
MapStyling.cs	The MapStyling defines the color assignment of edges based on their grades.
LinkDelayAlgorithm.cs **LinkQualityAlgorithm.cs** **LinkSaturationAlgorithm.cs** **PacketLossAlgorithm.cs**	These algorithms grade links based on predefined thresholds. As an example, the LinkSaturationAlgorithm grades links as follows: - Link saturation > 88% => Grade 2 - Link saturation > 50% => Grade 1 - Else => Grade 0

Table 11: Maps package - class description

8.3 Topologies

This package only contains the class Topology. It is described in Table 12.

Class	Description
Topology.cs	This class represents an entire topology, containing all its routers and links. - Each session gets a deep copy of the original topology, so all topology changes by the user affect only his or her session. - When a user makes any changes to the topology, such as removing or adding links or nodes, these changes will be reflected in this class. - After changes to the topology, this class triggers the recalculation of all shortest paths and the link saturation

Table 12: Topologies package - class description

8.4 Algorithms

This package contains the algorithms for the shortest path calculation and the link saturation, which are responsible for the core features of the application (see Table 13).

Class	Description
Dijkstra.cs	This is a static class that calculates all shortest path of a given topology.
LinkSaturation.cs	This is a static class that distributes all traffic of a topology based on the traffic data provided by segment routing. It requires the shortest paths to have already been calculated in advance.

Table 13: Algorithms package - class description

For more detail on how these algorithms are implemented, see Chapter 4.

8.5 Services

The Services package contains service classes (see Table 14).

Class	Description
TopologyService.cs	The TopologyService is started as a singleton from the ASP.NET startup class. It manages the original topology and provides deep copies to new sessions.
PollingService.cs	The PollingService is instantiated in the ASP.NET startup class. This service is responsible for polling data from the database and converting it to a Topology object. It does so once on system startup and then periodically every 15 minutes.
DelayHandler.cs	The DelayHandler tracks changes in the link delay values. Every time a new snapshot of the topology is taken from the database, this handler compares

	the new link delay values with the existing ones. It then keeps track of how often the value has changed for each specific link, so fluctuations can be shown in the UI.

Table 14: Services package - class description

8.6 Model

This package contains all the classes used to represent a Topology like Routers and Links (see Table 15).

Class	Description
Cluster.cs	Represents a cluster. Contains all nodes which belong to the cluster.
Router.cs	Represents a router with all its bidirectional links.
TrafficData.cs	The TrafficData object holds the number of bytes sent from a router to a specific prefix SID.
ClusterLink.cs	A ClusterLink represent a connection between two clusters. It contains a list of all BidirectionalLink belonging to the ClusterLink.
BidirectionalLink.cs	Represents a bidirectional connection between two routers. A BidirectionalLink consists of two unidirectional links. The BidirectionalLink also holds references to the two routers it connects to.
UnidirectionalLink.cs	Represents a unidirectional connection between two Routers. It holds all the properties which are used in the business logic and the UI such as bandwidth utilization, link delay, number of shortest paths, IGP metric and packet loss.

Table 15: Model package - class description

8.7 Converter

This package handles the conversion of the data retrieved from the database by the Infrastructure layer to the domain model classes used in the ApplicationCore layer (see Table 16).

Class	Description
Converter.cs	Static class which provides convertion methods. The methods are invoked by the PollingService. After converting all links and routers a Topology can be created using the convertet data.

Table 16: Converter package - class description

9 Data Access Layer

The data access layer is responsible for accessing the ArangoDB. It contains the C# library performing the requests to the database and implements methods which can be called by the ApplicationCore Layer to retrieve the data. Its package diagram is shown in Figure 31.

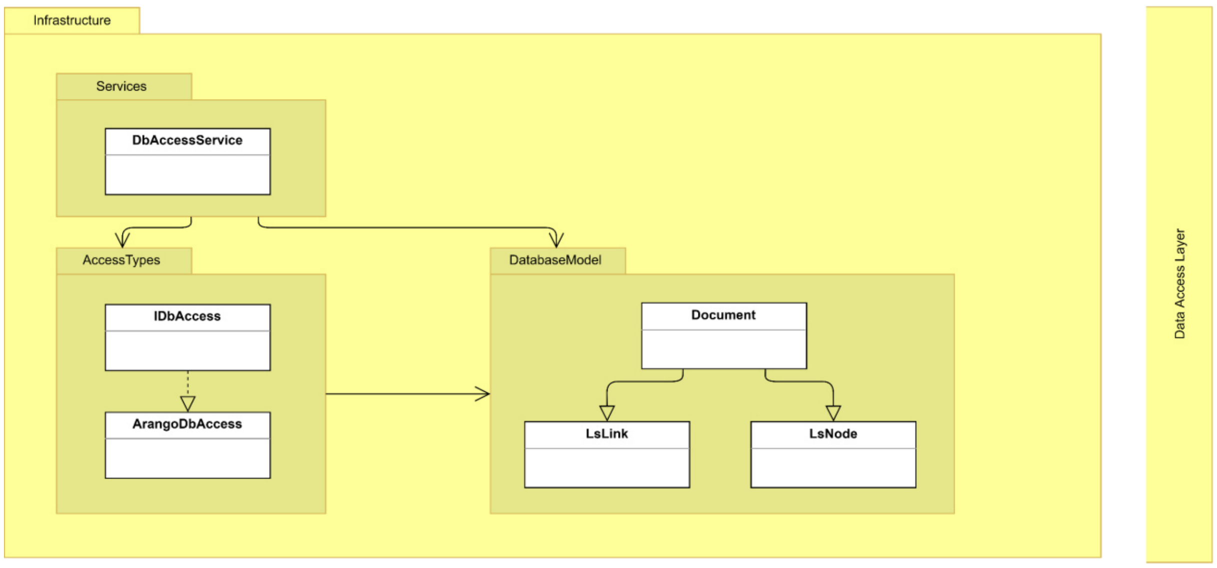


Figure 31: Package diagram - data access layer
Source: own creation

The layer consists of three packages. The Services package contains a service used by the upper layers to access the database, the DatabaseModel package contains the classes which model the data fetched from the database and the AccessTypes package contains the concrete implementation of the access to the ArangoDB.

9.1 Services

This package contains the DbAccessService class (see Table 17). The class abstracts the methods used for the data access from the implementation.

Class	Description
DbAccessService.cs	Contains methods to fetch data from the ArangoDB. The PollingService instantiates a new DbAccessService to fetch data from the database.

Table 17: Services package - class description

9.2 DatabaseModel

The DatabaseModel package contains the data model used to create objects from the data in the ArangoDB (see Table 18).

Class	Description
Document.cs	Represents a document as stored inside a collection of the ArangoDB. Holds the unique key attribute.
LSLink.cs	Contains the properties retrieved from the data inside one LSLink document.
LSNode.cs	Contains the properties retrieved from the data inside one LSNode document.

Table 18: DatabaseModel package - class description

9.3 AccessTypes

The AccessTypes package contains the concrete implementation of the data access (see Table 19). The ArangoDBNetStandard driver is used to access the database.

Class	Description
ArangoDBAcces.cs	Contains async methods to access the ArangoDB. The ArangoDBAccess is used in the PollingService. When the DbAccessService is instantiated a reference to the ArangoDBAccess instance is provided following the Strategy Pattern.

Table 19: AccessTypes package - class description

9.3.1 Most important operations

The most important operation is the initial retrieving of data from the ArangoDB upon application start. Figure 32 displays the interaction between the PollingService in the business layer and the ArangoDBAccess in the data access layer.

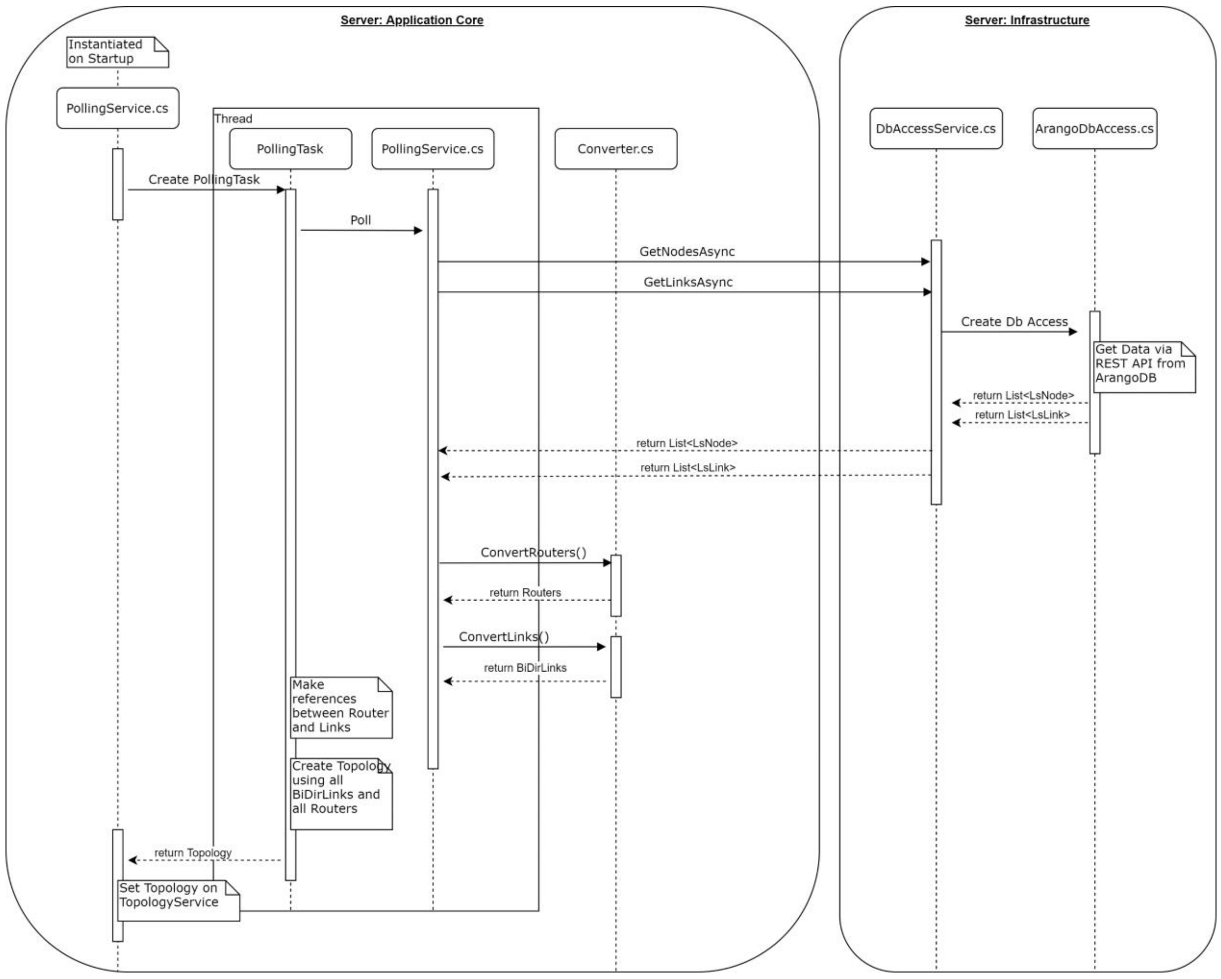


Figure 32: sequence diagram data access
Source: own creation

10 Libraries

Table 20 lists all external libraries used in the project which do not belong to .NET Core.

Layer	Library	Description
Infrastructure	ArangoDBNetStandard (1.0.0)	This library was installed using the C# NuGet package manager. It contains the assemblies which provide the classes and methods to access the ArangoDB. Internally all calls to the database are made using the API ArangoDB provides.

Table 20: External libraries

11 Processes and Threads

This chapter describes the processes and threads used inside the SR-App.

11.1 Processes

Due to the monolithic architecture of SR-App the whole application runs inside a single process. The main method in the Program.cs class of the Web layer is called first and starts all other components of the application.

11.2 Parallelization

The application uses the C# Task Parallel Library to perform the polling of data from the ArangoDB and some parts of the algorithms for Shortest Path and Link Saturation calculation asynchronously. The tasks used for polling data are started inside the Startup.cs class

11.2.1 Polling Task

A dedicated task is used for periodically polling new data from the database. It ensures that the business logic works with current data and to provides a responsive application while the data is loaded from the database. The task is started on system startup and runs for the lifetime of the application. It polls data from the database every 15 minutes.

12 Deployment

SR-App is a classic client-server application. The deployment is simple and can be containerized to scale the application. Figure 33 displays a sample deployment on a Linux Server using docker-compose. The deployment consists of two containers one containing the SR-App instance and the other the ArangoDB database containing the topology data.

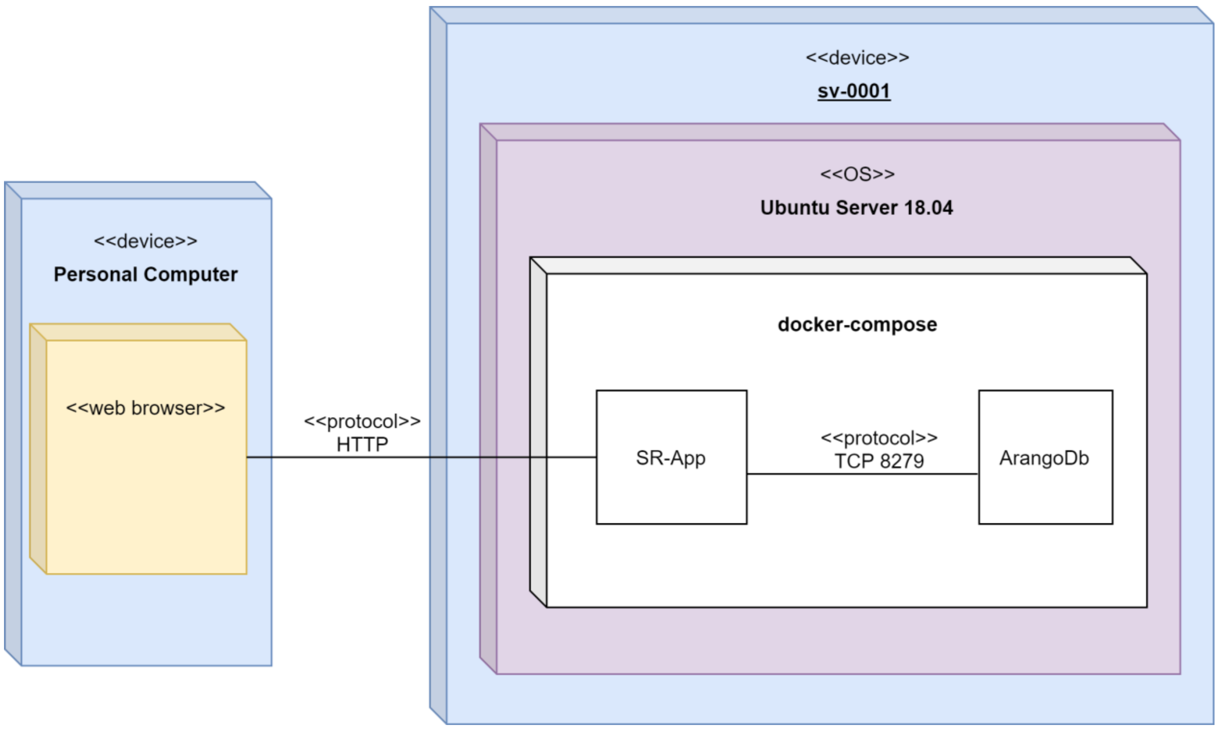


Figure 33: Deployment diagram

13 Illustration index

14 Glossary

Term	Description
API	Application Programming Interface
GHz	GigaHertz
IATA	International Air Transport Association
IGP	Interior Gateway Protocol
Prefix SID	Prefix Segment Identifier
RAM	Random Access Memory
SR	Segment Routing
UI	User Interface

Table 21: Glossary