V-SDN : Visualizer for Software Defined Network

Marius Faust, *TU Darmstadt* and Manjiri Birajdar, *TU Darmstadt*

***Abstract*—Human eyes require a meaningful layout to look at useful information. Similarly, its always easy to visualize the complex network functionalities instead of just reading the text description in console. With the help of modern browser engines, we are able to build interactive real-time visualization applications that enable structured understanding of the complex data. We present "V-SDN", an interactive web based application for visualizing the Software Defined Network (SDN). It includes a backend for processing the client requests based on python based web framework. Further as a frontend, a web interface based on HTML, CSS, JavaScript and capable of displaying network topology with additional node information. We also provide generic APIs to interact and build new visualizations. In this paper, we describe our approach including system overview, design and implementation, Application Programming Interface (API) endpoints and user interface (UI).**

***Index Terms*—Software Defined Network (SDN), Visualization, D3.js, Flask, Python, REST API**

I. Introduction and Motivation

Software-Defined Network (SDN) is model for designing, building and managing networks. SDN has three different planes decoupled from one another named Data Plane, Control Plane and Application Plane [1].

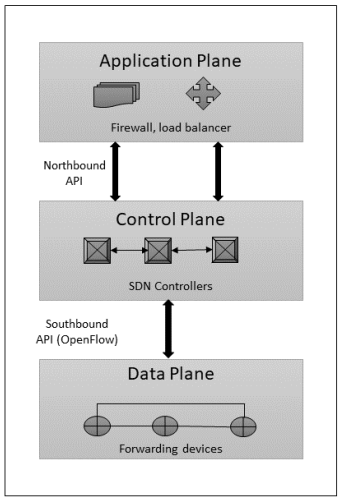

Fig. 1. SDN Architecture [2]

Fig. 1 depicts the SDN Architecture. We briefly explain about each layer as follows: **Data Plane** contains forwarding devices for example switches. This underlying data plane is mostly abstracted from the control and application layer. **Control Plane** acts as a brain of SDN. It contains SDN controllers which are designed to manage different kinds of networks, maintain states and topology details of a network and far more. For instance, SDN controllers must have functional logic for switching, routing, firewall security rules, DNS, DHCP and clustering [3]. At last, **Application Plane** is where innovative application development is feasible for instance building network configuration and management, network monitoring and troubleshooting, network policies and security, visualization tools, etc. These applications can provide end-to-end solutions for existing networks. For instance, HPE is a vendor having a SDN App store which contains variety of apps from different companies [4]. Control Plane lies in between data plane and application plane. It communicates with application plane via Northbound interface APIs such as through REST APIs of SDN Controllers. Whereas, it communicates with data plane using Southbound APIs such as protocols named Openflow and Netconf [3].

SDN Networks are huge and complex. Visualizing the entire network and its details is quite challenging task. Specifically, viewing the topology helps users to grasp the complex networks overview. Following are number of core reasons to build a visualizer for SDN:

1) **Interactive User Interface**
   - It is difficult to read from Command Line Interface (CLI) because it is a text-only interface.
   - It's easy to navigate through a system with graphical user interface (GUI).
2) **Node-link structures are more intuitive**
   - Most of the data structures consist of node-link form are easy to visualize.
   - It is easy to visualize communication among entities via links
3) **Easy to grasp complex coherence**
   - It simplifies complexity of the network
   - It gives overview look at the context of the network and helps to understand details
   - It provides simplified view to identify different networks and its nodes-links

There are variety of existing network visualizers and some of them which are relevant to SDN are listed in section II. However, most of them are only for mininet and based on manually copy-pasting script. Also, none of them had extensible API functionality which we wanted to implement. Therefore, We present "V-SDN", a interactive web based solution for visualizing SDN. It is based on "Flask" [5], a python web application framework. Our designed system consists of backend for processing the client's requests-response and frontend displays the network topology using D3.js [6]. V-SDN provides users generic APIs to create and interact with the network. Moreover, the chosen network layout is based on force-directed graph [7] because it draws node-link graph

(network graph) and displays additional pieces of information about nodes for example by hovering over the any node.

We organize our paper as follows: section II describes existing approaches to build visualization tools for SDN. With section III, we introduce our approach with technical reasoning. Next in section IV, we describe the system overview with technical implementation details of backend and frontend. Later in section V, we describe implementation details including generic APIs, protocols and data formats. Further in section VI, we evaluate the strengths and limitations of our application. We describe the challenges encountered and future scope for our application in section VII. At last, we conclude the paper in section VIII with Acknowledgement.

II. Related Work

The need for SDN visualization resulted in development of different types of systems. Table I on the next page, lists the benefits and drawbacks of some of the available approaches for visualizing SDN.

The HPE Network visualizer [4] is one of the best available SDN visualizer applications. It is a web based system and provides topology viewer, dynamic traffic capture with real-time network monitoring and much more. But the main drawback is that it's not open source. One must buy the license for this product and additional licenses for all the products listed in prerequisites. Moreover, the product has been discontinued in some of the countries.

On the other hand, SDN SPEAR [8] is a Mininet Topology Visualizer created by NARMOX. It can generate network topology graph for Mininet environment. Specifically, we need to use "dump" and "links" commands in Mininet console (CLI) [8]. Afterwards, we need to copy-paste the output of the commands to given boxes and then the graph will be rendered based on this input. It's easy to use and graph is simple enough to understand. However, the SDN Narmox Spear works on top of Aruba VAN SDN Controller which is proprietary and has no extensible functionality. Also, this approach only supports Mininet environment.

SDN topology editor (ME) [9] is a web based tool for visualizing the SDN. This tool provides import and export options for JSON files and python scripts. User Interface (UI) is a simple canvas with responsive design and easy to understand. It has all SDN nodes and includes startup and shutdown scripts. The implementation is based on Vuetify, Vue.js JavaScript framework. The downside of this tool is that it is completely GUI based and does not provide any generic APIs for extensibility.

Each tool listed in Table I has some or the other flaw. Next section III describes how our approach differs from all of them.

III. Our Approach

Our approach differs in many aspects from other approaches discussed in section II based on the following characteristics:

1) Generic APIs for extensibility
2) Platform independent
3) Multi-request handler
4) Decoupled frontend and backend
5) Interactive user interface
6) Development based on well known technologies

V-SDN also has the advantage that its based on open-source technologies and differs in implementation from other approaches. Align with the above characteristics, the main objectives for our lab were as follows:

1) Develop a SDN Visualizer (Display Network Topology)
2) Provide generic APIs with extensible functionality for instance: addSwitch(), addHost(), etc
3) Enable accessibility to various programs like JAVA SDN Controller, python scripts, etc.

Mainly, V-SDN consists of two decoupled components. A backend, which is responsible for data processing and generating intermediate data file called graph.json. The second part is a interactive browser based JavaScript application which displays the network topology created in graph.json. Following are the reasons behind choosing specific technology and decisions for building V-SDN:

We preferred web application over desktop application as its easier to install, develop and accessible from wherever you want. This leads to application being platform independent. REST APIs is the most standard, logical, efficient and simple way of creating generic APIs and also to decouple the components in system.

We specifically decoupled our solution in backend and frontend for multiple reasons: First, it's much easier to work apart on the components since we could define common interfaces. Second, we wanted to make it extensible for future projects and to simultaneously focus on development of different components of system. And the third, we made our code modular to some extent so that it is possible to send the network details to several frontends or develop the frontend to receive the data from multiple sources. Also, the backend can be accessed from all kinds of programs and applications regardless of their language and number of programs.

As a backend solution, we decided to use Flask [5], a python based web framework. Python is an easy to use programming language and provides numerous packages to help us with our approach. Furthermore, it's already heavily used by the stakeholders of our application. Since, our visualizer should be extensible (hence the decoupled solution), it was an obvious choice to chose python and make it easier for future projects to work on it.

Frontend is developed using D3.js [6], a library that allows data-driven manipulations of the websites document object model (DOM). There are plenty of frameworks and platforms available for building different network graphs for visualization. For example, Gephi, Cytoscape, Tulip, Prefuse, PyQt, Tkinter and etc. On the other hand, D3.js requires less time for understanding and implementation of required functionalities. D3.js is declarative in nature which is better for building interactive application. After researching about these frameworks, we found that, most of them has quite a steep learning curve and does not provide native look and feel like D3.js. Moreover, D3.js gives more importance to web standards such as HTML, Scalable Vector Graphics (SVG), and CSS and provides complete capabilities of latest browsers

Approach	Benefits	Drawbacks
HPE Virtual Application Networks (VAN) SDN Controller / Aruba Network Visualizer App. [4]	Provide dynamic traffic capture with real-time and detailed network monitoring	This product has been discontinued and is not for sale by HPE in any region
SPEAR : SDN Narmox Spear [8] (application enhances the functionality of HPE VAN SDN Controller)	Complete tool for SDN network administration, Manage network traffic, Shows graphical overview of networks current state	Dependency on Aruba VAN SDN Controller software and its not open source
Visual Network Description (VND) [10]	Authoring of SDN Network Scenarios via GUI [10]. Automatic creation of Mininet Scripts [10].	Allows only Drag and drop components, they are not accessible by code.
Mininet GUI's: consoles.py and miniedit.py	GUI for creating a Mininet, Display details of each node in the network	No run time visualization for the network and its architecture, Cannot change the number of nodes dynamically
Mininet Editor	Visualizes Mininet, Apply changes via drag and drop, Export topology and create new Mininet	Cannot be accessed by code, Dependency on server/website, No run time visualization
Network Topology Visualization	Visualizes live network using NETCONF and SNMP, using HTML/CSS and JavaScript + D3 library, uses JSON format for communication	No API for interacting with GUI,irrelevant for our task such as mapping with NETCONF,python and PyHPEcw7

TABLE I
BENEFITS AND DRAWBACKS OF EXISTING APPROACHES FOR SDN VISUALIZERS

by merging strong visualization modules and data-driven transformation for Document Object Model (DOM) manipulation [6]. Moreover, it's based on JavaScript which makes it easier to understand, flexible and helps building dynamic and interactive web pages. With D3.js, complex interactive visualizations of arbitrary data can be build efficiently [11]. V-SDN uses force-directed graph layout as it draws node-link graph (network graph) [7] using velocity Verlet integration [12]. Also, D3.js uses JSON structure of nodes and links.

IV. SYSTEM OVERVIEW

The system design has been shown in Fig. 2. As previously mentioned above, our system consists of two parts, backend and frontend. Both are decoupled and backend is available as a service. Further, backend consist of request handler which serves as entry point for the application. It can handle multiple requests from multiple clients simultaneously. The core application consists of APIs designed in python. The core functionalities like network creation, node and link creation are handled in the implementation of these APIs. The user/client request has specific format and later that would be processed by APIs. The final output will be a graph.json which contains node and links structure as shown in Listing 1. This graph.json acts as a input to frontend. By using D3.js, HTML, CSS and JavaScript, we render this graph.json to web browser and network topology will be displayed. Backend and frontend are described in details as follows:

A. Backend : Flask-python based web framework

Flask [5] is a package to simplify creation of an APIs in python. It is designed to create http-routes with a minimum overhead for the underlying functions. Simply, by adding the tag @app.route("routePath") to a function, makes it receiver for a http-endpoint. However, routing of APIs was out of our scope. Moreover, the negative aspect of implementing it would be an additional work for users to search through routes while extending the functionality.

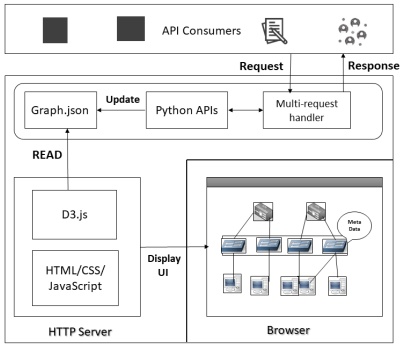

Fig. 2. System Design

B. Frontend : HTML+CSS+JavaScript+D3.js

The V-SDN's frontend is implemented as an interactive JavaScript based application that's automatically rendered using D3.js, HTML and CSS in a browser. V-SDN uses Force-directed graph layout [7] using velocity Verlet integration [12]. D3.js uses JSON structure of nodes and links.

V. IMPLEMENTATION

This section describes V-SDN's backend and frontend implementation in detail. Described approaches in [9] and [13] does not match with the approach we wanted to implement. Therefore, we did not use any existing code reference from mininet editors. However, our approach is comparable to this approach [14]. But our approach is more simpler and refined than this.

Nevertheless, our backend solution provides the necessary data for visualization. There is possibility for extending the backend without interacting with any visualization components. This technical flexibility allows for rapid development and early visualization of given data sets. In the following subsections, we explain each component of the system shown in Fig. 2 in detail.

A. API Consumers

The API can be accessed by any script of a language that can create http requests. In our case, it would be the frontend via JavaScript and by the user probably in a python script that also controls and manages the SDN.

B. Request Handler and Processor

When a route gets called, the request is handled by the function assigned to the route. Depending of the request method, it calls a processor function to create, delete, get or modify parts of the network.

The processor functions create, delete, get or modify parts of the network. Depending on the component, these functions can differ from each other. They also check the legality of the action.

C. REST API

Following are four main important data transactions in any REST system and HTTP specification: POST (create), GET (read and consult), PUT (edit) and DELETE [15]. All types of nodes have a GET, POST and DELETE endpoints. The network has only a GET endpoint whereas links has POST, GET, PUT and DELETE.

Fig. 3 shows the network API endpoint's description. It returns the JSON values of links and nodes. We dump these values in graph.json and frontend renders it to the browser. Next, Fig. 4, 5, 6 and 7 shows detailed API endpoints for controller, host, switch and link.

Endpoint network
package_name: NetworkApi

Description: This endpoint provides methods to get the network information

Method Summary:

Methods

Type	Parameters	Return Json	Curl-Example
GET		{"links": [Link], "nodes": [Node]}	curl 127.0.0.1:5000/network
	Description: The GET-endpoint provides the network structure. It should be used to by the GUI to get the data.		

Fig. 3. API Endpoint : network

D. Protocol

We used HTTP-based RESTful APIs. Messages sent between backend, clients and frontend are JSON encoded. JSON exchanged between backend and client have a specific data format. Please refer API-Endpoints Fig. 3 4, 5, 6 and 7 in this paper for more details.

E. graph.json

This topology file contains the graph nodes (devices) and links (interfaces) in JSON format and we use this as a input to visualization [14]. Here, Listing 1. shows the graph.json attributes:

Endpoint controller
package_name: NetworktApi

Description: This endpoint provides methods to manage the controllers in the network

Method Summary :

Methods

Type	Parameters	Return Json	Curl-Example
GET	id: Optional(int), component_id: int	{"component_id":Controller, "response_id":Optional(int), "success":boolean}	curl 127.0.0.1:5000/controller?component_id=3
	Descrition: The GET-endpoint is used to get informations about a specific controller.		
POST	id: Optional(int), ip: int, port: int, name: Optional(string)	{"component_id":int, "response_id":Optional(int), "success":boolean}	curl -X POST -d "ip=1&port=2" 127.0.0.1:5000/controller
	Description: The POST-endpoint is used to create a controller in the network.		
DELETE	id: Optional(int), component_id: int	{"component_id":int, "response_id":Optional(int), "success":boolean}	curl -X DELETE -d "component_id=1" 127.0.0.1:5000/controller
	Description: The DELETE-endpoint is used to remove a controller from the network.		

Fig. 4. API Endpoint : controller

Endpoint host
package_name: NetworkApi

Description: This endpoint provides methods to manage the hosts in the network

Method Summary :

Methods

Type	Parameters	Return Json	Curl-Example
GET	id: Optional(int), component_id: int	{"component_id":Host, "response_id":Optional(int), "success":boolean}	curl 127.0.0.1:5000/host?component_id=3
	Descrition: The GET-endpoint is used to get informations about a specific host.		
POST	id: Optional(int), ip: int, mac: int, name: Optional(string)	{"component_id":int, "response_id":Optional(int), "success":boolean}	curl -X POST -d "ip=1&mac=2" 127.0.0.1:5000/host
	Description: The POST-endpoint is used to create a host in the network.		
DELETE	id: Optional(int), component_id: int	{"component_id":int, "response_id":Optional(int), "success":boolean}	curl -X DELETE -d "component_id=1" 127.0.0.1:5000/host
	Description: The DELETE-endpoint is used to remove a host from the network.		

Fig. 5. API Endpoint : host

Endpoint switch
package_name: NetworkApi

Description: This endpoint provides methods to manage the switches in the network

Method Summary :

Methods

Type	Parameters	Return Json	Curl-Example
GET	id: Optional(int), component_id: int	{"component_id":Switch, "response_id":Optional(int), "success":boolean}	curl 127.0.0.1:5000/switch?component_id=3
	Descrition: The GET-endpoint is used to get informations about a specific switch.		
POST	id: Optional(int), ip: int, host_count: int, name: Optional(string)	{"component_id":int, "response_id":Optional(int), "success":boolean}	curl -X POST -d "ip=1&host_count=2" 127.0.0.1:5000/switch
	Description: The POST-endpoint is used to create a switch in the network.		
DELETE	id: Optional(int), component_id: int	{"component_id":int, "response_id":Optional(int), "success":boolean}	curl -X DELETE -d "component_id=1" 127.0.0.1:5000/switch
	Description: The DELETE-endpoint is used to remove a switch from the network.		

Fig. 6. API Endpoint : switch

Endpoint link
package_name: NetworkApi

Description: This endpoint provides methods to manage the links in the network

Method Summary :

Methods

Type	Parameters	Return Json	Curl-Example
GET	id: Optional(int), component_id: int	**{"component_id":Link, "response_id":Optional(int), "success":boolean}**	curl 127.0.0.1:5000/link?component_id=3
	Descrition: The GET-endpoint is used to get informations about a specific link.		
POST	id: Optional(int), source: int, target: int, source_port: Optional(int), target_port: Optional(int)	**{"component_id":int, "response_id":Optional(int), "success":boolean}**	curl -X POST -d "source=1&target=2" 127.0.0.1:5000/link
	Description: The POST-endpoint is used to create a link in the network.		
DELETE	id: Optional(int), component_id: int	**{"component_id":int, "response_id":Optional(int), "success":boolean}**	curl -X DELETE -d "component_id=1" 127.0.0.1:5000/link
	Description: The DELETE-endpoint is used to remove a link from the network.		
PUT	id: Optional(int), component_id: int, value: int	**{"component_id":int, "response_id":Optional(int), "success":boolean}**	curl -X PUT -d "component_id=1&value=13" 127.0.0.1:5000/link
	Description: The PUT-endpoint is used to change the link_value of a link which is used for its depiction in the GUI.		

Fig. 7. API Endpoint : link

Listing 1. Attributes in graph.json

```
Link : id, source, target, value

Node : group, id, ip_address, mac_address, port

Message : id, path[source, target]
```

Link has id, source address, destination address and value. Link value has to be any of 10,20,30 or 40. Each value decides the color and thickness plus which nodes to link. For example, value 10 should be used to create link between switch and any host.

F. D3.js JavaScript Library

D3.js draws Scalable Vector Graphics (SVG) graph using D3 force simulation [12] and reading graph.json. It uses D3's Y-axis gravity settings to create several gravity lines which form hierarchy to distinguish controllers, switches and hosts. This alignment has been referred from here [14]. This can be seen in Fig. 8. The need for preferring this view is that its simple to view and modifications does not lead to complete distortion of user interface. The nodes and node-link position stays stable even after modifying the topology. The user interface has similarities with [14] and [7]. On mouse hover of nodes, you can see the node's attributes mentioned in Listing 1.

G. Deployment on Heroku Server

The application is easy to deploy on any HTTP server. With the help of [16], we decided to use heroku server [17] as its free and easy to deploy using git code base.

VI. Testing and Evaluation

We deployed our application on heroku server and tested it. We evaluate our application's frontend by creating a test

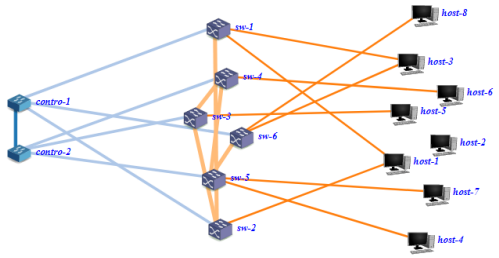

Fig. 8. V-SDN interface

scenario as shown below. We first consider the ability of frontend to handle huge amount of network components : controllers, switches and network with links. We consider a network of 5 controllers, 20 switches and 30 hosts. Thus, we tested operations such as add, modify and remove network components by editing graph.json. As expected, it performed very well with each operation without distorting the complete UI view. The transition was stable and smooth.

Secondly, to test our backend python APIs, we used *curl* [18] a command line tool to fetch the data via URL using HTTP protocol. For example, the curl command below :

curl 127.0.0.1:5000/network

gets the network structure data. The output will look like *"links": [Link], "nodes": [Node]*. We tested all other curl requests for creating networks, controllers, switches and host. More examples are explained in given API documentation in Fig. 4, 5, 6 and 7.

The lessons from above testing are mentioned below as strengths and limitations:

A. Strengths

The architecture of our system allows us to provide the functionality for all kinds of networks on variety of applications. The decoupled approach of frontend and backend makes it easier to change parts of the visualizer and expand its functionality. It also addresses the use case of multiple clients supplying input for our APIs as well. Due to web-based nature, any client with a modern browser can be used to interact with V-SDN without any additional software installations required.

B. Limitations

The drawback of force-directed graph technique used in the frontend is that nodes are connected using straight lines which results as overlapping lines [11]. Therefore, large data set with several nodes and links will lead to a huge amount of crossing lines. This decreases the links and nodes visibility in complex network. One of the solutions is to create force-directed layout of tree shown in [19]. This might result in increased visibility of network layout. Another new approach to visualize large-scale network is Hive Plots [20] mentioned in [11].

Another downside of the application is that if there are any errors in graph.json structure, the user interface will not be displayed. Therefore, every client's input has to be correct and validated at backend or by user himself, otherwise there

is no visible output. To solve this issue, we can write our own JSON schema and its validator in JavaScript or use existing implementations for JSON schema validators.

During design and implementation of our approach, we faced several challenges. Following section elaborates them in more details.

VII. Challenges and Future Work

1) Real-time update : Real-time updates of the visualization require a complete reload of the network structure in our current solution. This includes sending a request to the API, to receive a full network in form of the graph.json and reloading it. Depending on how big the network is, it might not be possible to reload the network in a rate, that it's suitable for real-time updates.

2) Overloaded complexity of the network : The visualizer's main functionality is to make the network more comprehensible for the user. In small to medium sized networks with a low link count, this works very well in a web browser. The bigger the network gets and the more colluded with links, the harder it gets for the user to comprehend the network since a browser window can only fit a limited amount of nodes.

3) Display network traffic : For displaying the message traffic in the network, we thought of different visualization methods. One possibility would be displaying a message as an icon traveling along the links of the networks from one place to another via given path links. This approach conflicts with the real time aspect of the visualizer, because a message traveling at real speed in the visualization wouldn't be visible for the user. Another possibility would be thickening the links with many traveling messages and thinning those with few. This would give the user a better overview of the network traffic, but takes away the opportunity to view individual message.

4) Generic API design : There is a conflict between making the API generic to support all kinds of networks on all kinds of applications and displaying the maximum detail of a network. Currently, we optimized our solution for a generic approach, but that makes it more complicated to add details or constraint checks. For example, it might be possible that in one network a host can only connect to one switch and not to a controller. In a more specialized visualizer, the API could check that these constraints are met, but in a generic API, it is not possible.

5) Recognize the network after modification : The feature that the nodes of a network can be added at any time, makes it necessary that the nodes in the visualization do not have a set coordinates. Adding a node might make a reorganization necessary to keep the visualization stable and alike previous view. Writing an algorithm for that was not in the scope of our lab, so we decided to use a library for it. This comes with the restrictions of the used library, in our case it means that we have to load the network from a json.

As a future work in our approach, we would like to extend it to visualize the message (packets) transfer via given path links among different nodes in SDN. One way to implement it is to take a list of link Id's (path) from a user then at backend, convert it into a list of links structure as [source, target and value] and highlight the paths with some delays. Positively, we already implemented few steps towards it but there are serious bugs to fix in visualization part. There are some other possible approaches to improve the visualization, for instance, including responsive design to support view in mobile and other devices. We could also implement solution for displaying different groups of networks using tree-based visualization.

VIII. Conclusion

In this paper, we introduced V-SDN, an interactive web application for SDN visualization. Our decoupled system approach offers generic APIs for clients who wants to extend this application. Handling multiple different clients request is one of the unique features we implemented. Our application functions correctly with medium sized data set but there are few limitations too. For SDN researchers, this visualizer is an important tool that can help to understand complex SDN and its characteristics. Finally, the above results denotes that researchers can use this valuable application as a base and extend it to build a new and innovative featured visualization tools for better interaction with Software Defined Network.

Acknowledgment

We thank our supervisor Dr.-Ing. Rhaban Hark for guiding us and Multimedia Communications Lab (KOM) department for giving us this opportunity and platform.

References

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie. A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, 17(1):27–51, Firstquarter 2015.

[2] Margaret Rouse. software-defined networking (sdn). Website, https://searchnetworking.techtarget.com/definition/software-defined-networking-SDN.

[3] Himanshu Arora. Software defined networking (sdn) - architecture and role of openflow. Website, https://www.howtoforge.com/tutorial/software-defined-networking-sdn-architecture-and-role-of-openflow/.

[4] Hewlett Packard Enterprise Development LP. Hpe network visualizer. Website, https://community.arubanetworks.com/t5/HPE-Network-Visualizer-Free/ct-p/HPENetworkVisualizerFreeTrial.

[5] Flask. Website, https://flask.palletsprojects.com/en/1.1.x/#.

[6] D3 js data-driven documents. Website, http://d3js.org.

[7] Force-directed graph. Website, https://observablehq.com/@d3/force-directed-graph.

[8] SPEAR by Narmox. Mininet topology visualizer. Website, http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet.

[9] Tom Vytal. Mininet editor. Website, https://thomaash.github.io/me/#/home.

[10] Dr. Ramon Fontes. Vnd (sdn version). Website, https://github.com/ramonfontes/vnd.

[11] Lothar Braun. Flow-inspector: A framework for visualizing network flow data using current web technologies. *First IMC Workshop on Internet Visualization (WIV 2012), November 13, 2012, Boston, Massachusetts, USA*, 2012.

[12] d3-force. Website, https://github.com/d3/d3-force.
[13] Mininet. Mininet guis: consoles.py and miniedit.py. Website, https://github.com/mininet/mininet/wiki/Ideas#mininet-guis-consolespy-and-minieditpy.
[14] PETER HAVRILA. Network topology visualization. Website, https://networkgeekstuff.com/networking/network-topology-visualization-example-of-using-lldp-neighborships-netconf-and-little-python-javascript/.
[15] BBVA Open4U. Rest api: What is it, and what are its advantages in project development? Website, https://bbvaopen4u.com/en/actualidad/rest-api-what-it-and-what-are-its-advantages-project-development.
[16] Combining python and d3.js to create dynamic visualization applications. Website, https://towardsdatascience.com/combining-python-and-d3-js-to-create-dynamic-visualization-applications-73c87a494396.
[17] Getting started on heroku with python. Website, https://devcenter.heroku.com/articles/getting-started-with-python.
[18] Curl. Website, https://curl.haxx.se/.
[19] Force-directed tree. Website, https://observablehq.com/@d3/force-directed-tree.
[20] Martin Krzywinski, Inanc Birol, Steven JM Jones, and Marco A Marra. Hive plots - rational approach to visualizing networks. *Briefings in Bioinformatics*, 13(5):627–644, 12 2011.