Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik
Arbeitsbereich Kommunikationsnetze

Bachelorarbeit Informatik

Implementation of a Web-Based Visualizer for Service Function Chaining Infrastructure

Charlotte Rupp

30.04.2022

Gutachter

Prof. Dr. habil. Michael Menth
Fachbereich Informatik
Arbeitsbereich Kommunikationsnetze
Universität Tübingen

Betreuer

Benjamin Steinert, M.Sc. Informatik

Charlotte Rupp:

Implementation of a Web-Based Visualizer for Service Function Chaining Infrastructure
Bachelorarbeit Informatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 01.01.2022 - 30.04.2022

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Tübingen, den 30.04.2022

(Charlotte Rupp)

Contents

This thesis makes recent technological advancements in Cloud Computing accessible by designing and implementing a visualization system for Service Function Chaining (SFC) infrastructure. The visualized network architecture builds on technologies like Software Defined Networking (SDN), Virtual Network Function (VNF) and SFC. The project is supposed to become part of an application to simplify a SFC orchestrator to a point where it can be used without detailed knowledge about the underlying technical frameworks and processes.

The web application and the visualization system are designed to be scalable and modular to ensure future reuse of the work. A flexible and interactive visualizer was implemented using Typescript and the D3.js package and integrated into a full-stack web application. It provides an overview of the network structure by creating a clear, interactive and multilayered network graph visualization from network infrastructure data. The application consists of an Angular frontend with a responsive user interface layout and a Django backend connected via a REST API. This bachelor thesis also offers necessary information on how the implemented visualizer can be used for arbitrary SFC architectures and possible ways to get the infrastructure data. The SFC Infrastructure Visualizer can be used for validating and debugging SFC infrastructure.

Acknowledgment

Special thanks to Benjamin Steinert and Marco Häberle for answering all my questions and motivating me to keep on working. I would also like to thank my family and friends for supporting me, especially during the last few months. I am grateful for the opportunity to work on this project which has provided me with valuable experience and information about network architecture and web application architecture.

1. Introduction

Traditional computer network management is a labor-intensive task, where very skilled people are necessary to configure and manage a wide variety of hardware and software. Many parts of the system have to be managed separately. For example, the settings of different servers have to be configured and updated regularly. Scripts and configuration management software like Ansible [4] can be helpful. But even then, changing hardware and software need to be integrated into a complex system, maintained and regularly checked for emerging vulnerabilities and problems. On a bigger scale, this can become an overwhelming task without a perfect team of up-to-date engineers always monitoring and solving problems. Even then, mistakes occur regularly, which threaten the reliability and safety of the network because so many parts of the network have to be configured manually, and people make mistakes. In a time when every institution is expected to have an internet presence, traditional computer network management becomes unfeasible. The drive to more and more automation aims to reduce the workload of computer network management and increase reliability and resource efficiency.

Cloud technologies have become very popular in recent years because they offer dynamic scalability and a more centralized approach to network management. The concept of Infrastructure as Service offers the possibility to separate hardware and software management. Cloud Service Providers (CSPs) offer self-service on-demand virtual computing resources [62]. Because big CSPs manage vast amounts of computing and storage resources, they can provide more reliance, flexibility, and performance. Leveraging automation technologies and employing specialized staff leads to a level of efficiency unmatched by any company or institution with a small number of self-managed servers. Relying on a CSP, however, is not always the best option. It leads to a high level of dependency on the CSP with many downsides. Especially with only a few major commercial players in the market, they could charge high prices or threaten to deplatform services if they dislike the content. Although dynamic network architectures theoretically enable switching to a different provider, it can still be a tremendous operational risk. Therefore it is crucial to research technologies that enable institutions to improve the management of their own infrastructure.

Besides leveraging efficient technologies, a vital part of improving network infrastructure management is the technological understanding of the infrastructure by different management parties. A visualization of the computer network can help understand its infrastructure and its enabled management policies. Many people find it easier to grasp a complex concept if they are not only told an abstract description of a concept but can see a visual representation of it. If many services are offloaded to

automation and divided among administrators with different assignment profiles and permissions, a visualization tool might also bridge the gap between specialized tasks and the bigger picture. Another important aspect of leveraging a new network management technology is the validation of its functionality. Network management is a complex task, and failures might lead to serious information security and reliability problems. In the case of using a network orchestrator, the trust might be increased by a visual representation of the network infrastructure data. This representation can then be used to observe the behavior of the orchestrator and validate that the orchestrator implements the given instructions. A visualizer might also be helpful to developers of the network management tool to discover flaws and errors in the implementation.

1.1. Thesis Objective

The Service Function Chaining (SFC) architecture enables fine-grained orchestration of Virtual Network Functions (VNFs) and is a valuable technology for modern 5G-enabled networks. SFC offers services like scheduling VNFs and tailored network solutions to administrators and customers [1]. Because the SFC architecture only recently reached technical maturity, just a few visualization tools are available. The bwNET2020+ project funded by the Ministerium für Wissenschaft, Forschung und Kunst of Baden-Württemberg supports the expansion and modernization of the BelWü network and other university networks [10]. Supported by this project, a P4-SFC orchestrator architecture was proposed [66]. This thesis aims to build a web-based visualization tool to visualize SFC infrastructure like the network topology managed by the P4-SFC orchestrator.

1.2. Thesis Structure

This bachelor thesis has the following structure. Chapter 2 explains the technical foundations and terms necessary to understand the web-based visualizer and the SFC architecture. Chapter 3 presents related works. Chapter 4 analyzes considerations and first decisions on the visualizer implementation. Chapter 5 describes important details of the implementation and explains decisions considered in the implementation. Notable features of the web application are covered and characterized. Chapter 6 discusses and evaluates the implemented project. Chapter 7 summarizes the thesis project and its relevance to modern network management.

2. Technical Foundations

The following chapter explains the technical terms and foundations necessary to understand the visualized architecture and the concept and implementation of the web-based visualizer.

2.1. Software Defined Networking

In a traditional computer network, data plane and control plane are working combined on a switch or a router. Software Defined Networking (SDN) is a network architecture design paradigm that separates the data plane from the control plane in network devices [8]. The control plane defines and controls how network traffic is processed and forwarded by configuring the data plane [31]. The data plane does the actual forwarding. The control plane communicates with the forwarding plane through a Control-Plane Southbound Interface (CPSI) [31]. Protocols used to implement the CPSI are for example Openflow, Open vSwitch Database (OvSDB), Border Gateway Protocol (BGP), Path Computation Element Communication Protocol (PCEP), Network Configuration Protocol (NetConf) [40]. The control plane communicates with the applications via the Northbound Interface, leveraging, for example, REST Application Programming Interfaces (APIs) [40]. Communication between different control plane instances is also possible through west- and eastbound interfaces, usually with gateway protocols like BGP in the case of a westbound interface [59].

The Openflow protocol was used in SDN networks to correspond with the data plane installing flow rules on hardware devices [1]. Over time programmability has increased, allowing operators to define packet processing and pipelines with high-level languages like P4 [50]. SDN controllers maintain a global view of the infrastructure and help to steer traffic between network functions and scale across multiple hardware devices. SDN controller can be used to implement centralized network control, making it easier to manage and deploy an extensive network [1]. This can become a benefit when changing to an updated or different forwarding control implementation. Because changes can be made in one place for the entire network. Instead of configuring many small devices separately, they can be configured by changing a policy in one place. It makes network configuration more efficient and more accessible and leads in the process to better and less error-ridden network management [8].

Because of the growing demand for cloud services, the concept of SDN is more relevant than ever. Especially in this context, SDN offers significant advantages compared to traditional networking. The functional separation of data plane and control plane enables far more flexibility and complexity in the programming of the

control plane because it can be separated from the hardware instance. This can improve the computing performance of the forwarding hardware device with only the data plane functionality to process. Overall, SDN provides more network management precision which can also be helpful for network function virtualization [8].

2.2. Network Function Virtualization

Network functionality consists of end-to-end services. Network Functions (NFs) process and forward the traffic in a computer network. In traditional network management, the architecture of these network services is bound to topology requirements and static. Virtualization is a technology that provides deployment flexibility by splitting and allocating physical resources of computing systems. That is done by a hypervisor, a piece of software running on the host machine. The hypervisor creates and manages virtual guest machines on the host machine. This enables abstraction of the system software from the hardware device and mainly provides significant advantages regarding capacity optimization and scaling [60]. A common virtualization practice is server virtualization using virtual machines or containers [6]. Network Function Virtualization (NFV) decouples network functions from the physical network topology. Leveraging virtualization technologies and commercial off-the-shelf (COTS) programmable hardware, NFs can be run and be configured on another device than the location where they are applied [33].

NFV enhances network service provisioning flexibility and enables a faster transition to improved network practices. Along with better overall management and control, this architecture is advantageous because there is a shift to massive and growing computer networks. This also means the number of new proprietary devices facilitating coupled network functions would make it harder to manage a network. Therefore decoupling the network functions from the physical devices can provide a great benefit [72].

NFV platforms consist of different software and hardware components. Many different existing NFV platforms generally consist of three components. The NFV Management and Orchestration plane (MANO) is responsible primarily for service provisioning. It consists of components to manage and orchestrate VNFs and virtual infrastructure. The service plane contains VNFs ordered in service chains to realize network services. The NFV infrastructure is the collection of all hardware and software components needed to fulfil those promised services [73]. The European Telecommunications Standards Institute (ETSI) is an independent nonprofit standardization organization for the telecommunications industry. The ETSI NFV Management and Orchestration Group Specification (ETSI MANO) defines an NFV platform framework for configuration and provisioning of VNFs and managing the underlay infrastructure deployment [18]. SDN and Network Function Virtualization are fundamental concepts for the deploying of SF chains in a service provider network [1].

2.3. Overlay Network

Overlay networks are networks built on top of existing networks to add a virtualization layer. They can also change the properties of the underlying network and alter network layer structure [27]. With a changing paradigm in network architecture towards more agile and cloud architecture [8], network functions can be decoupled from topology constraints by an overlay network which serves as a platform for virtualized network functions [58].

2.4. Underlay Network

The underlay network is the physical network infrastructure that serves as an underlying platform for the virtualized overlay network. It is mainly responsible for forwarding traffic and providing connectivity between virtualization edges (NVEs). An NVE is a network entity at the edge of the underlay to the overlay network, which can implement hardware layer (L2) and network layer (L3) functions [39].

2.5. Service Functions

Network functions provide services like management, control, security, and connectivity of a network. In traditional networking, these functions were connected by physically connecting the hardware devices where they are located. NFV enables virtualized network functions without the former topology constraints. Which can be deployed independently at various locations in the physical network topology and can be split into smaller Service Functions (SFs) according to [38]. In NFV terminology, SFs are also known as VNFs [46].

A service function is responsible for the specific processing of received packets and can act on various network layers. Multiple instances of the same service function can exist in one administrated network. A Service Function (SF) can be, for instance, a firewall, or a load balancer [58].

2.6. Service Function Chaining

Network services can consist of multiple SFs and often need to be applied in a particular order. Service function chaining is the ordering and composition of these SFs into SF chains, ordered sets of abstract SFs and policies being applied to network packets [58]. To manage a virtualized topology, specific chains can be implemented for different traffic categories. The SFC paradigm can be a valuable tool for automating the management of virtual network services. It decouples the forwarding process from the underlay network [66].

SFC is an architecture that makes a more dynamic traffic steering approach possible. The deployment of new services can be increasingly difficult in static architectures because it can lead to a cascade effect [58]. That means that changing elements of

the service function structure can affect other elements of the construction, which again can affect other elements themselves. Using Service Function Chaining (SFC), SFs can be directed and chained flexibly without these concerns. It allows us to implement complex structures and manage our traffic more precisely. Different research groups have been putting forward SFC architecture models. Most notably is the SFC architecture proposed by the Internet Engineering Task Force (IETF) [32], which is presented in the following paragraph.

SF chains are deployed in a SFC-enabled domain with clear boundaries. Traffic steered through a specific SFC is first classified and then SFC-encapsulated by the service classification node at an ingress node to the network. The traffic type of a SFC is stated in the corresponding SFC policy. The SFC encapsulation provides any information needed to forward the traffic along a Service Function Path (SFP). An SFP for a corresponding SFC is instantiated to route the classified network packets to instantiations of the SFs stated in the SFC policy in the order stated in the SFC policy. For each SF an Service Function Instance (SFI) of it is deployed. Packets leaving the SF-enabled domain leave through an egress node, where every information about the SFC is removed, before forwarding the traffic out of the network [32].

The SFC infrastructure might contain SFs that are SFC-aware or SFC-unaware nodes. SFC-aware nodes can process routing information and SFC-encapsulation from packets. SFC-unaware nodes need to receive the traffic without encapsulation. In order to do this, SFC proxies are used to decapsulate SFC encapsulation from SFC traffic and forward the content to SFC-unaware node. The SFC-proxy stores the SFC-encapsulation and reapplies it after the traffic leaves the SFC-unaware SF. The SFC-enabled domain also can contain Service Function Forwarders (SFFs). A SFF is able to read the routing information from the SFC encapsulation of a packet and can forward it accordingly. Figure 2.1 shows the basic SFC architecture.

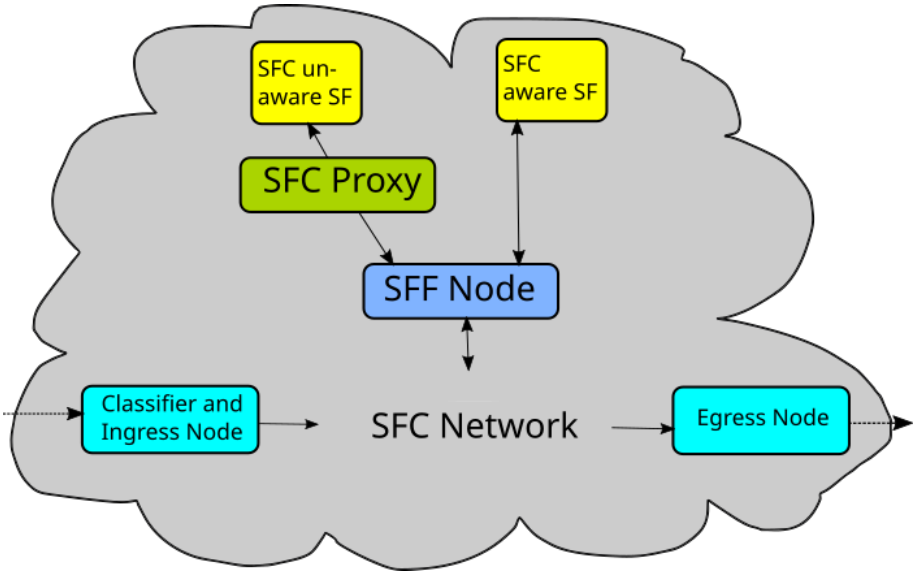


Figure 2.1.: Basic SFC architecture

A service orchestrator is a way to manage network services. It can be used to deploy virtual machines in and manage a virtual overlay network according to a deployment plan. It also is responsible for ensuring that the requirements for each network service are fulfilled. An orchestrator is provided with the topology information of the underlay network and can be configured to provide, optimize, and manage specified network services [61]. SFC can be implemented in an orchestrator-based approach. To manage SFC infrastructure, an orchestrating entity needs data about the network topology and SF chains. Information bases about the underlay, overlay network, and Service Function Paths (SFPs) are necessary to provide this information.

In [38], a Service Function Path Management entity (SFP-ME) establishes, deletes, and updates a SFP with the information provided by a SFC policy. For the instantiation of an SFP, the Topology Information Base (TIB) and the Service Information Base (SIB) are needed. The TIB (see chapter 2.7.1) stores essential information about the physical network infrastructure [38]. The SIB stores the information about the service function in the network further explained in chapter 2.7.2. The sequence of traversal is stated in the SFC policy [38].

In order to deliver packets to specific SFs, there needs to be a data-plane representation that specifies a forwarding path. In the SFC architecture in [57], a Network Service Header (NSH) is proposed. An NSH is a network header that contains information about the current SFP, the location within this SFP and other metadata [57]. An example of SFC using MPLS-based Segment Routing (see chapter 2.5.3) is presented in [66]. The architecture is further explained in chapter 2.5.4. SFC can be part of an intelligent service orchestration scheme managing many different SFs. That can be adequately archived by leveraging an architecture approach with SFC and SDN with integrated VNF [1].

2.6.1. Topology Information Base (TIB)

The topology information about the underlying computer network is used to optimize and establish the SFP and the deployment of the SFs [38]. According to [38], each entry for a service function node of the TIB structure consists of the following entries: name of the service function node (SF node) with identifiers (for example, MAC addresses, ids) and additional information, the number of neighbor nodes and the neighboring nodes with identifiers [38].

2.6.2. Service Function Information Base (SIB)

The information about the SFs again is stored for establishing and updating the SFP. According to [38], each entry for a service function in the SIB structure consists of the following entries: an index for a quick lookup and a unique identifier for each SF. An SFC locator entry states the location, such as a service function node used as a host for deployment. A SF node can be the host of multiple SFs. Each SF can have multiple different hosts [38]. The SF type entry is used for the possible classification

of service types and can refer to the provided service [38]. The SF capacity entry determines the traffic processing capacity [38]. A SF load field indicates the current traffic load of a service function. A SF status field shows the current status of the SF and can, for example, signal availability, or unavailability [38].

2.6.3. MPLS-based Segment Routing for SFC

Multiprotocol Label Switching (MPLS) is a forwarding technology that adds labels to a network packet to encode forwarding procedures. Source routing is the process of specifying a route of a a packet on the data-link layer [55]. MPLS can be used for source routing. By adding an entire MPLS label stack to a network packet and reading the top-most label and altering the stack at each hop, the next hop can be identified and complex forwarding can be implemented. Actions taken to alter the MPLS label stack are usual stack operations like popping, swapping, and pushing MPLS-labels onto the stack [22]. Traffic is forwarded along a label-switched path by basing routing and forwarding decisions on MPLS-labels. In [22], a logical NSH can be represented by MPLS. This is archived by altering the MPLS stack at each SFC hop [22].

Segment Routing (SR) is based on the source routing paradigm and often utilizes MPLS or IPv6 to steer traffic through an ordered list of segments. Segments are routing instructions like passing packets through SFs or forwarding them along a certain path. SR introduces a mechanism that can restrict a network flow to a specific topological path at an ingress node to the SR domain [26]. The SR technology can be leveraged to realize SFC policies in SFC infrastructure. It can be used, for example, to encode an SFP in the MPLS stack. Which can be archived, for example, by pushing MPLS labels encoding an SFP to the MPLS stack at the beginning and then popping an MPLS label after each step of the SFP [66].

Analogous to a SFC domains, there can be SR aware and SR unaware services in a SR domain. An SR aware service can process the SR information represented, for example, by the MPLS stack, while a SR unaware service cannot process the stack. In this case, a SR proxy is used to encapsulate the packet or provide other modifications like detaching the SR information from the packet and attaching it later on [11].

2.6.4. P4-SFC

P4 is a programming language used for configuring data-plane devices in a computer network [50]. In [66], a P4 service function chaining architecture is described, which uses a P4-capable ingress switch. Using P4, the SFC paradigm is introduced by configuring the switch with MPLS-based segment routing as an ingress switch. The switch functioning as an SFC ingress node classifies incoming traffic. Each service function chain is matched with a specific flow. In the next step, a corresponding MPLS label stack is pushed to the packets, encoding the forwarding route of the traffic and the specific service function instances. Apart from the ingress switch and

possible other switches configured to do the same, the network's hardware nodes and SFs remain unaware of the SFC. Service function instances in the network establish a logical network [66]. They are reachable from a corresponding switch which adds a Virtual Local Area Network (VLAN) tag pointing at the SFI if the top-most label of the MPLS stack is an MPLS label pointing at the SFI and forwards it to its hardware host [66]. Forwarding within virtual machines and containers is possible using the Linux kernel to forward the IP packet. The label stack is stored and added afterwards before sending the packet through a corresponding egress interface [66].

After the traffic traverses the SFI, it is sent back from the host to the switch in the VLAN. The switch then detaches the VLAN tag and pops the next segment label from the label stack [66].

The P4-SFC Orchestrator architecture [66], utilizing the P4-SFC paradigm, manages an overlay network by deploying SFs on hosts, computing SFC paths, and implementing and terminating SF chains. Different groups can access it with different permissions and purposes. For example, it is possible to let customers have individual SFs and specify SF chains for the owned SFs while allowing administrators access to configure the network [66].

2.7. Importance of NFV/SFC in 5G Networks

Network slicing is the process of defining slices in a network dependent on the needs of different customers. Each slice has different guarantees, for example, regarding latency, bandwidth, or mobile broadband. The advantage of this process is that network slices can be provided to the end-users and tailored to their needs. To enable network slicing, techniques like SDN and NFV have to be embedded [67].

SFC is relevant for Beyond 5G networks, because it leads to efficient service delivery. 5G networks are characterized by low latency and programmability. VNF, SDN and SFC can provide tailored solutions for different customer demands. NFV can help deploy services for Radio Access Networks (RAN) and mobile core networks. NFV can enhance 5G RAN and reduce capital expenditure for telecommunications service providers (TSPs). SFC enables GI-LAN mobile core networks and administrational and customer services in inter and intra-datacenter networks. Challenges to 5G networks are efficient scalability of VNFs, maintaining guaranteed VNF performance while supporting the deployment of hardware and VNFs. They can be significantly improved by architectural frameworks leveraging VNF, SDN and SFC [1].

2.8. Internet Control Message Protocol

The The Internet Control Message Protocol (ICMP) is a network layer protocol used primarily for error reporting and network connectivity analysis in a computer network and between networks. ICMP is typically used for error reports concerning datagram processing and is not designed for reliability. For example, when two devices are

connected over the Internet, and a router drops a sent packet because it is too large, the router sends an ICMP error message to the original source [37].

ICMP can be used as a traceroute tool to trace paths in the data link network topology utilizing ICMP messages. In this case, the actual routing path between two physical devices is determined [43]. It is also used for Neighbor Discovery by sending router/neighbor solicitation and advertisement messages [70].

2.9. Network Discovery Protocols

Internet-layer Network discovery protocols are a fundamental part of IPv4 and IPv6 network connectivity. They operate below the network layer and establish bindings between the IP address of an device on the network layer and the Media Access Control (MAC) identifier on the data-link layer. They keep track of known devices and their mappings in a computer network using a cache. The Address Resolution Protocol (ARP) is used in IPv4 networks and replaced by the Neighbor Discovery Protocol (NDP) in IPv6 networks. IPv6 NDP is additionally the corresponding protocol to a combination of ARP, ICMP Router Discovery, and ICMP Redirect in IPv4 [70].

To connect devices on the hardware level, they have to communicate with each other via a certain configuration exchange protocol providing link-layer network discovery. There are many proprietary network discovery protocols on the link-layer. While notably the LLDP is vendor-neutral and announced as a a standard protocol by the IETF [5].

2.9.1. Address Resolution Protocol

ARP uses a simple message format with a request and a response option. That might work as follows. Suppose a computer receives an IP packet destined to another device in the same Local Area Network (LAN). In that case, it looks for a corresponding MAC address for the destination address of the received message in its ARP cache. If it cannot find a suitable mapping, the device broadcasts an ARP request to all devices in the LAN. If a device receives an ARP request for its IP address, it responds with an ARP response message and may add the data about the other device to its own cached ARP table. This message contains the IP addresses and the MAC address of the device. When the requesting entity receives the message, it adds a record to its ARP cache mapping IP addresses and the MAC address [54].

2.9.2. Neighbor Discovery Protocol

The Neighbor Discovery Protocol (NDP) uses advertisement and solicitation messages to determine neighboring hosts and switches [7]. The protocol solves the following network connectivity problems: router, parameter, and prefix discovery, address autoconfiguration, address resolution, next-hop determination, neighbor unreachability detection, duplicate address detection, and redirection. NDP does this by defining

five different ICMP messages, caching link-layer, and IP address mapping. Following ICMP packet types are part of NDP: Router Solicitation, Router Advertisement, Neighbor Solicitation, Neighbor Advertisement, Redirect.

When an interface is enabled, hosts may send Router Solicitations messages to signal the router to send out Router Advertisement messages before the next scheduled time. Router Advertisement messages are either scheduled or sent after a received Router Solicitation message. Routers generate and send these messages to advertise their presence in the network, related links, Internet parameters and prefix data. Neighbor Solicitation messages are sent by a network node to request information about the connectivity status and link-layer address of a neighboring node and can also be used to detect duplicate addresses in the network. Neighbor Advertisement is the response to a Neighbor Solicitation and may also be sent unrequested on a link-layer address change. Received data about the network is cached in various caches and lists for each interface [64].

2.9.3. Link Layer Discovery Protocol (LLDP)

The Link Layer Discovery Protocol (LLDP) allows devices in a network to communicate device information to their immediate connected neighbors in a link-layer network. Network nodes send information about them like MAC addresses and system names and listen for data on the devices of each open connection. These messages also contain the endpoint address a protocol like the SNMP would require. From the received information, each device in the network can calculate a picture of the network and the neighboring nodes it can reach [5].

2.10. Simple Network Management Protocol

The Simple Network Management Protocol (SNMP) puts management agents on network elements like hosts, terminals, and gateways. The task of these agents is to execute network management functions demanded by network management stations [23]. The architectural structure of SNMP also contains a Management Information Base (MIB) with all the gathered information about the network. The data is stored as objects and represents the network elements and corresponding data. The variables in the MIB can be made fully and partly accessible to different management entities. The network can be configured by changing the variables in the MIB and instructing the corresponding agent to change the device configuration accordingly [23].

2.11. Label Distribution Protocol (LDP)

In a network that uses MPLS, Label Switching Routers (LSRs) have to negotiate the meaning of MPLS labels to forward traffic. A common understanding between all parties is archived by using the Label Distribution Protocol (LDP) [69]. LDP does this by message exchange of certain message types. Discovery messages announce and

maintain the status of an LSR in the network. Session messages establish, maintain, and terminate sessions between LDP entities. Advertisement messages advertise changes in labels and their mapping to forwarding behaviour policies. Notification messages are used for signal error and advisory information.

2.12. Web Application Architecture

The World Wide web was promoted as a virtual library by Tim Berners-Lee. Early websites were often just static HTML pages linked together. With the Web becoming increasingly dynamic, the term web application emerged. A web application presents dynamically calculated content based on request parameters, user behaviour, and security concerns. It is an application that can be accessed by a client using a web browser and connects interactively with servers over a network [41]. A web application often operates on the client/server paradigm and has two structural components: a frontend and a backend. The client/server paradigm means servers receive client requests and then execute server-side code to generate a server response. The server response contains client-side code that is run in the browser to generate the requested content or a web page with an error message. The term frontend often refers to client-side component of the application including the browser and the web application client, while the backend is the server-side component that facilitates services, including data storage, validation, authorization, authentication [68].

Backend and frontend are interconnected through APIs. To interact with a website, the user utilizes a browser to send a client request to the webserver of the web application. The webserver then calculates the application's client-side code, possibly requesting other services in the backend, for example, a database. After calculation, the code is then sent as a server response back to the browser [42]. Various architectural and design paradigms exist in modern web application development. Notable is a trend towards modular web application architectures. Important web application architectures for modern, responsive web design are the Model-View-Controller Architecture and the Three-tier Web Architecture [36].

2.12.1. Model-View-Controller Architecture

The Model-View-Controller (MVC) architectural paradigm decouples the user interface from the data and the logic of the web application [36]. The MVC architecture consists of three parts. The model component determines the data structure of the web application. The controller component defines the calculation and control logic of the web application. It serves as a bridge between the other parts of the architecture and manages, for example, API communication between the application components. The view component defines the user interface [42].

2.12.2. Three-tier Web Architecture

One of the most common models for web architecture is the three-tier architecture that also adopts the MVC architectural pattern. This architecture divides into

presentation, data, and logic tiers. The logic tier and the presentation tier are connected through a dedicated API, and the data tier is accessed by the logic tier. Similar to the MVC architecture, the presentation tier displays web content and provides user interactivity. The logic tier conducts operations like processing and data exchange between the other layers and the data tier is responsible for data persistence [42]. The figure 2.2 illustrates this architecture.

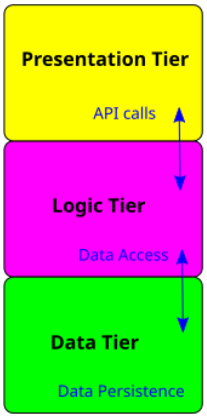


Figure 2.2.: Three-tier architecture

2.12.3. REST Architecture

The Representational State Transfer (REST) architecture style is a set of architectural constraints including the client-server (CS), stateless (S), cache, layered systems style (LS), and code-on-demand style. Client-server means separation of concerns into user interface and data storage concerns. It simplifies server components improving scalability and portability, and allows components to evolve independently. The stateless constraint requires communication to be stateless. That means a client request contains all the data necessary for the server to understand the request resulting in no stored state data. This improves the reliability and the scalability of a system and makes monitoring easier. It may decrease performance because more data needs to be sent for each request. The cache constraint improves network efficiency. Caching means storing and reusing some response data, increasing efficiency and user-perceived performance.

The uniform interface constraint of REST architecture is a central feature and refers to a uniform interface between all components of the REST architecture. It simplifies and standardizes the architecture overall but has degraded efficiency as a tradeoff. The layered system (LS) style introduces hierarchical layers to enable load balancing, increase security by encapsulating legacy clients and decrease system complexity. The disadvantages of the LS architectural paradigm are added overhead and latency. The code-on-demand style (COD) is an optional constraint within REST. It allows clients to download and execute code in the format of applets and scripts. It reduces the number of features clients need to have implemented but also

comes with a caveat of more challenging monitoring [25].

REST interfaces are efficient for large-grain hypermedia data transfer and optimized for common web architecture use cases. REST has become a de facto standard in the Web and commonly uses Hypertext Transfer Protocol (HTTP) for message transfer. It relies on Uniform Resource Identifiers (URI) to facilitate resource detection and interaction. These URIs provide the location and name of the resource. The action to perform on the resource is defined using HTTP verbs like GET, DELETE, and POST, among others. REST is a common design choice for exposing service APIs [48].

2.13. Full-Stack Web Application

A web application stack is the collection of critical software services needed for developing web applications. It is a subtype of a collection of software performing specific tasks called solution stack. As a minimum, a web application stack includes an operating system, data storage software, programming language, and a webserver. Full-stack web applications meet all the requirements of the web application stack. Full-stack development involves client-side and server-side programming, and often a data server. The client-side is often written using HTML, CSS, and Javascript. In recent years frontend frameworks have often been leveraged. Backend frameworks are widely used for the server-side implementation. Data servers can use databases that either are SQL based like SQLite or NoSQL based like ElasticSearch and MongoDB [42].

2.14. Frontend Frameworks

The frontend component is essential to the service performance and user satisfaction of the web application. Software frameworks are classes and libraries providing a collection of functions for software development [63]. Frontend frameworks are useful building blocks for clean and performant web applications. Notably, three open-source frontend frameworks have become popular in recent years. They are all built on top of Javascript, HTML, and CSS [71].

2.14.1. VueJS

Vue is JavaScript ES6 based and was developed by Evan You in 2014. The intention was responsive data binding and UI components connected through a straightforward API. Vue was built with limited functions, especially for single-page applications and therefore not the best choice for commercial usage. But the open-source community and third parties provide packages and libraries to create complex web applications [71].

2.14.2. ReactJS

React was developed by Facebook to increase the user experience of the websites of Facebook and Instagram and was released as an open-source Javascript ES6 library in 2013. React creates website components and mounts them into the Document Object Model (DOM). That makes websites significant faster. When the user data of an HTML website changes, the entire page needs to be re-rendered. With React, only changed components have to be re-rendered [71].

2.14.3. Angular

Angular was developed by Google in 2010 and has since been improved multiple times. It started out being Javascript ES5 based and is now Typescript based. The core concept of Angular 1 was two-way data binding in web browsers reducing the backend data processing. Angular 2 introduced event binding between a module called component and its view [71]. The latest Angular version was released in November 2021 [29].

Newer Angular versions have many convenient features. Basic concepts of Angular are modules, components, services, and dependency injection. As building blocks of the framework, *NgModules* combine code and components into functional modules. Components are classes containing logic, data, and a view defined by an HTML template. Services are sharing data between components. Dependency injection enables the injection of services and functions into components without injecting all the dependencies separately [29].

2.15. Backend Frameworks

The backend component provides multiple services to the web application. APIs have to be protected against external attacks. Users have to be authenticated. Seamless but secure access to the database needs to be established. User requests have to be handled, and corresponding data needs to be stored. Backend framework can enable these services in an updated manner without building everything from scratch [63]. Advanced backend frameworks drastically decrease development time and can ensure that basic development requirements are met.

According to a survey in 2021 [65], the most commonly used dedicated backend frameworks are Express, Flask, and Django. Flask is a lightweight web development framework based on the Python programming language that does not enforce any dependencies or project layouts [52]. Express is designed to be a fast and minimalistic framework for Node.js and has many HTTP utility methods [49]. Django is a high-level Python framework for secure and scalable backend development [16].

2.15.1. Django

Django is a versatile web framework commonly used as a backend framework. It takes care of tasks like user authentication, site maps, content administration, Really Simple Syndication (RSS) feeds, and is used in famous web applications like Udemy, Instagram, and applications from Mozilla. It follows the MVC architecture. The model component in Django interacts with the website interface and the database [63].

3. Related Work

In this chapter, projects related to SFC Infrastructure Visualization are presented. Network Visualization is a broad field in computer science. In the context of computer networks, Multilayer Network Visualization is especially interesting. In [45], the state of the art of Multilayer Network Visualization is discussed. SFC Tracerouting is essential for the validation and debugging of SFC orchestrator implementations. Network Oberservability plays an important role in managing modern Cloud Native Infrastructure. Visual SFC Network Tools, however, just started to emerge and are a vital tool to understand SFC infrastructure.

3.1. Computer Network Visualization

There are different approaches to the visualization of complex network structures. Notable are especially multilayer network visualization concepts. Layers in a multilayer visualization can be defined by a modelled system or even a physical reality [45]. In the case of visualizing a computer network, there already are layered structure models to use as guidance. The ISO-OSI reference model describes an architecture of seven layers, while the TCP/IP protocol stack proposed by Cerf/Kahn only has five layers 3.1. The topmost layer of the TCP/IP stack is the application layer split into the application, presentation and session layers in the ISO-OSI model. Then comes the transport layer with protocols like UDP and TCP [51]. Interesting for the visualization is especially the network layer and the data-link layer. The network layer is responsible for routing traffic between different networks, while protocols on the data-link layer manage traffic between entities in a network. However, the separation between multiple layers is not always clear cut. There is a need for protocols to bridge the edges between the layers [51].

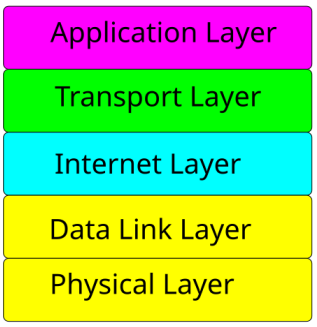


Figure 3.1.: TCP/IP stack

The idea to visualize computer networks as traffic dispersion graphs was proposed in [35] and later implemented in [28]. In [28], an application with a Java-based frontend and a SQL database was implemented and used to calculate computer network visualizations from network traffic. This tool intends to monitor traffic with computer network graph visualization.

3.2. SFC Tracerouting

Tracing the route of network packets in the SFC topology is a crucial part of validating and verifying SFC architecture implementation and configuration. In [21], a SFC Path Tracer was proposed and implemented. It provides a SFC encapsulation agnostic solution by generating probe packets that traverse implemented chains and have their route being mirrored to the trace tool each time they are sent to the next hop by a SF forwarding switch. SF paths can be identified, and SFC policy implementations can be validated by comparing the traceroute data with the given instructions.

Other tools able to troubleshoot SFC architecture are SDN tracerouting tools. SDN traceroute was introduced as the first tool for this use case. Its design is based on the ICMP traceroute tool and leverages probing packets. The tool installs high-priority rules in every SDN switch to trap probes sent by their neighbors and forward them to the SDN controller. The packet content is then recorded. Afterwards, the probing packet is changed, reinjected, and sent to the next hop [2]. Track is proposed as a tracerouting tool for SDN networks. It can trace service chains in SDN-enabled networks. It can be used without knowledge about involved NFs and does not pollute internal states of NFs [74].

3.3. Observability Tools for Cloud Infrastructure

In June 2020, an open-source tool called Suzieq was released. It is the first open-source application for a multi-vendor network observability platform [20]. It was introduced to help network engineers and designers improve the understanding of their network. Suzieq uses an agentless model to gather data from routers, bridges, and Linux servers transported via SSH and REST API. Platforms from many different vendors, SoNIC devices, and Linux servers are supported. In the first step, the data is normalized into a vendor-agnostic format. The network analysis then is exposed via GUI, REST API, CLI, or Python and can be rendered in various formats like JSON or CSV. In the latest release, the pandas package is used as an analysis engine [20]. The GUI of Suzieq has the ability to show the status of entities in the network and the path between two IP addresses, even through an EVPN overlay. It visualizes the path by linking rectangles representing hops with arrows and can be a helpful tool to find irregularities in the connection between two network components [12].

Prometheus is an open-source project that joined the Cloud Native Computing Foundation in 2016 and is their second project after Kubernetes. The Prometheus tool stores and collects metrics data about network topology. It can be linked with

Grafana to export and visualize the data [56]. Grafana and Prometheus are part of a network observability stack. Grafana provides data visualization, many different queries, and monitoring alerts. It is a very popular open-source Github project but also has a commercial version [30].

3.4. Visual SFC Network Tools

The NFV paradigm, in comparison to more established network architecture practices, only reached technical maturity recently. Implementing SFC architecture still requires a great amount of detailed knowledge about the technology. Because of that, there are only a few visual SFC network tools [9]. A prototype of a tool for the visual construction and validation of SFC Packages was implemented in [9]. In this context, a SFC Package is a package containing information about a SFC, including a list of service functions, a forwarding path, and deployment requirements. The GENEVIZ tool prototype [9] enables users to create these packages on a visual platform. Then based on this data, a SFC policy can be implemented.

The option to store a hash of the SFC Package in a blockchain was added to provide data integrity and validate implemented Service Function Chainings (SFCs). It is supposed to be used to verify the integrity of SFC Packages by establishing a decentralized, cryptographic, and therefore supposedly trustworthy chain of immutable records [24]. However, a question notably to ask in this context is if the qualities decentralized and immutable are even beneficial for SFC policy validation. Tampering with SFC policies could be a possible way for an attacker to access unauthorized resources. But if the blockchain validation of SFC policies can be somehow bypassed, there is no real benefit in leveraging a blockchain.

The use of decentralization instead of having one trusted authority is also debatable when having a blockchain for a SFC domain. It is questionable where the validating blockchain instances should be stored to regularly synchronize while the majority of them remain safe from attackers. The aspect of decentralization and immutability might also become a major downside if the SFC policies should remain private [24]. In the case of SFC architecture, a database validated by one trusted authority might be a better and more resource-efficient way to keep track of the SFC policies.

In [34], a self-service portal for P4-SFC architecture was implemented that allows users to configure SF chains with a simple drag-and-drop mechanism. It includes a user and permission model to give only authorized users access to implement specific SFC policies. An inventory provides the VNFs for the SF chains. They can be predefined by the administrators or configured by the authorized user [34].

4. Web-based Visualizer Concept

The following chapter presents questions and possible solutions in the process of designing and implementing a web-based SFC visualizer.

When designing a web-based visualizer, it is essential to know what services the visualization system should provide and in which situations it is leveraged. In this case, the constructed visualization system should represent a SFC network topology orchestrated by a SFC orchestrator. It should depict the SFC infrastructure so that a user with an administrative assignment can understand it better and, if necessary, discover problems made in the instructions given to the orchestrator.

It should also be a tool to quickly find basic information about network infrastructure without much prior knowledge about the infrastructure needed. The SFC orchestrator implements and manages the network and how incoming packets are processed on different layers. Since it operates based on the knowledge of the physical network topology of the underlay network [38], the visualization system should display the network topology on the data-link layer. Especially the ingress switch and service function hosts are interesting. However, other hardware like switches and routers are also relevant to the SFC topology.

Because SFFs can directly run on hardware switches and hosts [66], even non service function hosts are of interest. So the data about SF hosts, ingress nodes, all hosts and switches used as intermediary forwarding nodes and how they are connected needs to be obtained. In addition, virtualized network components such as SFs and SF chains should be represented. For the representation of the overlay layer, data about the deployed Service Function Instances (SFIs) like deployment location and data about the deployed SFs is necessary. Furthermore, for the SFC visualization, detailed information about all the implemented SF chains and how one or more corresponding SFPs are instantiated is needed. That is because service functions can be run on multiple SF hosts and SFPs calculated from SF chains might change depending on the available SFIs.

Because the tool should assist debugging and understanding what is happening in the network, not only from a theoretical standpoint, the information about the SF chains and SFs with data and depiction of their instances have to be combined. There needs to be a combination of the abstract SFC notion with the instantiated Service Function Path (SFP).

Different layers must be visualized separately to provide a clear picture of the network topology in all areas. However, there should also be visualization elements

to show the relation of components across different levels, for example, SFIs and their relation to their SF hosts. In the case of many instances or SFIs on many different hosts, it can become unfeasible to represent a clear picture of the topology and all the essential information. An abstracted representation might minimize this problem. Therefore, functions to calculate an abstracted depiction should be implemented to visualize the overlay network and SF chains that play a critical role in the functionality of the SFC topology. An essential feature of a visualizer used for debugging is switching between the bigger picture and details. Detailed information about selected network components should be provided on demand to accomplish this goal.

For the web-based part of the visualizer, constructing a full-stack application and embedding the visualization system into a frontend framework would be a viable solution. An alternative approach would be implementing the frontend without a framework. This could be a more lightweight solution but would take much more time and is more prone to mistakes. The application should provide the following services to enable SFC infrastructure visualization. Backend and data storage services are used to obtain and process the corresponding data and store it somewhere to visualize the network. A frontend service defines the viewscreen presented to the users. The backend needs to communicate with the frontend and the data storage service to provide the frontend with the necessary data to calculate the visualization. Full-stack applications include a backend, a frontend and a database implementation.

Further details about aspects of the web-based visualization concept are characterized in the following chapters below.

4.1. Data Sources

There are many ways to obtain the data for a data-driven visualization. The SFC architecture paradigm already includes information bases for establishing a service function path. The Topology Information Base (TIB) provides information about the SF hosts and their neighbours. The Service Information Base (SIB) holds data about the service functions and their location on SF hosts [38]. The Service Function Path Management entity (SFP-ME) in [38] also holds a service function chain-path base (SFPB) with information about service function chain-path mapping. This contains useful information about instantiated SFPs and their corresponding SF chains [38].

Apart from that, there are many different tools and protocols to get data about network topology. This can become very important if the incentive is to use the visualization system to debug SFC instructions and verify if instructions are implemented as planned. Validation of the orchestrating setup could be a useful addition to the functionalities of the visualization tool. A visual side by side comparison of network instructions held by the orchestrator and their implementation is possible and an excellent asset for debugging.

In the case of the underlay network, a first look can be taken at some common protocols already used in almost every modern computer network. Classic discovery protocols are interesting for the data link layer visualization. That includes the Neighbor Discovery Protocol (NDP) [64] in case of IPv6 and the Address Resolution Protocol (ARP) for IPv4 [54]. Both maintain a cache with data about devices in the corresponding network. Further explanation was provided in section 2.9. Other typical protocols to obtain data about the underlying network infrastructure can be described as network control and management protocols.

The The Internet Control Message Protocol (ICMP) is used for neighbour discovery, as explained in chapter 2.8. Direct links between hosts can be discovered by getting data about neighbouring nodes in a network. It is also possible to use the ICMP traceroute tool to trace paths in the data link network topology utilizing ICMP messages [43]. However, traceroute is not always applicable to get path information. Load balancing is a common practice today. Using strategies like equal-cost multi-path routing (ECMP) can result in distorted topology information [44].

A commonly used protocol to get neighbour relations of network entities is the Link Layer Discovery Protocol (LLDP). LLDP provides data that enables devices in a network to calculate a picture of the network and reachable neighbouring nodes [5]. More details about the protocol were brought up in chapter 2.9. Information about each managed object in a network could also be deducted from the corresponding entries in the Management Information Base (MIB) of the Simple Network Management Protocol (SNMP) architecture [23].

The other parts of the visualization system need to get additional data about the overlay network, including the active SF chains. In [53], a protocol is proposed that examines the liveness status and identity of the service hops of a Service Function Path (SFP) with a mechanism similar to the ICMP traceroute using the Network Service Header (NSH). When a SFC trace request packet is sent to a service function instance (SFI), it decrements the Service Index (SI) in the NSH. If the SI is equal to the Services Index Limit (SIL), it adds identifying data to the end of the header and forwards the packet back to the SFF [53].

As further explained in chapter 2.11, Label Distribution Protocol (LDP) includes a basic and an extended discovery mechanism. The basic discovery mechanism can discover LSR entities directly connected at the data link level. The extended discovery mechanism can also find not directly connected LSRs [69]. This can be useful to get data about existing SFIs because each LSR holds a label information base (LIB) with an entry for each learned label and a corresponding LDP entity. LDP mechanisms for loop detection, on the other hand, can provide information about SFPs because LDP messages contain Type-Length-Values with a list of traversed LSR Ids and a hop count [69].

4.2. Data Model

A data model can be derived from the structure of the available topology data. Based on three important information databases in the SFC architecture, the data model for the SFC visualizer should have three main layers representing this data. The first part is the data about the underlay network topology stored in the TIB. As mentioned before, this data can also be obtained by analysing the network topology with various protocols. Nevertheless, key to understand the process of forwarding traffic between virtualized network components is to know their SF hosts and how network traffic is forwarded between them. The second important part of the SFC infrastructure is the information about the running SFs that are deployed on the hosts and are chained in specific order using SFC. This data can be accessed in the SIB of the SFC architecture. The third essential part representing the SFC infrastructure are SF chains and their corresponding instantiations and SFPs. This data can be sourced from the information about SFC policies or the implemented SFPs in the SFC orchestrator implementation [38].

Information about the hardware topology is necessary for the underlay network visualization, including the hosts and the switches. Network addresses, and links are relevant to the depiction of the SF hosts. In the case of an MPLS-based SFC infrastructure, corresponding MPLS labels are also important.

For the service functions in the overlay network, data about their hosts to locate them, in case of MPLS usage their MPLS labels is fundamental. Information about provided services could be helpful. A status update determining if the virtual machine is running could be valuable data to determine connectivity problems. Knowledge about SFPs, SFs, and their instances, package classification, and an individual ID is necessary for the SFC visualization. Additional information on any network layer can be helpful, as long there is a way to visualize it systematically.

When visualizing many different items in a network, it is essential to group them by categories and visualize them accordingly. It helps the user keep an overview. Classifying items by function also makes it easier to understand how the network elements interact and work within the system. Virtual network functions with different service types should be told apart on the virtual network layer. In order to recognize specific hardware nodes or virtual network function nodes in the network, there is also a need for a specific ID for each node and some additional information to display, which are easier for humans to identify them by.

Links have to hold information about the two points being linked. In a computer network, links are primarily bidirectional because they are often established between two communicating parties. In a visualization, a link can be represented by a unidirectional line between two objects, drawn from a starting point to an endpoint. Information about bidirectional links needs to be represented by two lines for every link.

Static data models with a precise format and defined fields could be leveraged to order and read the data. A strict data model definition might be relevant to the security of the web-based application because it would prevent injections from outside. On the other hand, it could become a problem if a slightly different data structure is introduced. That is because the infrastructure data from different virtual network functions and hardware types can be very heterogeneous and is supposed to be combined in one visualization tool. Therefore, it is better to use less strict data model definitions and add algorithms that catch exceptions and errors. For every layer of the data model, a format needs to be assigned that is suitable for being used in visualization algorithms.

4.3. Network Visualization

Visualizing a computer network on different layers can be compared to the challenge of depicting a dynamic multi-planar network graph. On the one hand, the visualization should be separated into layers. Nevertheless, it should also provide an overview of the network and show the relationship between data on all layers. Because the network structure can change over time, the network visualization needs to be calculated dynamically based on the given network data. Different layers of the network infrastructure and the relationship of network components across network structure layers and visualization system layers should be visualized simultaneously. Therefore, the visualization system needs to combine a multi-layered network graph visualization rendered dependent on the recent network data, a layered separation corresponding to network layers and functionality. On top of that, interactive elements have to be introduced to connect the visualization layers [45].

The SFC infrastructure on the data-link layer roughly corresponds to the data about the hardware devices in the network and established links between them. However, the SFC architecture references a SFC domain that is not relegated to one subnet or one physical network of computers connected to each other. The deployed SFs and SF chains can be addressed using MPLS labels, but the SFs can also work on a higher level. The concept of virtualization also introduces a multi-layered perception of the network with an underlay and an overlay part. The underlay network encompasses the hardware devices and links between them, while the overlay network refers to the virtualized components and logical links between virtualized entities [58].

Based on the data model and the architecture of the visualized system, the visualization of the SFC infrastructure should be divided into three main components. A suitable visualization scheme needs to be implemented for each visualization part.

4.4. Web Application Structure

A web-based application should meet the requirements of a web application stack. Preferable is the use of a suitable backend and frontend framework. The web application should contain a frontend component, where the visualization program is

integrated. The frontend needs to be connected through an API with the backend to request the infrastructure data and forward it to the visualization program. The backend needs a function to request an API that provides the data about the SFC infrastructure or access to a synchronized database. It should deliver the necessary up-to-date infrastructure data to the frontend. The structure of the web application concept is illustrated in picture 4.1.

There are two main options to ensure the visualization of updated infrastructure data. A frontend function could invoke each time the backend gets relevant updated information about the network structure and request the data. Alternatively, the frontend could call a function requesting data from the backend on refreshment. In this case, there should be a mechanism that triggers the refreshment regularly and in case of a user request. The second option has significant advantages over the first approach. An automated infrastructure might have frequent changes in the topology data. This could interrupt the user frequently because values are updated that are insignificant for the user's objective at the time, or just the same data is refreshed.

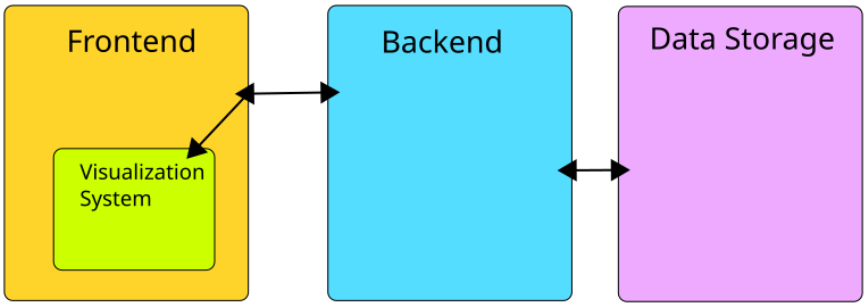


Figure 4.1.: Web Application Concept

A modular web application structure is significant for the reusability of the visualization system or other parts of the web application. Good project documentation is essential if the visualization is supposed to be integrated later into another web application. The use of separate backend and frontend frameworks can improve modularity.

4.4.1. Data Storage

The information about the network topology should be up-to-date and in a specific format to access information about every component of the visualized network components. An API can be requested regularly by a script in the backend framework to archive this goal. A small database could be leveraged to store the requested data. Otherwise, the data could be directly requested from a database synchronized with the data provided to the orchestrator. Directly requesting the data from the orchestrator would increase efficiency but might be a security risk. For testing purposes, a small database with generated test data can be leveraged.

4.4.2. Backend

The application needs a flexible and lightweight backend framework to do database requests and possibly database synchronization. Corresponding data models of the presented infrastructure data need to be defined in the backend. A suitable API should be implemented in the backend framework to connect it to the frontend of the web application. A possible option for this interface could be a RESTful API because it is an optimized implementation for web application architecture. The backend framework needs to be linked to a database to get infrastructure data. That can be done by leveraging another REST API. Some backend frameworks like Django have already built-in database connection management [15].

Suitable backend frameworks should be lightweight because only a few minor tasks have to be managed for the visualization application. Security concerns play a role because the data about the SFC infrastructure could provide attackers with crucial information about the network infrastructure. Therefore, a backend framework with modern encryption standards should be chosen. In this context light-weight frameworks also could make monitoring easier and decrease the number of possible vulnerabilities. The backend framework should also be easy to comprehend and widely used so that potential users can customize the tool without much effort. A modern backend framework also can decrease the development time and possible implementation errors.

4.4.3. Frontend

The frontend of the web application defines the view, elements, and internal routing. A structured frontend framework is needed to display the SFC infrastructure in a structured manner. The view of the application is embedded in the frontend system. In addition to the network visualization system, there needs to be a simple design that matches the network visualization visually and does not distract from the visualization itself. Because the visualization consists of three main parts, multiple links and buttons to switch between visualized network layers could be essential for creating a link between them in the user's perception.

It is difficult to display all the information a user might want on the network graph visualization alone and keep it clear at the same time. People often cannot process much information simultaneously and might lose the overview. Especially when being confronted with the visualization of an extensive computer network. Therefore, ordered and highlighted tables below and to the side of the network graph screen could be an excellent way to balance information and conciseness. To further ensure clarity while increasing the amount of information provided, interactive highlighting of single parts of the network graph visualization could be introduced. The frontend design should also provide features to interact with and scan for infrastructure data of every item separately. A suitable framework should provide a structured implementation of visualization-calculating logic and a corresponding view. It should implement fundamental security standards and provide easy options for deployment and connection to the rest of the web application.

5. Implementation

The implementation chapter aims to explain how the web-based visualizer is constructed and what functionality is provided. As already discussed in chapter 3, the visualizer is divided into three main layers to make it well-structured. This three-layered structure is also based on three important information databases in the SFC architecture. The first part represents the hardware network topology (underlay network), and the second part represents the virtualized network topology (overlay network). The third system component is the visualization of the SF chains. The data models are also implemented based on this structure.

5.1. Abstraction Level

The network topology data necessary for the visualization will be provided by the actual users integrating the application into their system. The methods of connecting the application to a stream of the latest network data are variable and dependent on the system itself. Example data stored for a P4-SFC orchestrator was provided.

5.2. Data Model Implementation

The decision not to use a strict data model with predefined key-value fields is based on the fact that no actual network data was provided during implementation and example data has to suffice. Also, the project's goal is to implement a web-based application for SFC infrastructure. It is easier to adjust the application for different SFC infrastructures with a less strict data format definition. Also, it might become advantageous later for the web application development or reuse process because there will be fewer failures because of malformed data. Because the web application will be mainly used in the internal network, security concerns are sidelined in favour of flexibility. This can be improved in the future when the precise scope of the displayed data is known.

In the search for a suitable data model, a first look can be taken at the internal data storage done by the P4-SFC orchestrator. JSON-formatted files were provided, examples of the data stored in the orchestrator. Because JSON is a lightweight open standard data-interchange format, it is also used as the format for the data model implementation [14].

5.2.1. Underlay Network Data

According to the SFC architecture, a core component of a service function chaining enabled domain is the Topology Information Base (TIB) [38]. It stores for each SF node the number of neighbors, the identification of these neighbors, and additional information about the node [38]. With this data, a picture of the known network topology on the hardware layer can be compiled.

To create a data model suited for the visualization system, the following JSON-formatted data structure is introduced. Available data about network components in the SFC topology is put inside an array referenced as *nodes* and each network component is given an individual ID and a *group* key to classify different node categories in the hardware layer network. How the visualization can be specified for each *group* key value will be discussed later on. This key could also act as a placeholder representing hardware node differences.

```
"nodes":[
    {
    "id": "randomstring1",
    "group": "1",
    "hostname": "machine-1",
    "username": "osboxes",
    "ip_address": "192.168.122.10",
    "hypervisor": "qemu+ssh://osboxes@192.168.122.10/
     system?no_verify=1",
    "lxd_daemon": "https://192.168.122.10:8443",
    "mpls_label": 100,
    "net_iface_out": "ens3",
    "MAC_address": "0c:21:c5:da:70:00"
    },
    ...
    ]
```

Listing 1: Altered example for the web-application

Test data is generated, and hypothetical links between hosts are added. These links can be calculated from the TIB data because all the neighbors of each SF node are known. A link for each neighbor for every SF entry in the TIB structure can be added to archive this. To visualize a link, a target and a source point are necessary. Therefore links are added in the following format using the ID field of the network nodes as a reference to the nodes.

```
"links":[
        {"source":"<node1>", "target":"<node2>", "value":10},
        {"source":"<node2>", "target":"<node1>", "value":10},
        {"source":"<node2>", "target":"<node3>", "value":20},
        {"source":"<node3>", "target":"<node2>", "value":20},
        ...
     ]
```

Listing 2: JSON-formatted example data of links between nodes in the underlay network

The links are now defined in a directed format. Because links for every SF node and all neighbors are calculated, a bidirectional link is defined by combining links for both directions. The key *value* is added for each link, representing differences in links like forwarding latency and capacity. This variable will be used later on to change the visualization of a link accordingly. That can be helpful, for example, to debug a service function chain, which is not optimized for certain link values and therefore fails or specific problems arise. The *links* and the *nodes* arrays are later accessed via the key *Network* inside a JSON-formatted structure with another field called *NetworkID*.

```
{   "Network": {
                    "nodes": [...],
                    "links": [...]      },
    "NetworkID": ...                                  }
```

Listing 3: Underlay data model structure

The key *NetworkID* is used to determine the latest set of network data to display up-to-date changes in the network infrastructure. It is defined as an *AutoField* and will be incremented each time, new network data is added to the database. The network data can then be ordered by the *NetworkID* value. This can potentially enable a look at previous network configurations.

5.2.2. Overlay Network Data

Another core component of the SFC architecture is a Service Information Base (SIB) [38] with entries for each SF in the service function domain. Listing 4 above shows an example of an SFI. The provided example data does not only reference service functions but already instantiated service functions. That is because each component in the array has a specific *vnf_label* and a specific *vnf_host*.

In the context of Network Function Virtualization (NFV), the terms SFs and virtual VNFs are used interchangeably [46]. Because these examples of SFIs were stored at the key *vnfs* in the received SFC example data, the term is used in the application documentation and for the names of variables and data models referencing data about instances when VNFs are actually applied.

The provided SFI data consists of the data about the service function itself like owner, name, and instantiation resource requirements and data specific to the instance like the actual VNF host, the virtual network label, and an individual IP address. Because the data corresponds to specific service function instances, a record called *virtual_network_port* was added to represent data to tell service function instances apart and information on how to access specific instances.

```
{   "id": 27,
    "owner": "c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67",
    "applicationName": "ICMP-Firewall",
    "serviceType": "firewall",
    "bidirectional": "FALSE",
    "ebpf": false,
    "virtualization": "container",
    "vcpus": 1,
    "vmemory": 1,
    "firewallRules": [
    {
    "name": "Deny ICMP",
    "policy": "DENY",
    "direction": "BOTH",
    "protocol": "ICMP",
    "port": 0
    }
    ],
    "vnf_host": "1",
    "vnf_label": 102,
    "ip_address": "10.180.241.196",
    "virtual_network_port": "103"                                    }
```

Listing 4: JSON-formatted service function instance data

The VNF instance data is stored in an array with the key *nodes* and then encapsulated in a JSON structure like the underlay network data. This time, no links are added because VNFs are not directly linked to each other and will be logically chained to each other by implementing SFC policies.

5.2.3. Service Function Chain Data

As already mentioned in chapter 4, the SFC architecture includes a service function chain-path service function path base (SFPB). SFPB entries contain data about SFP instances. Provided example data of an instantiated SF chain is shown below in listing 5. The SF chain instantiation data includes fields named *owner*, trafficType, and *vnfs*. The *owner* of a service function chain is a specific user who implements the SFC policy. The *trafficType* states which traffic is forwarded to the SFC. In this case traffic owned by the *owner* and with the *id: 70*.

```
{   "120": {
    "owner": "c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67",
    "trafficType": {
    "id": 70,
    "owner": "c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67",
    "name": "KundeC",
    "ipAddress": "10.10.10.11/32"
    },
    "vnfs": [
    {
    "id": 26,
    "owner": "c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67",
    "applicationName": "dummy1",
    "serviceType": "packetsniffer",
    "bidirectional": "FALSE",
    "virtualization": "container",
    "vcpus": 1,
    "vmemory": 1,
    "firewallRules": "None"
    },
    ...     ]    }                                                    }
```

Listing 5: SF chain data example

This example is adopted with only a minor change as a SF chain data model because it has all the necessary information to calculate the Service Function Path (SFP). The key referencing the SF chain is deleted and instead a individual id is added to the general SF chain information at the beginning. The rest of the structure is kept this way. However, it would be sufficient only to store the IDs of the SFIs because later on, the individual IDs are used to calculate the SFP visualization. Further data about the instances can be requested from the *VNFs* API.

5.3. Application Architecture

Django is used as a backend framework because it is very widely used and easy and fast to implement. It comes already with an option to integrate a database without much further work. For testing purposes, a small database with test data is leveraged. The data is automatically generated using Python scripts and can be sent with the HTTP POST request via a REST API to the database. SQlite is chosen as a lightweight solution for the test data. Angular is leveraged as a frontend implementation because is is an efficient framework to create single-page web applications. Moreover, it was used for the SFC self-service portal web application [34] the visualization is supposed to become part of.

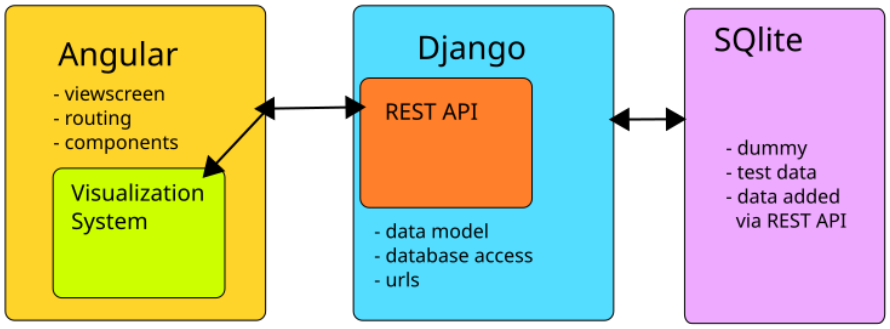


Figure 5.1.: Web Application Implementation

The visualization is implemented in Typescript. It is making use of the D3.js library because the information about the network is in a JSON format. D3.js can be used to visualize data in this format and also offers a wide range of animation features like the D3.js Force, which allows animating a responsive and interactive network graph. It was developed by Mike Bostock to avoid proprietary visualization frameworks and is taking advantage of web standard technologies such as HTML, DOM, CSS, and SVG. D3.js dynamically changes the HTML and CSS code of a website.

For the implementation part of the thesis, Conda is used [3] to set up a virtual environment, a package management, and virtualization tool. It enables running code during development and does efficient package management without interfering with the rest of the system.

5.4. Backend Implementation

The Django framework is installed with packages for a REST framework and Cross-Origin Resource Sharing (CORS) headers. The REST framework package is used to implement a REST API between the backend and frontend. The CORS header package is used to create a CORS whitelist that only allows the backend and frontend

to communicate with each other. Because of security concerns, most modern browsers come with a default setting that blocks cross-origin in web applications. The CORS package can enable secure cross-origin sharing between parts of the web application without the user deactivating this important security feature. It also can allow the resources to be accessed on other domains and has to be configured carefully [17].

The Django framework architecture is based on the Model-View-Controller paradigm. In the model component of the backend frameworks, three data structure models are defined. They are called *VNFs*, *SFCs*, and *Layer2*. Every model consists of an ID field and a JSON field. In the view part, functions are defined that implement a REST API for each model. The URLs for each API are included in the URL specification of the backend application. Also serializers are defined to manage model instances.

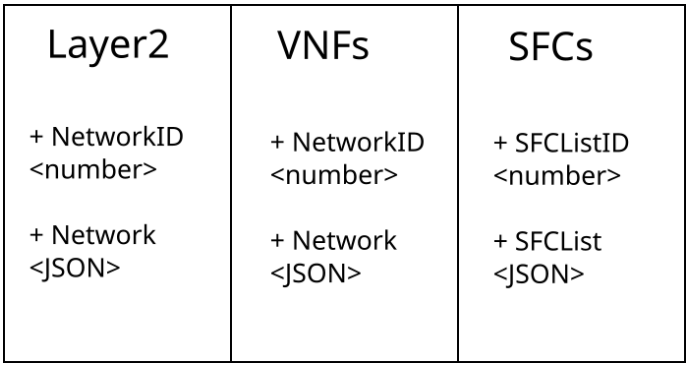


Figure 5.2.: Django Models

The Django models are leveraged to configure the structure of the database so that data sent to the APIs can be stored in the database for every model. To ensure the latest version of network information is sent back by default, the API implementation receives the instruction to send back the data with the highest corresponding ID and IDs are used to track the version of the sent data.

5.5. Frontend

The Angular framework is installed using the npm packet manager for JavaScript, and a new frontend application is created to build a frontend implementation. For all three visualization parts, separate components were introduced to create an ordered visualization view for each part of the visualization. Three preview pictures are presented with hyperlinks to the three different visualization components on the application component view. Services like an API can be defined separately, but for testing purposes, the visualization component implementations directly access the APIs to the network data implemented in the backend framework.

Also an internal *app-routing.module* was leveraged to enable switching between different visualization parts. Additionally, a Navigation Bar was defined in the app component to trigger the routing behaviour. The Navigation Bar consists of three buttons for each visualization part. If a button is pressed, the view of the corresponding frontend application component is displayed, and the button is highlighted with a blue border.

5.6. Visualization Implementation

The visualization implementation is separated into three components that also build on each other. They all consist of the main visualization screen with a network graph and additional tables, tooltips, hyperlinks, and highlighting. The following implemented features are the basis to all visualization layers.

5.6.1. Visualization Screen

There are two standard options to visualize data in an HTML structure. According to the data, a graph can be drawn onto a canvas or in a Scalable Vector Graphics (SVG) format for each data point. Corresponding elements can be defined in scalable vector graphics. The network is visualized inside a SVG element instead of a Canvas element in all three visualization parts because SVG is part of the DOM. Therefore, it is easier to define elements that can be addressed later on. That is useful, for example, to zoom in or highlight a specific part of the network graph visualization. SVG is also more accessible to screen-reading aides for sight-impaired people and is usually easier to debug [13].

5.6.2. Zoom Functionality

A container for SVG elements (`<g>`) named zoomContainer was defined for later reference to zoom into the SVG [47]. It defines a box around the network graph and enables zooming behaviour on mouse scroll events. Later on, all the elements of the network graph are added to the zoomContainer with a suitable D3.js function. The entire network graph visualization can also be dragged in all directions by clicking next to the graph and then moving the mouse accordingly.

5.6.3. Force Simulation

Different forces are simulated between the network nodes in each visualization. The *link force* binds the network nodes to each other and enables the user to pull the whole network graph by dragging individual nodes. The *distance force* implements a threshold defining the space between the nodes. It is leveraged to create a less distorted picture by automatically aligning the network nodes with distance to each other. The center force is used to give the network nodes the attribute that they tend to float to the center if they can, so they stay at the center of the network visualization. A *collision force* was enabled to create a visually appealing simulation. It simulates a collision of the network nodes if they reach certain proximity to each

other. A negative charge force was added to let the nodes float towards each other instead of floating away. This might also trigger the collision force from time to time. Overall, the forces are supposed to draw attention to the visualization, help systematically align the network, and provide the opportunity to manually change the alignment to investigate parts of the network further.

5.6.4. Placing

The network graph visualization is placed right below the Navigation Bar at the top of the view of every visualization part. Tables and a Sidebar are leveraged to provide a better overview of the visualized network. The tables display all the data about the hardware and virtualized parts of the network in an ordered manner below the other parts of the application view. The Sidebar is added on the left side and lists all the items of the network graph the visualization layer is mainly focused on. The hyperlinks lead to a view of the network visualization in a selection mode, where the chosen item is highlighted in the network graph representation. This selected view displays the original network data of the selected item and other corresponding lists below the network graph visualization at each visualization part. The selected view is determined by committing URL parameters, determining which item is going to be selected. A *Refresh* button at the Sidebar can be clicked to get back to the view without a selection. This button is also used to update the network visualization with the newest available network data.

5.6.5. Tooltips

Tooltips are introduced to show more information about selected network nodes when the user moves the mouse over the element in the network graph. The placement of the tooltip is calculated according to the position of the item in the network graph and the borders of the SVG. Tooltips are implemented to bridge the gap between providing the user with all the necessary information quickly and only showing the relevant information to prevent it from being overwhelming.

5.6.6. Colors

The colors blue and yellow are chosen to highlight selected network elements and important values and are used overall in the visualization. These colors are selected to make the web application as accessible as possible by avoiding the colors green and red. Color vision deficiency is a condition impacting one in every 12 men, with the inability to differentiate green and red being the most prevalent.

5.6.7. Underlay Network Visualization

The underlay network visualization aims to visualize the hardware machines and links in a SFC topology. Important to the SFC topology are especially the SF hosts. However, also how these servers are connected on the data-link level. Important to the data-link layer are also other hardware elements like the switches and servers

that are not SF hosts, but can act as a Service Function Forwarder (SFF). The data for this visualization step is stored at the Layer2 API in the backend. It can be accessed by functions in the frontend. For testing purposes, a functionality from D3.js is leveraged to request the Layer2 API and reference the received answer as json-formatted data. A function is called that defines a SVG. Then a function is called to request the data and calculate the visualization.

All elements of the network elements are defined attached to the *zoomContainer*. An HTML anchor is defined for every node in the underlay network data. Then for every unidirectional link reference, a line is put between the source node and the target node. The first positioning of the node anchors is calculated depending on the order of the nodes in the underlay network data. The position of the nodes can be altered by dragging functions. The user can drag a network item by selecting it and pressing the right mouse button while moving the mouse positioning.

The *group* key of every network node is used to decide how to visualize the network element. At the top of the underlay implementation code, a class named *Hosttype* was defined 5.1. It has a public constructor that gives each instance of the Hosttype a *group* field containing a number, a boolean value named *isSwitch*, and a *color* string. A function *addHosttypes()* is provided, where different types of network elements can be defined. Already implemented are two basic switch types and two types of network hosts 5.2.

```
1  //define class of different hosttypes
2  class Hosttype{
3
4    public constructor(
5    readonly group: number,
6    readonly isSwitch: Boolean,
7    readonly color: string
8    ){}
9
10   }
```

Listing 5.1: class Hosttype

```
1  //define new hosttypes
2  private addHosttypes(){
3      let host1 = new Hosttype(1, false, "lightblue");
4      this.hosttypes.push(host1);
5      ...                                                    }
```

Listing 5.2: function addHosttypes

After the *hosttypes* are defined, they are all added to a list. A function called *appendvisible()* first appends a default visualization rectangle to each network element. It then alters this visualization according to the *group* value and the corresponding *hosttype* from the list. Examples of this visualization scheme are pictured in 5.3, 5.4,

and 5.5.

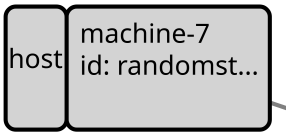


Figure 5.3.: Host

Figure 5.4.: Switch

Figure 5.5.: Default

Because all the nodes were defined as anchor elements, they can be clicked to open the selection mode, as explained before. Every time a node is clicked, the id of the hardware machine is committed as an URL parameter called device. The same thing can be archived by clicking on the corresponding link in the Sidebar. In the selection mode view, the SF node in the network graph visualization is highlighted by changing the fill colors of the rectangles to yellow as shown in figure 5.6.This mode of the hardware layer visualization presents the corresponding JSON-formatted data to the selected hardware device and lists all hosted SFIs by the device. Each list entry is a link to a selection mode view of the SFI in the overlay network visualization.

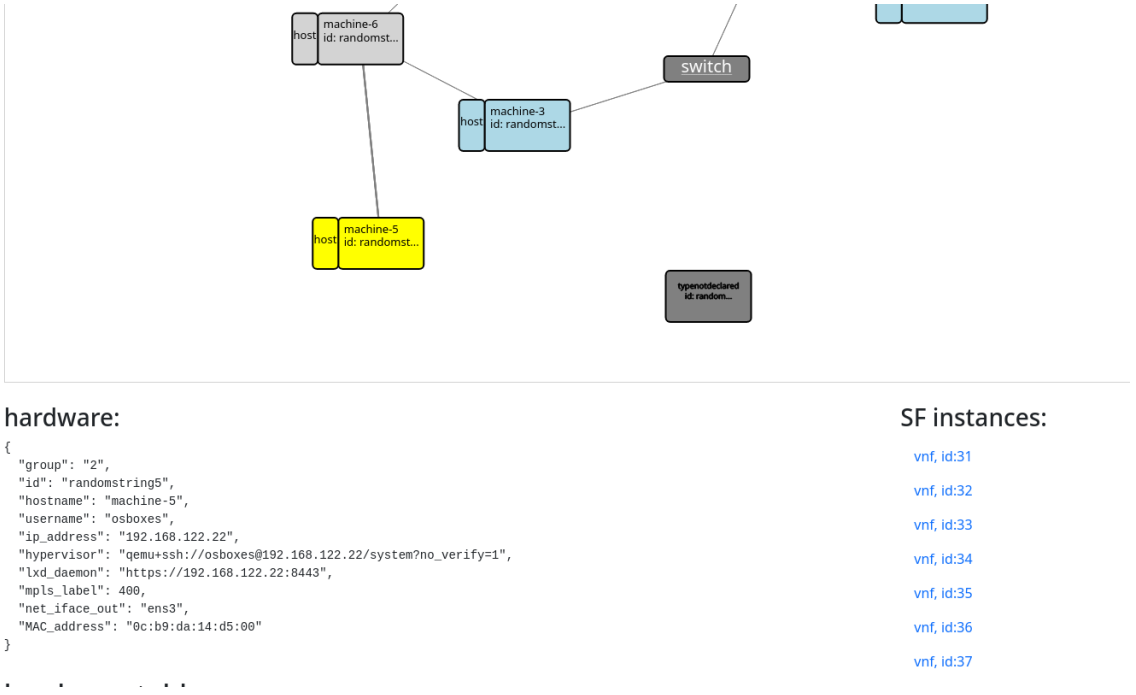


Figure 5.6.: Underlay selection mode

Tooltips appear when the user hovers over an item in the graph visualization. A tooltip provides information about the device like the full MAC address, the full corresponding MPLS label and the full id. While the tooltip appears, the network

item is highlighted by setting the border colors of the rectangles to yellow as shown in figure 5.7.

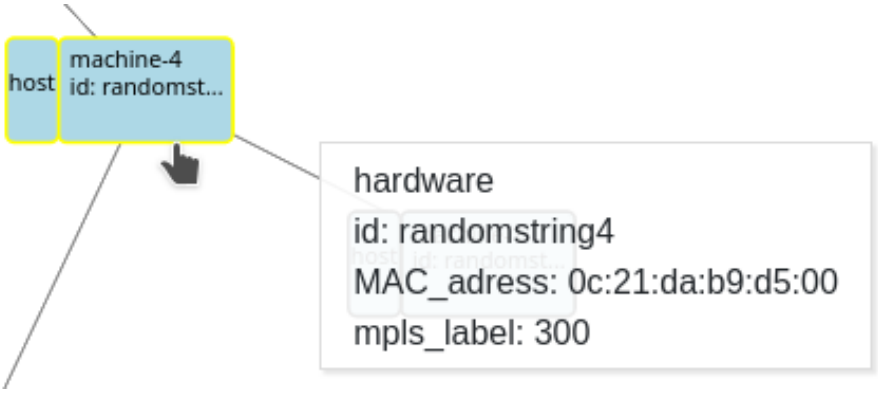


Figure 5.7.: Underlay tooltips

5.6.8. Overlay Network Visualization

The overlay network visualization builds on the underlay visualization implementation. A *VNFtype()* class was defined but not used in the visualization. The class can be used to define how specific SF *serviceTypes* are visualized and is implemented similar to the *Hosttypes()* class in the underlay visualization. Instead of the group, the *serviceType* value is leveraged to decide how to visualize a SFI. The *Hostttypes()* class is also used in the overlay visualization to show different SF hosts.

The overlay visualization shows the SF network, emphasizing the SFs running on the hosts. This is archived by depicting the SFIs placed on the corresponding hosts. To reduce the amount of information displayed at the same time, only the SF hosts are displayed and their connections to each other. Other network components from the underlay visualization functions are filtered out of the original data structure.

Because SF hosts can be linked to each other through other network components, a function called *addVirtualLinks()* is leveraged. The function calculates the links from the *Layer2* API data connections between SF hosts through network components that are filtered away. Links are added to the data structure, marked with the *value*-key entry 13000. The entry is used as a marker for virtualized links that are therefore presented without a link *value*. This decision is based on the fact that virtualized links are possibly calculated from multiple links with multiple *value* field entries, resulting in a distorted result for the *value*-entry of links. However, different value fields could be added to the links, for example, by changing the original structure of the *Layer2* API data beforehand, if necessary.

The overlay network graph depiction is calculated similarly to the underlay network graph. First, anchors are added to the *zoomContainer()* for every SF host. Then the virtualized links are drawn between the corresponding points. The positioning of these points is based on the order the SF hosts are listed in the underlay network

data. Like before, rectangles are added to the anchors of the hosts depending on the determined visualization for the *hosttype*. The figure 5.8 and 5.9 show how SF hosts are depicted in the overlay network graph visualization.

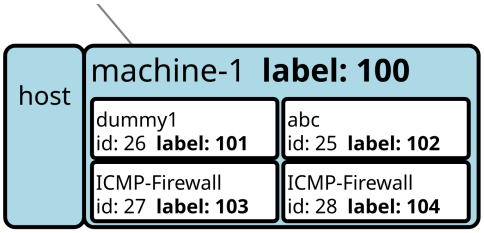


Figure 5.8.: Overlay Host 1

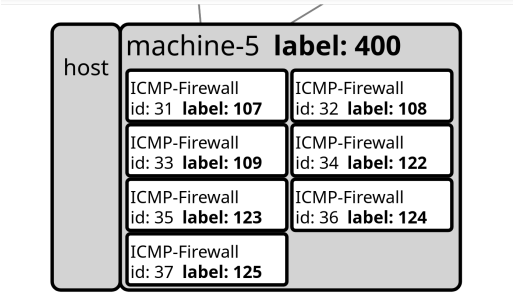


Figure 5.9.: Overlay Host 2

The amount of SFIs running on the hosts is calculated beforehand, and the length of the rectangle holding the SFI rectangles is calculated accordingly. The SFIs are depicted as white rectangles with titles displaying important information like the id, the MPLS label, and the corresponding name of the application or function. The information displayed on the host includes the name of the host and the MPLS label because network packages on the overlay network layer in the P4-SFC orchestrator are routed according to their MPLS labels. The MPLS labels can easily be replaced by information on different NSH implementations for different SFC infrastructure architectures. Because the SFI rectangles have a limited size, the information can only be displayed as a cropped version. Therefore, the full version and additional information like a corresponding network port are relegated to tooltips.

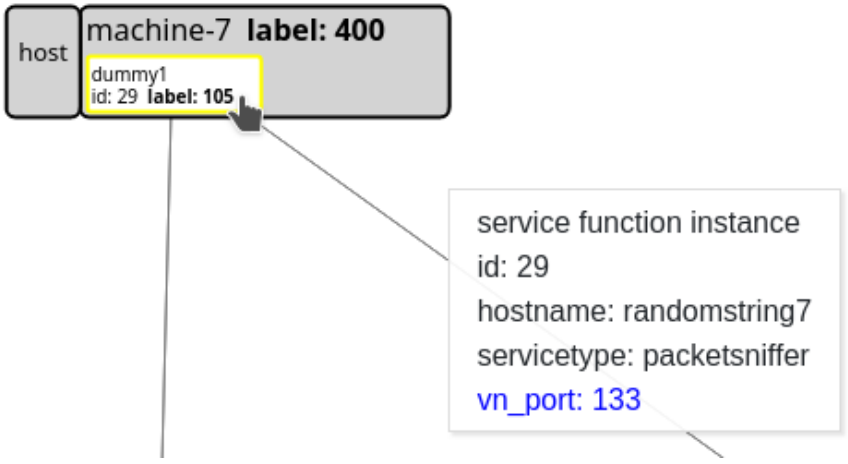


Figure 5.10.: Overlay tooltip

As shown in 5.10, tooltips appear every time a user moves their mouse in the rectangle

of a SFI depiction. The tooltip provides the extended version of the SFI id, the host identification, the *serviceType*, and the network port. The network port entry was highlighted blue because it might become important to tell SFIs from one another. The overlay network graph inherits features like zoom, forces, and dragging ability from the previous visualization of the underlay network. Only the space between the network nodes is increased because network host depiction might become bigger when displaying more hosts. There are at least two possibilities to prevent distortion of the network visualization by two many deployed SFIs on a host. Only one SFI could be depicted representing all SFIs of this type, or the view of SFIs could be limited to only the SFIs owned by one user at the same time.

The Sidebar of the network visualization includes a *Refresh Overlay* button to update the visualization or reset to a view without selected network items. Below the button, hyperlinks for every SFI in the network graph visualization are presented. The hyperlinks lead to a view in the selection mode of the visualization layer highlighting the selected SFI and displaying the original data about the SFI below and providing a link to the SF host under the caption hardware. The selection mode at the overlay level is shown in 5.11.

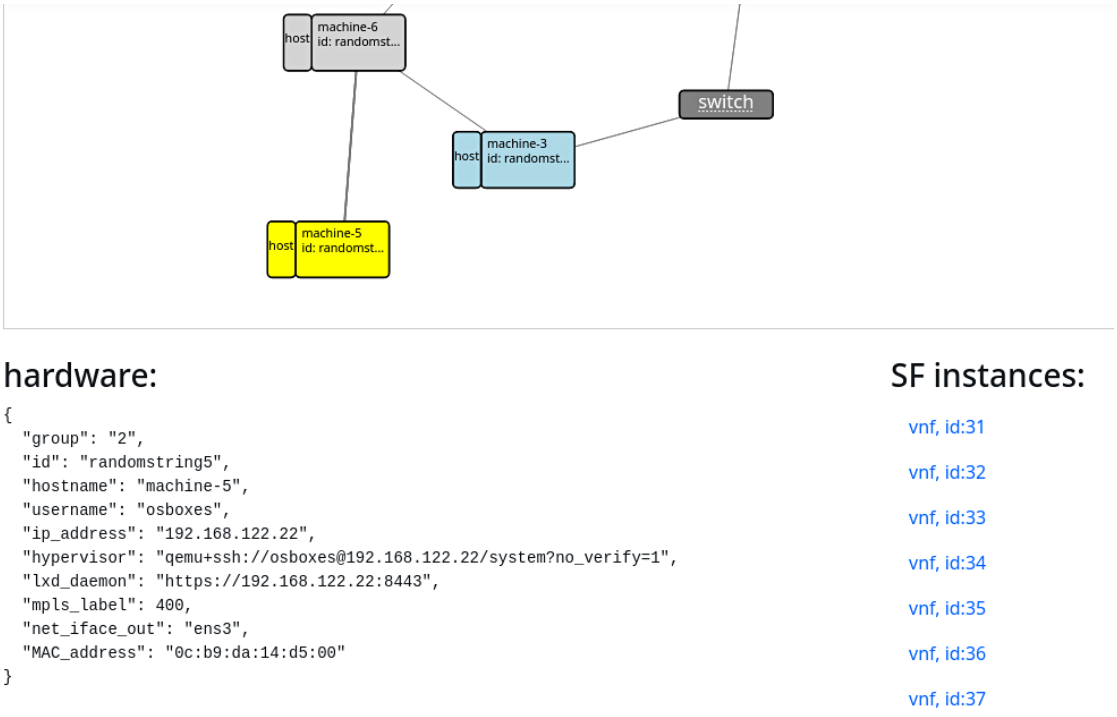


Figure 5.11.: Overlay selection mode

The SF host link leads to the underlay network with the SF hosts id given as an URL parameter and the SF host being selected in the selection mode view of the underlay network. The corresponding SFI is highlighted by filling the SFI rectangle yellow. Below the visualization, tables of the hardware and the virtualized network

components are presented.

5.6.9. Service Function Chaining Visualization

The SFC network graph visualization takes the network graph from the overlay and makes all the links from before invisible. The Sidebar has a *Refresh SFCs* button and lists all the possible SF chains as clickable links. The links lead to the selection mode of the SFC visualization part. When they are clicked, the original data about the SF chain is displayed below the network graph visualization. Also, all the hardware machines and SFIs involved in the SF chain instantiation are listed below. Only the SF hosts with SFIs in the SF chain instantiation are set to maximum visibility in the network visualization. All the other nodes are set to less visibility to focus on the SF hosts of the SFIs in the SF chain instantiation. The function *getPathlist()* is leveraged to determine a SFP for the SFC instantiation. This is done by leveraging known virtualized links and a recursive function to calculate a possible SFP between network elements to visualize the corresponding SF chain. The algorithm for this calculation is greedy. To archive randomness, the list of known virtual links between hosts is shuffled before using a function created for this use called *shuffle()*.

The SF chain is represented by a visualization of a corresponding SFP in the network. The path is visualized by leveraging directed arrows that show the route of the SFP in the SFC network along with the virtualized links. First, the visibility of SF hosts that do not host SFIs involved in the SFP is diminished by lowering their opacity value to create a clear picture. Opacity is a property in CSS that determines how see-through elements in the DOM are. In the next step, the opacity of the virtualized links that are part of the SFP is set to the highest value. Then the colors of the links are set to blue, and the line width is increased. Markers are added for each link. These markers are arrow tips that point from the source to the target node. This time, a directed SFP is visualized. Therefore, only link definitions directed in the direction of the route are transformed. If an edge between two nodes is passed from both directions, both links are made visible, and arrow tips for both directions are shown. This creates a view as shown in picture 5.12.

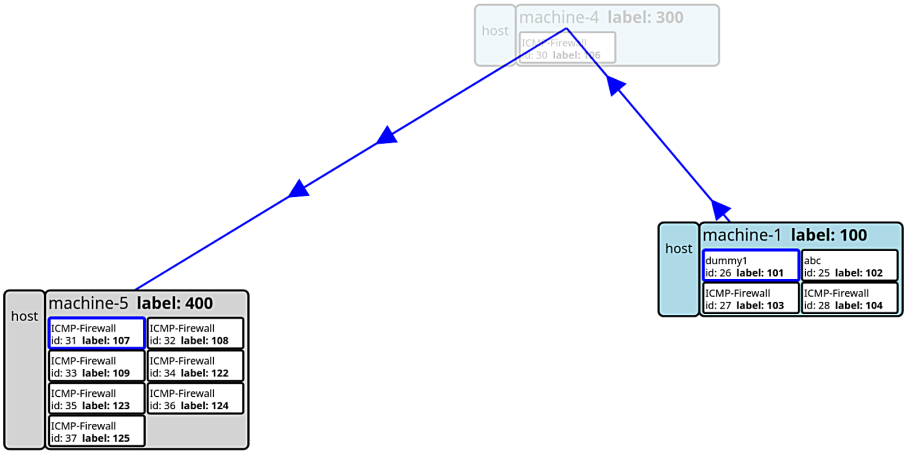


Figure 5.12.: SFP visualization, first example

To emphasize, which SFIs are part of the SFP, the borders of SFI rectangles are enhanced and colored blue. If a SFP is only within one SF host, just the involved SFIs are highlighted as shown in the picture 5.13.

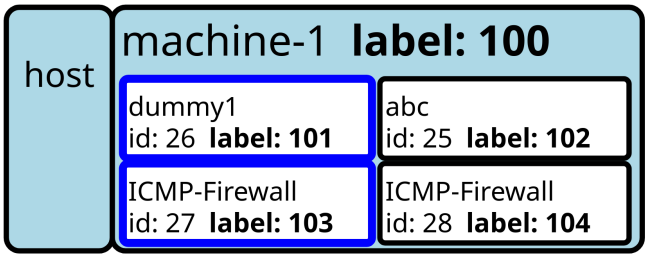


Figure 5.13.: SFP visualization, second example

The tooltip implementation from the overlay network visualization is adopted with minor changes. Instead of highlighting the borders of the SFI representation, the background of the rectangles is painted yellow. This is done because the borders are already highlighted blue to show the route of the SFP. Also, clicking on the rectangles still leads to the component selected in the selection mode of the overlay visualization.

When a SF chain is selected, not only the corresponding SFP is visualized. Comparable with the selection mode of the other visualizations, the SF chain id is utilized as an URL parameter. Below the network graph visualization, the original data of the SF chain instantiation is shown, and listings of the SF hosts and virtualized components that are included in the SFP in the order they are traversed in the SF chain declaration. These list entries are themselves hyperlinks to the network components in selection mode at their corresponding visualization part. The intention behind this list is to show the correct order of the SFIs in the SFP systematically

and to provide the opportunity to investigate each item separately. This might be an important feature to quickly find the cause of a malfunctioning SFC policy. The selection mode of the SFC visualization is shown in figure 5.14.

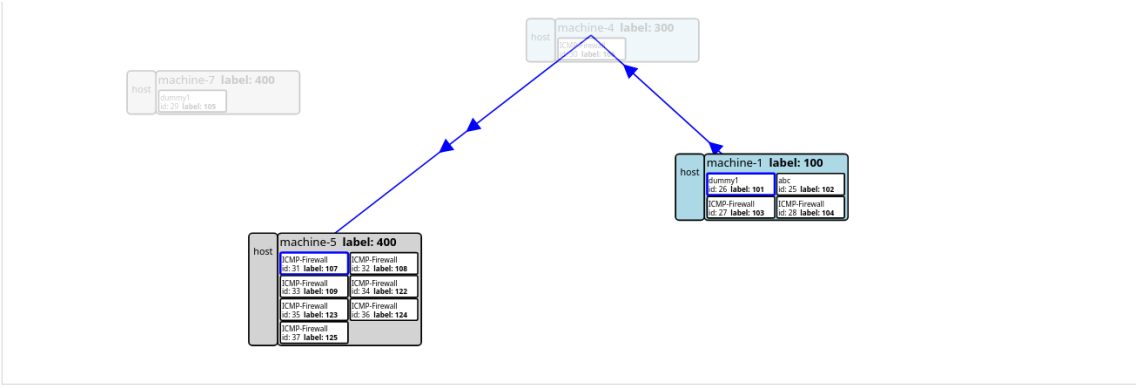


Figure 5.14.: SFC selection mode

6. Evaluation and Discussion

After presenting the implemented features of the SFC infrastructure visualizer, the implications of the implementation approach are discussed in the following chapter. The visualizer is designed to help debug and understand the concept of SFC topology. Important aspects of the implemented design are scalability, modularity and reusability, flexibility, and a clear presentation of the depicted network graph.

6.1. Scalability

The D3.js implementation of the application is designed to be scalable even in the context of visualizing a huge SFC infrastructure with various components. Visualizing a thousand nodes and more is possible because D3.js directly changes the HTML and CSS representation of the website. D3.js is optimized for high-performance data dashboards in the browser.

The scalability of the visualizer was tested by generating network examples with n SF hosts, n SFIs and 2n edges using python scripts. Testing data is sent to the web application in a Post request via curl. At n=100, everything still worked without complications. However, the picture of the network graph started to become crowded. While monitoring a SFC topology with many hosts is possible, limiting the number of monitored SF hosts in the network is recommended. An application called Postman was leveraged to send testing data batches of n=1000, n=500, n=300, and n=200.

Using Postman, it was possible to test sizes up to n=1000. At n=200, the web application still performed at every visualization layer. Only the clarity of the depiction decreased. At n=1000, after each refresh, the network graph simulations are noticeably oscillating in the first three seconds but stabilize after that. Turning down or disabling all the simulation forces except the link force is recommended. Notably, the SFPs are still visualized clearly. Bigger sizes, for example, with n=5000, could not be transmitted through the REST API at once, resulting in an internal server error.

n	posting method	hosts	vnfs/edges	performance	recommended
100	curl	100	200	good	yes
200	Postman	200	400	good	yes
1000	Postman	1000	2000	shaky	no
5000	Postman	5000	10000	unknown	no

Table 6.1.: Scalability Testing Results

Sizes greater than n=200 are not recommended for a clear visualization experience. To oversee that many instances would be an arduous task and often not the best procedure. Therefore, debugging more than 200 servers at a time might not be a plausible use case. The performance of the visualizer is also dependent on the browser's performance. All D3.js simulation forces except the *link force* can be turned off to increase scalability even further. The browser used in the testing is a Firefox browser of version 99.0 running on a P14s Thinkpad with a Manjaro operating system. The scalability of the web-based application itself depends on how the application is hosted.

6.2. Modularity and Reusability

The architecture of the web application has a modular structure that enables reusability. The structure consists of a frontend framework implementation that manages the direct interaction with the user and a backend framework application that manages the access to the database and provides a REST API and a data format to request structured data to calculate the visualization. The frontend framework itself defines different separate components. The code generating the view of each component can also be leveraged separately from the frontend with some minor changes. Even the connected database can be changed in the backend and switched to another data storage option. The documentation of the code and defining multiple distinct functions is intended to ease the integration of parts of the visualization programming into other web applications.

6.3. Flexibility

While a documented implementation with a modular structure provides flexibility in the context of reusability, the web application also provides a highly adaptive operation. The network visualization is calculated from the given network data and changes depending on the data. It is even possible to change aspects of the data structure by introducing new fields with values or new types of hardware and virtualized components without prior declaration in the web application programming. The hardware and virtualized network tables list all the fields from the network data and associated values for every network component. Undefined hardware types are represented by a default visualization, while undefined virtual components are visualized like other service functions and are represented with all known parameters. Unknown field values are just left blank.

The web application has a responsive user interface that alters the layout to fit different screen sizes. The implemented visualizer also enables a flexible user experience by introducing a responsive network graph visualization and lists related links to switch between visualization layers dynamically.

6.4. Clear Network Graph Presentation

A clear network graph presentation is implemented by separating the visualization into different layers and leveraging functions to present abstracted depictions of the network topology. At the overlay visualization layer, only hosts with running SFs are shown. Hops between these hosts and all original links are not depicted. Virtualized links in this network visualization are calculated by determining if two SF hosts are connected to each other without intermediate active SF hosts. These interconnections are unweighted and undirected links and are visualized with lines between the visual SF host representations.

In the SFC visualization part, the abstracted network graph from the overlay visualization is shown with links that completely fade out. Only if a specific SF chain is selected the links between the SF hosts are visible and highlighted, and involved SFIs are emphasized. In this case, SF hosts that are not part of the displayed SFP fade out to some degree of transparency.

6.5. Limitations

The network visualization implementation might become less clear with vast numbers of SF hosts. A solution for this problem would be only to show the network data relevant for each user. This could be determined by looking at the SF chains owned by the user and displaying all related SFIs and SF hosts. Basic security standards in connecting the frameworks in the web application were applied. However, the current web application has no access restriction for unauthorized users. User Authentication and Authorization were side-lined primarily because these aspects are already implemented in the SFC self-service portal. The application can easily be integrated into this web application. Only test data was leveraged in the process of implementing the web application. The web application needs to be synchronized with actual SFC infrastructure data to evaluate aspects that need improvement.

7. Conclusion

Network management is a labor-intensive task and benefits significantly from modern network architectures that improve management, and automation. The SFC architecture enables centralized network processing by classifying network traffic and steering it automatically through ordered chains of SFs. It enables fine-grained orchestration of SFs and is a useful technology for modern 5G-enabled networks [1]. Network visualization is a powerful tool to improve management and the comprehension of network infrastructure by network operators. Especially when leveraging sophisticated network architecture, measures to increase the overall understanding of the system are vital.

Few visual tools for SFC are available because the architecture has only recently reached technical maturity. Most of them provide a visual platform to implement SFC policies and do not visualize SFC infrastructure. A notable open-source observability tool for Cloud Infrastructure is Suzieq. It provides a GUI that can visualize the path between two IP addresses in a network even through an EVPN overlay [19].

In this thesis, a visualizer for SFC infrastructure was designed and implemented. The concept of SFC infrastructure and related architectures was explained. It was also theorized how the data for the SFC infrastructure visualization could be obtained in various SFC domains. The visualization system was implemented to present a clear picture of the network that can be leveraged to debug the visualized SFC infrastructure and also to validate the visualized architecture.

A web-based application with a responsive layout has been designed and built leveraging state-of-the-art web application architectures and frameworks. As part of the application, a scalable and accessible visualizer has been implemented that provides an interactive overview of the network infrastructure. The visualization system was designed to provide cross-layer entity connectivity and layer-level interaction. The visualization system consists of three components. The underlay network component aims to visualize the hardware machines and links in a SFC topology. The overlay network visualization builds on the underlay visualization implementation and shows the SFs and their SF hosts. The SFC visualization part visualizes SF chains by plotting corresponding SF paths. Various methods of highlighting and abstracting parts of the network visualization provide a good overview of the SFC infrastructure topology and a clear picture at the same time.

The web application architecture realizes scalability, reusability, modularity, flexibility and a clear network graph presentation. No less than a thousand servers with multiple running instances of SFs can be visualized simultaneously. More extensive

networks could not be tested because of API framework limitations. Due to this, it is recommended to use smaller networks or not visualize the entire network, as this offers a clearer view of the network topology and provides a better overview.

To conclude, a web-based application was implemented that can visualize various SFC architectures while providing a user-friendly and interactive visualization framework. The visualizer tool can be leveraged to debug, validate and better understand SFC infrastructure.

Bibliography

[1] H. U. Adoga and D. P. Pezaros. Network Function Virtualization and Service Function Chaining Frameworks: A Comprehensive Review of Requirements, Objectives, Implementations, and Open Research Challenges. *Future Internet*, 14(2), 2022.

[2] K. Agarwal, E. Rozner, C. Dixon, and J. Carter. SDN Traceroute: Tracing SDN Forwarding without Changing Network Behavior. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, page 145–150, New York, NY, USA, 2014. Association for Computing Machinery.

[3] Anaconda Inc. Environments. `https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html`, 2017. Accessed: 2021-02-19, Revision c75b4b56.

[4] Ansible, Red Hat. How Ansible works. `https://www.ansible.com/overview/how-ansible-works`, 2020. Accessed: 2021-12-19.

[5] V. Attar and c. Piyush. Network Discovery Protocol LLDP and LLDP-MED. *International Journal of Computer Applications*, 1, Feb 2010.

[6] D. Barrett and G. Kipper. *How Virtualization Happens*, pages 3–24. Dec. 2010.

[7] F. Beck, T. Cholez, O. Festor, and I. Chrisment. Monitoring the Neighbor Discovery Protocol. pages 57 – 57, Mar 2007.

[8] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui. Software-defined networking (SDN): a survey. *Security and Communication Networks*, 9(18):5803–5833, 2016.

[9] M. J. J. Bucher. GENEVIZ — Generation, Validation, and Visualization of SFC Packages, May 2019. bachelor thesis.

[10] bwNET2020+. BWNET2020+. `https://bwnet.belwue.de/`. Accessed: 2022-02-11.

[11] F. Clad, X. Xu, C. Filsfils, D. Bernier, C. Li, B. Decraene, S. Ma, C. Yadlapalli, W. Henderickx, and S. Salsano. Service Programming with Segment Routing. Internet-Draft draft-ietf-spring-sr-service-programming-05, Internet Engineering Task Force, Sept. 2021. Work in Progress.

[12] contributors of the Suzieq Github project. Suzieq – Healthier Networks Through Network Observability. `https://github.com/netenglabs/suzieq#readme`. Accessed: 2022-04-03.

[13] C. Coyier. When to Use SVG vs. When to Use Canvas. `https://css-tricks.com/when-to-use-svg-vs-when-to-use-canvas/`, 2019. Accessed: 2022-04-06.

[14] D. Crockford and C. Morningstar. Standard ECMA-404 The JSON Data Interchange Syntax, Dec 2017.

[15] Django Software Foundation. Documentation: Databases. `https://docs.djangoproject.com/en/4.0/ref/databases/`. Accessed: 2022-03-21.

[16] Django Software Foundation. Why Django? `https://www.djangoproject.com/start/overview/`. Accessed: 2022-03-22.

[17] Django Software Foundation. django-cors-headers 3.11.0. `https://pypi.org/project/django-cors-headers/`, Jan 2022. Accessed: 2021-03-23.

[18] R. Donato. What is Etsi Mano? `https://www.packetcoders.io/what-is-etsi-mano/`, Jan 2021. Accessed: 2022-02-10.

[19] D. G. Dutt and J. Pietsch. Introducing Suzieq. `https://elegantnetwork.github.io/posts/Suzieq/`, Apr 2020. Accessed: 2022-02-19.

[20] D. G. Dutt, J. Pietsch, et al. Suzieq – Healthier Networks Through Network Observability. `https://github.com/netenglabs/suzieq/blob/28f26c1e5d407394537dcd5fd69278e99d4a0f80/README.md`. Accessed: 2022-02-19.

[21] R. A. Eichelberger, T. Ferreto, S. Tandel, and P. A. P. R. Duarte. SFC Path Tracer: A troubleshooting tool for Service Function Chaining. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 568–571, 2017.

[22] A. Farrel, S. Bryant, and J. Drake. An MPLS-Based Forwarding Plane for Service Function Chaining. RFC 8595, June 2019.

[23] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case. Simple Network Management Protocol (SNMP). RFC 1157, May 1990.

[24] T. Fernández-Caramés and P. Fraga-Lamas. A Review on the Use of Blockchain for the Internet of Things. *IEEE Access*, 6:32979–33001, May 2018.

[25] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[26] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir. Segment Routing Architecture. RFC 8402, RFC Editor, July 2018.

[27] J. Galán-Jiménez and A. Gazo-Cervero. Overlay Networks: Overview, Applications and Challenges, 2010.

[28] P. Gomér and J.-E. Johnzon. Computer Network Analysis by Visualization. `https://publications.lib.chalmers.se/records/fulltext/130002.pdf`, 2010. Master's Thesis.

[29] Google LCC. Angular versioning and releases. `https://angular.io/guide/releases`. Accessed: 2022-03-19.

[30] Grafana Labs. Introduction to Grafana. `https://grafana.com/docs/grafana/latest/basics/?pg=oss-graf&plcmt=quick-links`. Accessed: 2022-04-04.

[31] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou. Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, Jan. 2015.

[32] J. M. Halpern and C. Pignataro. Service Function Chaining (SFC) Architecture. RFC 7665, Oct. 2015.

[33] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[34] M. Häberle, B. Steinert, and M. Menth. Firewall-as-a-Service for Campus Networks Based on P4-SFC. In *Electronic Communications of the EASST, Volume 080 (2021)*, 2021.

[35] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (TDGs). pages 315–320, Jan 2007.

[36] M. D. Jacyntho, D. Schwabe, and G. Rossi. A Software Architecture for Structuring Complex Web Applications. *J. Web Eng.*, 1:37–60, Jan 2002.

[37] J.Postel. Internet Control Message Protocol. RFC 777, Apr. 1981.

[38] J. Lan, Y. Hu, G. Cheng, P. Wang, and T. Duan. Service Function Path Establishment. Internet-Draft draft-lan-sfp-establishment-12, Internet Engineering Task Force, Oct. 2021. Work in Progress.

[39] M. Lasserre, F. Balus, T. Morin, D. N. N. Bitar, and Y. Rekhter. Framework for Data Center (DC) Network Virtualization. RFC 7365, Oct. 2014.

[40] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang. A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications*, 156:102563, 2020.

[41] R. R. Leon Shklar. *Web Application Architecture - Principles, protocols and practices*. John Wiley sonst, Ltd, 2003.

[42] Z. Liu and B. Gupta. Study of Secured Full-Stack Web Development. In G. Lee and Y. Jin, editors, *Proceedings of 34th International Conference on Computers and Their Applications*, volume 58 of *EPiC Series in Computing*, pages 317–324. EasyChair, 2019.

[43] G. Malkin. Traceroute Using an IP Option. RFC 1393, RFC Editor, January 1993.

[44] P. Marchetta, A. Montieri, V. Persico, A. Pescapè, I. Cunha, and E. Katz-Bassett. How and how much traceroute confuses our understanding of network paths. pages 1–7, Jun 2016.

[45] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. The State of the Art in Multilayer Network Visualization, 2019.

[46] A. Medhat, T. Q. Thanh, and S. Covaci. Orchestrating Service Function Chaining in Cloud Environments, 2016.

[47] Mozilla Foundation. SVG: Scalable Vector Graphics: MDN. `https://developer.mozilla.org/en-US/docs/Web/SVG/Element/g`. Acessed: 2022-03-29, Portions of this content are ©1998–2022 by individual mozilla.org contributors. Content available under a Creative Commons license.

[48] A. Neumann, N. Laranjeiro, and J. Bernardino. An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing*, 14:957–970, Jul 2021.

[49] OpenJS Foundation. Express. `https://expressjs.com`. Accessed: 2022-03-20.

[50] Open Networking Foundation. P4 open source programming language. `https://p4.org/`. Accessed: 2022-02-02.

[51] J. O. Padallan. *Internet Distributed Systems*. Arcler Press, 2019.

[52] Pallets. Flask Documentation, Foreword. `https://flask.palletsprojects.com/en/2.0.x/foreword/`. Accessed: 2022-03-18.

[53] R. Penno, P. Quinn, C. Pignataro, and D. Zhou. Services Function Chaining Traceroute. Internet-Draft draft-penno-sfc-trace-03, Internet Engineering Task Force, Sept. 2015. Work in Progress.

[54] D. Plummer. An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826, Nov. 1982.

[55] S. Previdi, C. Filsfils, B. Decraene, S. Litkowski, M. Horneffer, and R. Shakir. Source Packet Routing in Networking (SPRING) Problem Statement and Requirements. RFC 7855, May 2016.

[56] Prometheus Authors. What is Prometheus? `https://prometheus.io/docs/introduction/overview/`. Accessed: 2022-04-04.

[57] P. Quinn, U. Elzur, and C. Pignataro. Network Service Header (NSH). RFC 8300, Jan. 2018.

[58] P. Quinn and T. Nadeau. Problem Statement for Service Function Chaining. RFC 7498, Apr. 2015.

[59] Y. Rekhter, S. Hares, and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, Jan. 2006.

[60] F. Rodríguez-Haro, F. Freitag, L. Navarro, E. Hernánchez-sánchez, N. Farías-Mendoza, J. A. Guerrero-Ibáñez, and A. González-Potes. A summary of virtualization techniques. *Procedia Technology*, 3:267–272, 2012. The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science.

[61] C. Rotsos, D. King, A. Farshad, J. Bird, L. Fawcett, N. Georgalas, M. Gunkel, K. Shiomoto, A. Wang, A. Mauthe, N. Race, and D. Hutchison. Network service orchestration standardization: A technology survey. *Computer Standards Interfaces*, 54:203–215, 2017. SI: Standardization SDNNFV.

[62] S. Shahzadi, M. Iqbal, Z. Qayyum, and T. Dagiuklas. Infrastructure as a Service (IaaS): A Comparative Performance Analysis of Open-Source Cloud Platforms. Jun 2017.

[63] J. Shetty, D. Dash, A. K. Joish, and G. C. Review paper on Web Frameworks; Databases and Web Stacks. `https://www.irjet.net/archives/V7/i4/IRJET-V7I41078.pdf`, Apr 2020.

[64] W. A. Simpson, D. T. Narten, E. Nordmark, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, Sept. 2007.

[65] Stack Exchange Inc. 2021 Developer Survey: Web frameworks. `https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks`. Accessed: 2022-03-20.

[66] A. Stockmayer, S. Hinselmann, M. Häberle, and M. Menth. *Service Function Chaining Based on Segment Routing Using P4 and SR-IOV (P4-SFC)*, pages 297–309. Oct. 2020.

[67] P. Subedi, A. Alsadoon, and P. W. C. e. a. Prasad. Network slicing: a next generation 5G perspective. *EURASIP Journal on Wireless Communications and Networking*, 2021(1), Apr 2021.

[68] M. Sultan. Angular and the Trending Frameworks of Mobile and Web-based Platform Technologies: A Comparative Analysis. 2018.

[69] B. Thomas, L. Andersson, and I. Minei. LDP Specification. RFC 5036, Oct. 2007.

[70] D. J. Tian, K. R. B. Butler, J. I. Choi, P. McDaniel, and P. Krishnaswamy. Securing ARP/NDP From the Ground Up. *IEEE Transactions on Information Forensics and Security*, 12(9):2131–2143, 2017.

[71] Y. Xing, J. Huang, and Y. Lai. Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development, Feb 2019.

[72] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang. A comprehensive survey of Network Function Virtualization. *Computer Networks*, 133:212–262, 2018.

[73] T. Zhang. Design of NFV Platforms: A Survey. *CoRR*, abs/2002.11059, 2020.

[74] Y. Zhang, L. Cui, F. P. Tso, and Y. Zhang. Track: Tracerouting in SDN networks with arbitrary network functions. In *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pages 1–6, Sep 2017.

List of Figures

Acronyms

API Application Programming Interface. 5, 14–17, 20, 27, 28, 33–35, 38, 40, 46, 47, 50

ARP Address Resolution Protocol. 12, 24

BGP Border Gateway Protocol. 5

CORS Cross-Origin Resource Sharing. 34, 35

CPSI Control-Plane Southbound Interface. 5

CSP Cloud Service Provider. 3

DOM Document Object Model. 17, 34, 36, 43

ETSI European Telecommunications Standards Institute. 6

ICMP The Internet Control Message Protocol. 11–13, 20, 24

IETF Internet Engineering Task Force. 8, 12

LAN Local Area Network. 12

LDP Label Distribution Protocol. 13, 14, 24

LLDP Link Layer Discovery Protocol. 12, 13, 24

LSR Label Switching Router. 13, 14, 24

MAC Media Access Control. 9, 12, 13, 39

MIB Management Information Base. 13, 24

MPLS Multiprotocol Label Switching. 9–11, 13, 25, 26, 39, 41

MVC Model-View-Controller. 14, 15, 18

NDP Neighbor Discovery Protocol. 12, 13, 24

NF Network Function. 6, 20

NFV Network Function Virtualization. 6, 7, 11, 21

NSH Network Service Header. 9, 10, 24, 41

NVE virtualization edge. 7

OvSDB Open vSwitch Database. 5

SDN Software Defined Networking. 1, 5, 6, 9, 11, 20

SF Service Function. 7–11, 22, 25, 26, 31, 32, 40, 48, 49

SFC Service Function Chaining. 1, 4, 7–11, 19–33, 37, 41, 43, 45, 46, 48–50

SFF Service Function Forwarder. 8, 22, 24, 38

SFI Service Function Instance. 8, 11, 22–24, 31–33, 39–44, 46, 48

SFP Service Function Path. 8–10, 22–25, 33, 43, 44, 46, 48, 57

SIB Service Information Base. 9, 23, 25, 31

SNMP Simple Network Management Protocol. 13, 24

SR Segment Routing. 9, 10

SVG Scalable Vector Graphics. 34, 36–38

TIB Topology Information Base. 9, 23, 25, 30

VLAN Virtual Local Area Network. 11

VNF Virtual Network Function. 1, 4, 6, 7, 9, 11, 21, 32

A. Web Application Startpage

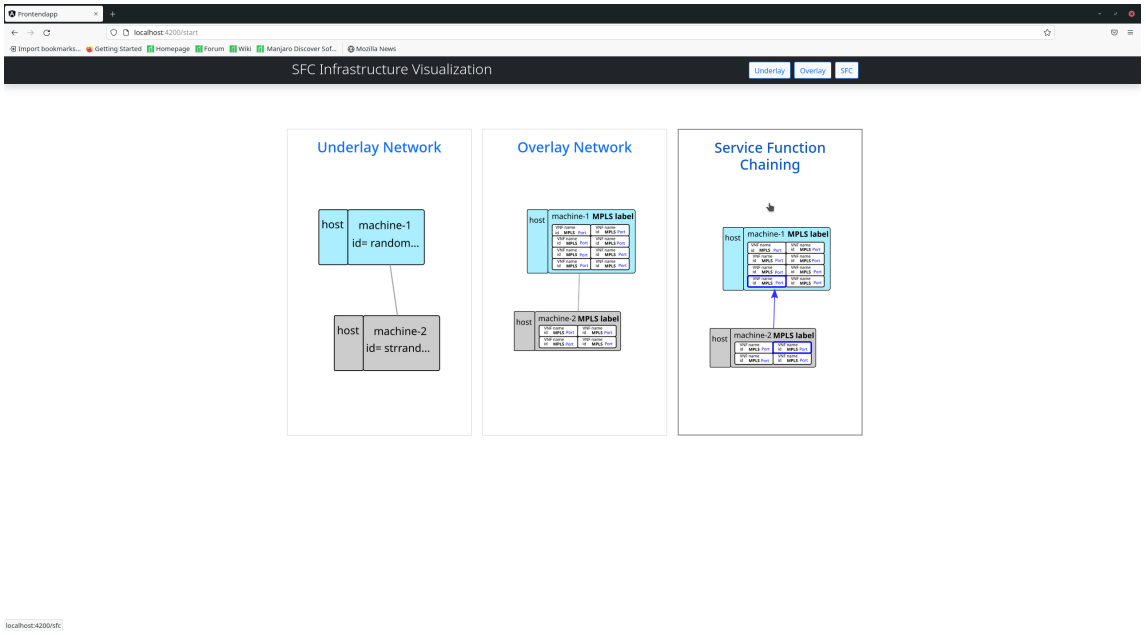


Figure A.1.: Startpage

B. Underlay Visualization

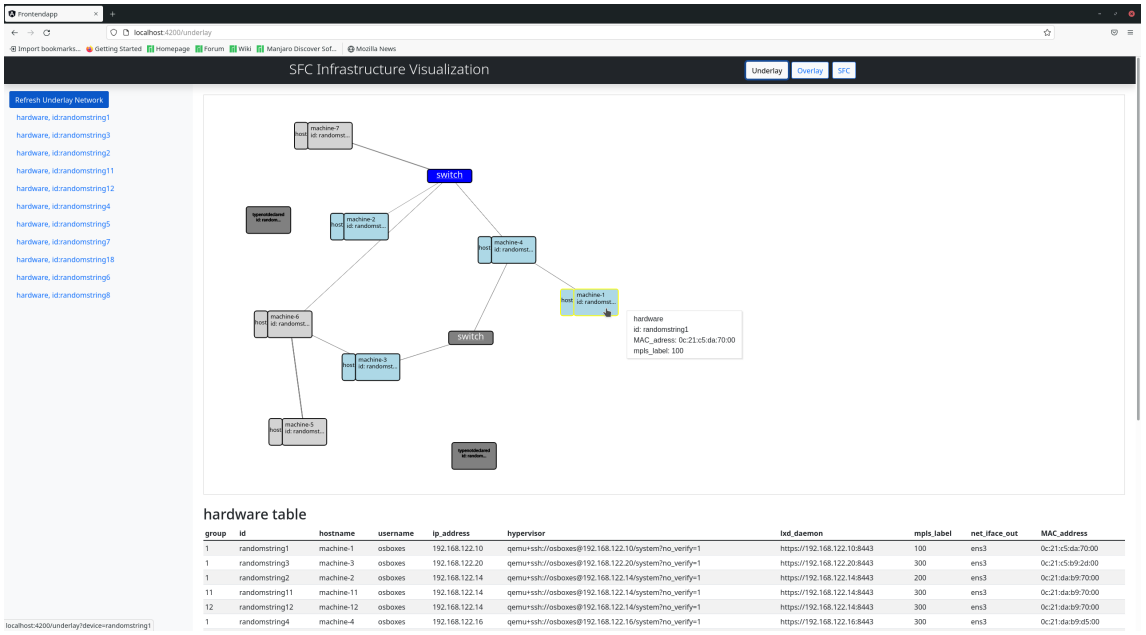


Figure B.1.: Underlay visualization with active tooltip

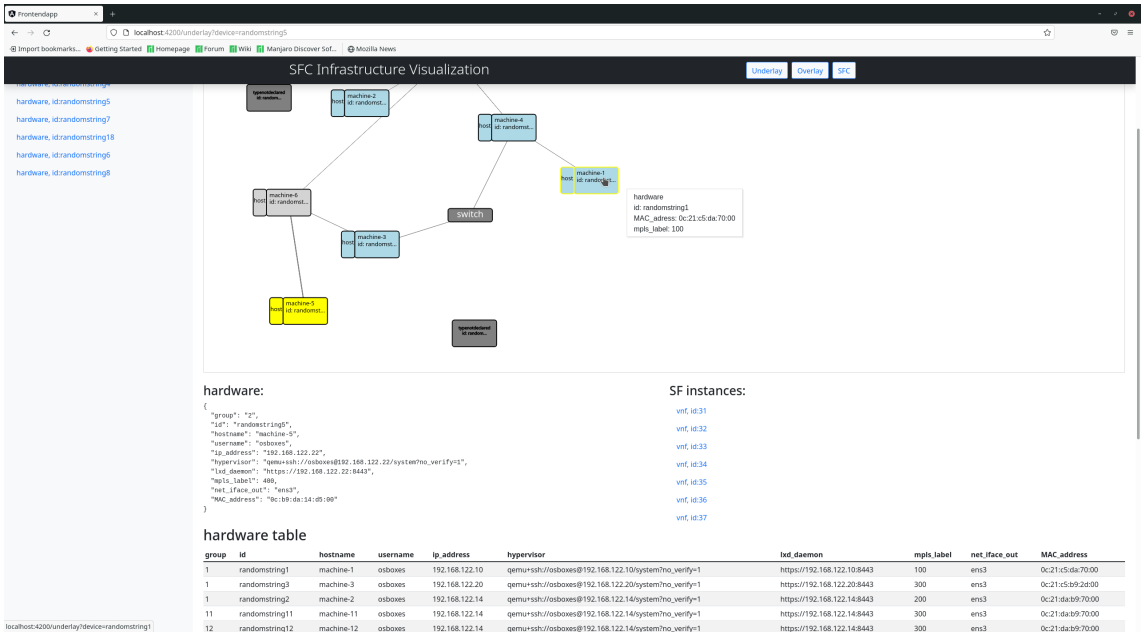


Figure B.2.: Underlay visualization in selection mode

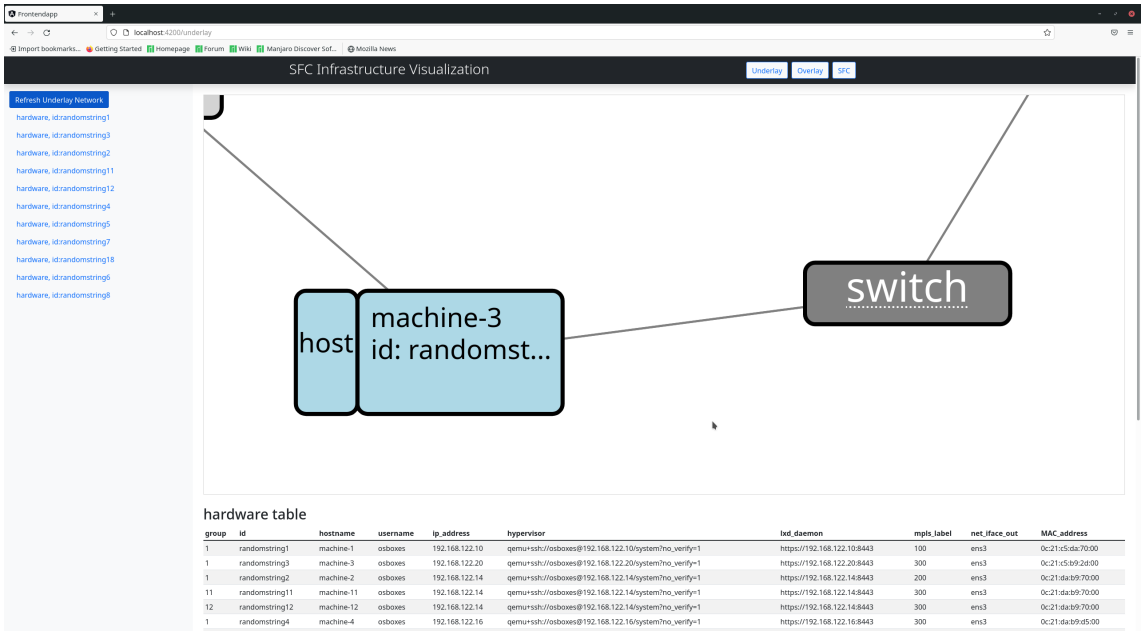


Figure B.3.: Zoomed in underlay visualization

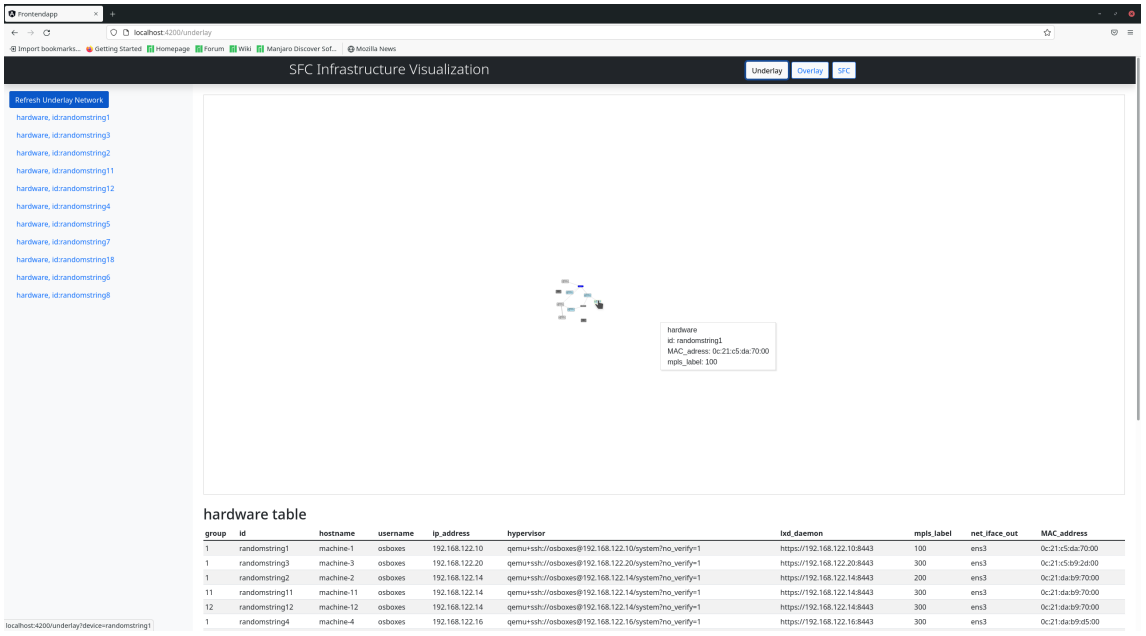
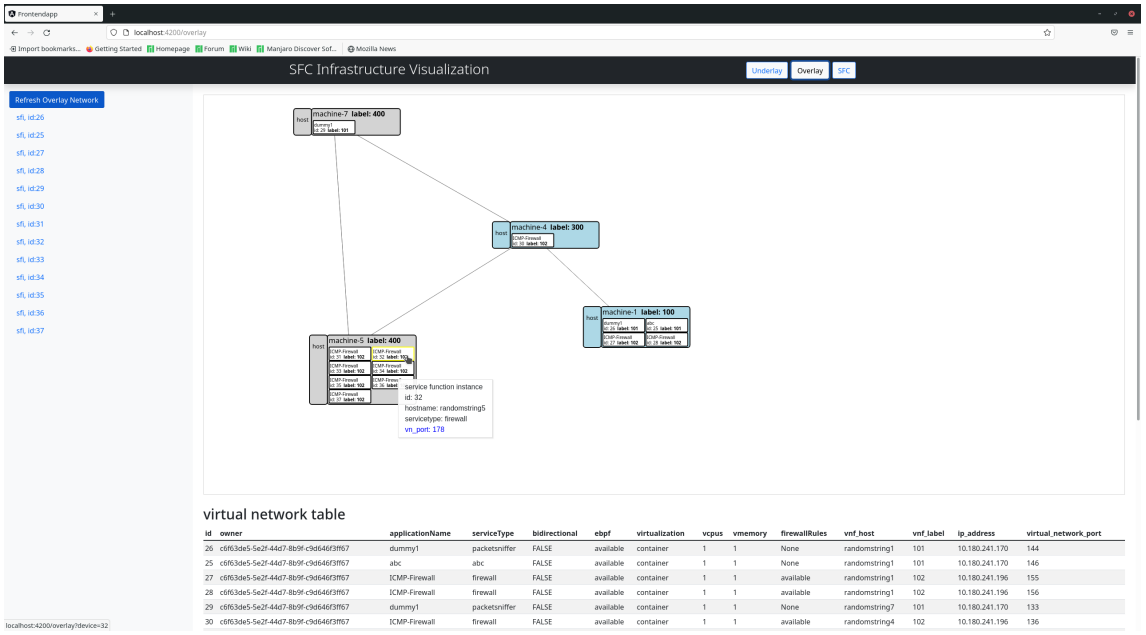


Figure B.4.: Zoomed out underlay visualization

C. Overlay Visualization



id	owner	applicationName	serviceType	bidirectional	ebpf	virtualization	vcpus	vmemory	firewallRules	vnf_host	vnf_label	ip_address	virtual_network_port
26	c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67	dummy1	packetsniffer	FALSE	available	container	1	1	None	randomstring1	101	10.180.241.170	144
25	c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67	abc	abc	FALSE	available	container	1	1	None	randomstring1	101	10.180.241.170	146
27	c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67	ICMP-Firewall	firewall	FALSE	available	container	1	1	available	randomstring1	102	10.180.241.196	155
28	c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67	ICMP-Firewall	firewall	FALSE	available	container	1	1	available	randomstring1	102	10.180.241.196	156
29	c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67	dummy1	packetsniffer	FALSE	available	container	1	1	None	randomstring7	101	10.180.241.170	133
30	c6f63de5-5e2f-44d7-8b9f-c9d646f3ff67	ICMP-Firewall	firewall	FALSE	available	container	1	1	available	randomstring4	102	10.180.241.196	136

Figure C.1.: Overlay visualization with active tooltip

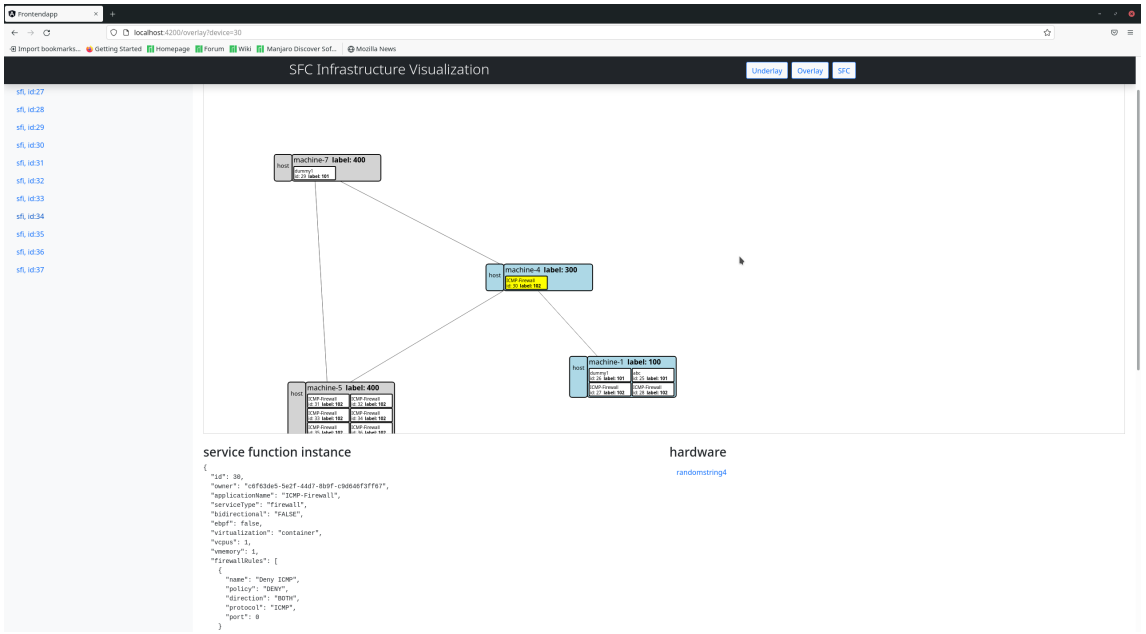


Figure C.2.: Overlay visualization in selection mode

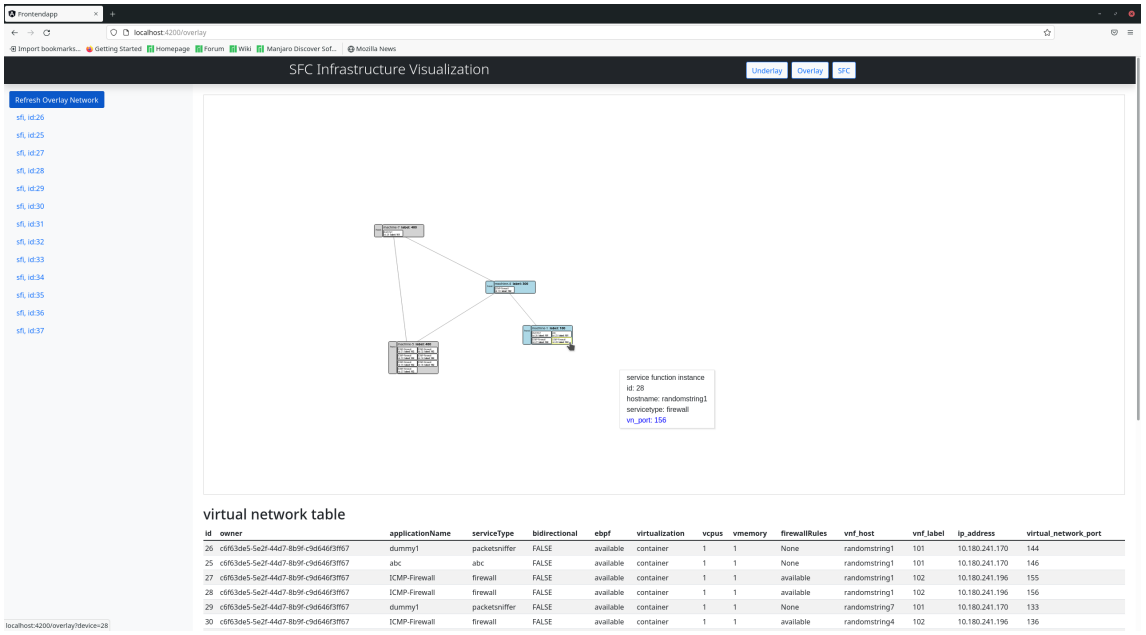


Figure C.3.: Zoomed out overlay visualization

D. SFC Visualization

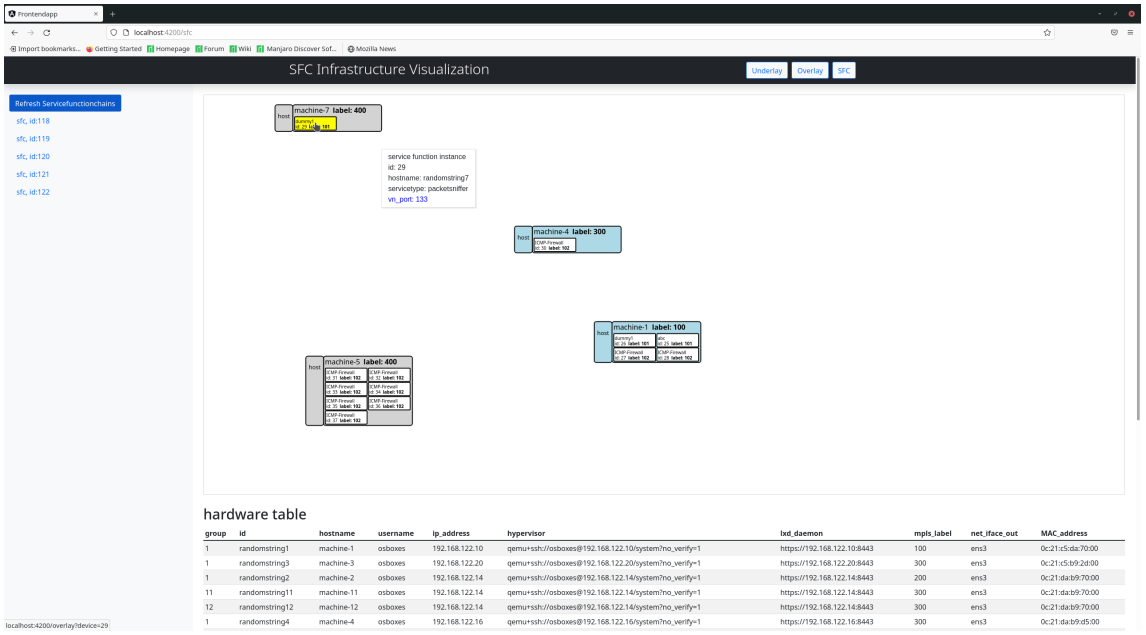


Figure D.1.: SFC visualization with active tooltip

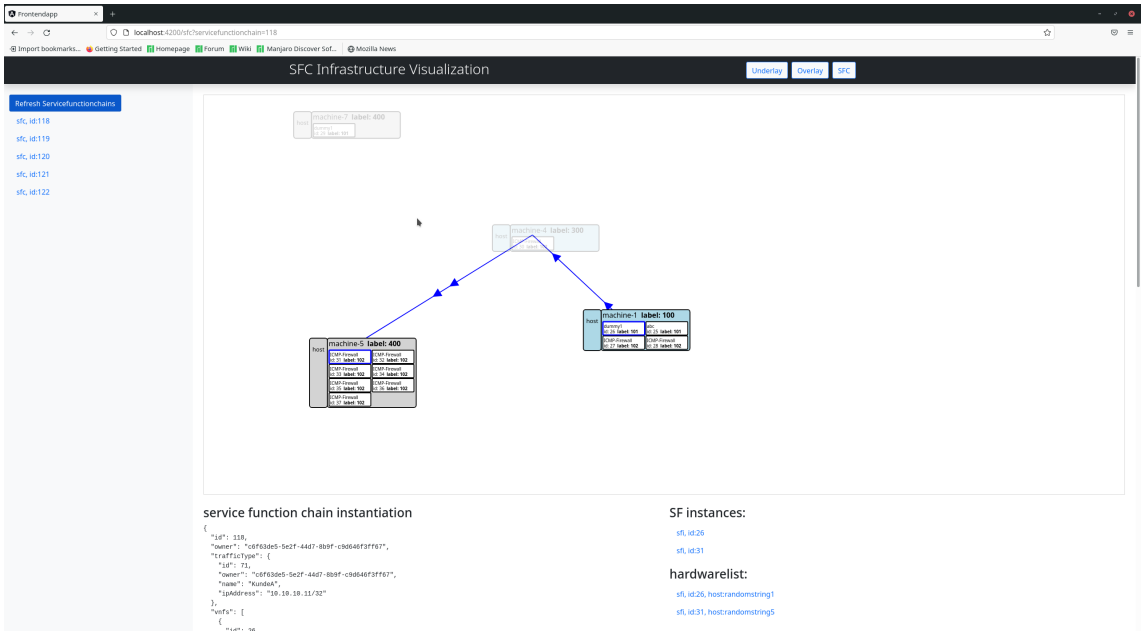


Figure D.2.: SFC visualization in selection mode

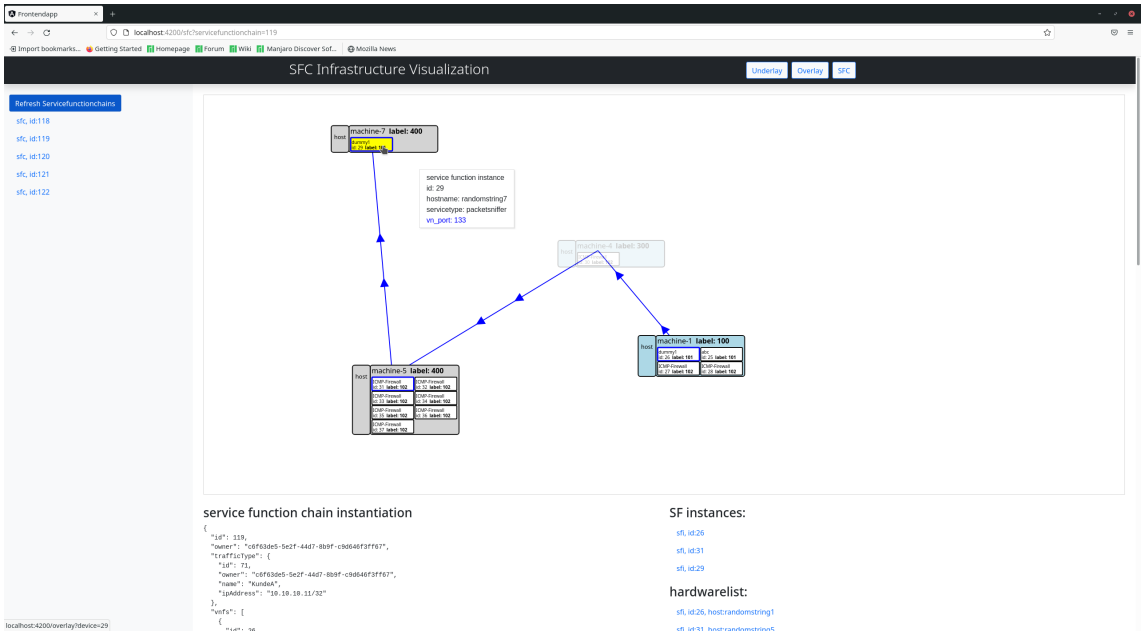


Figure D.3.: Overlay visualization in selection mode with active tooltip

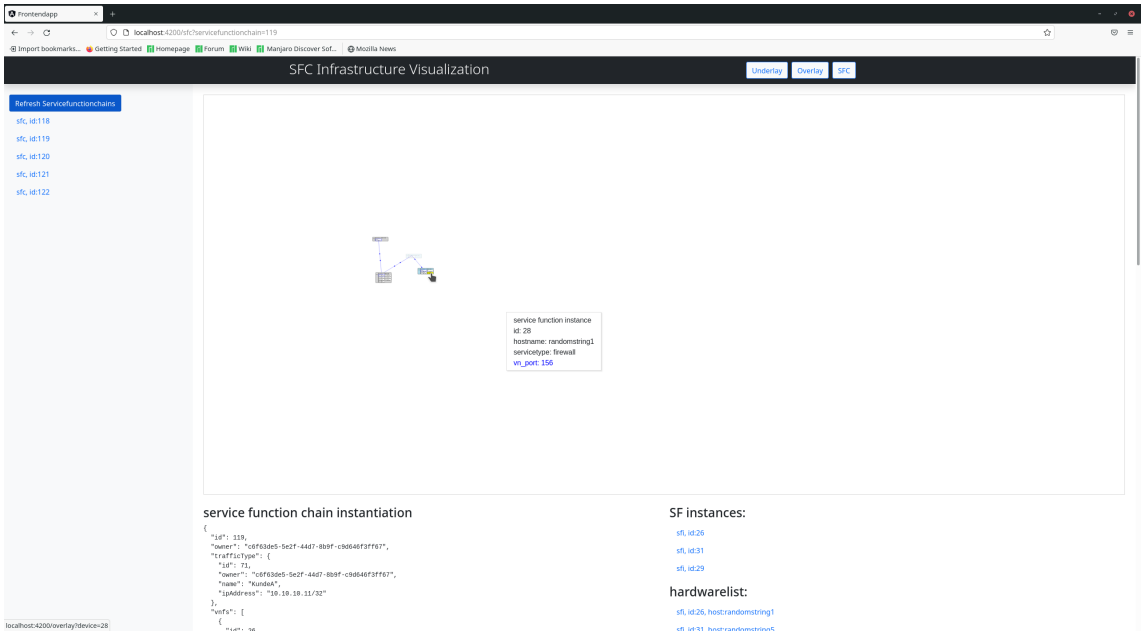


Figure D.4.: Zoomed out SFC visualization

E. Scalability Testing

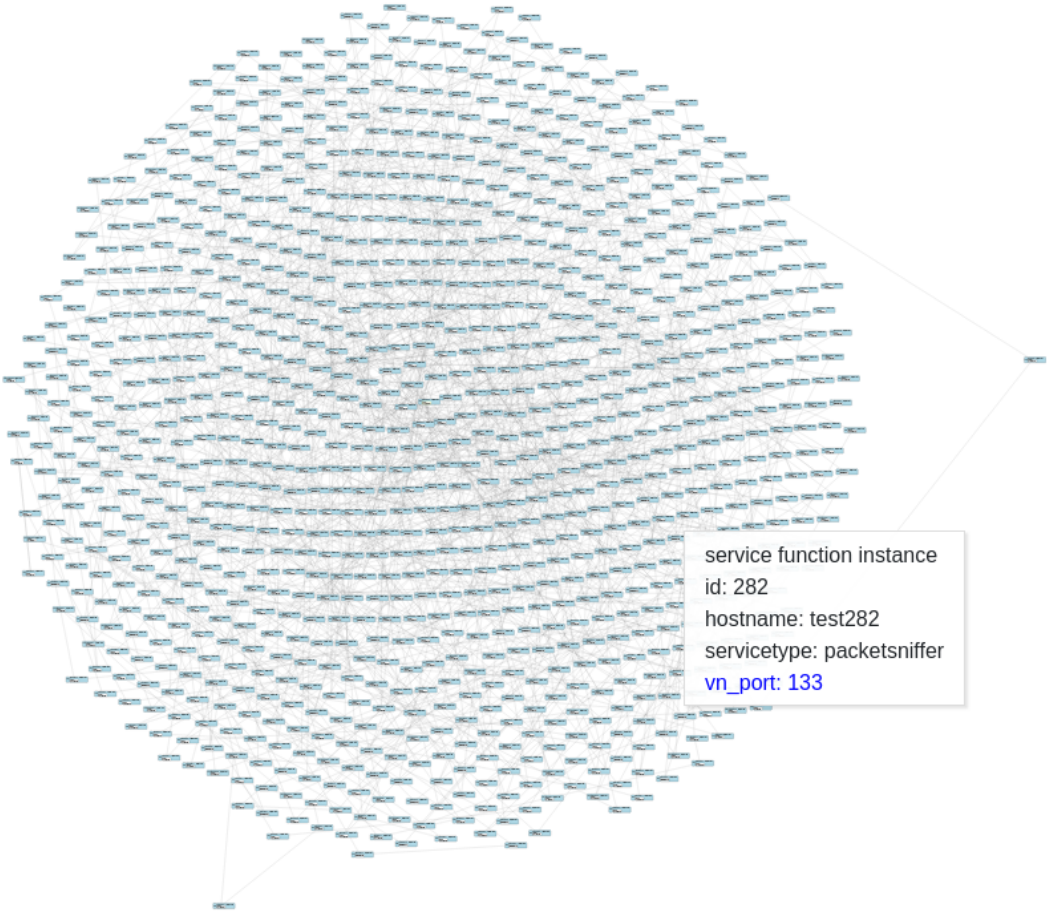


Figure E.1.: Underlay visualization test with n=1000 hosts

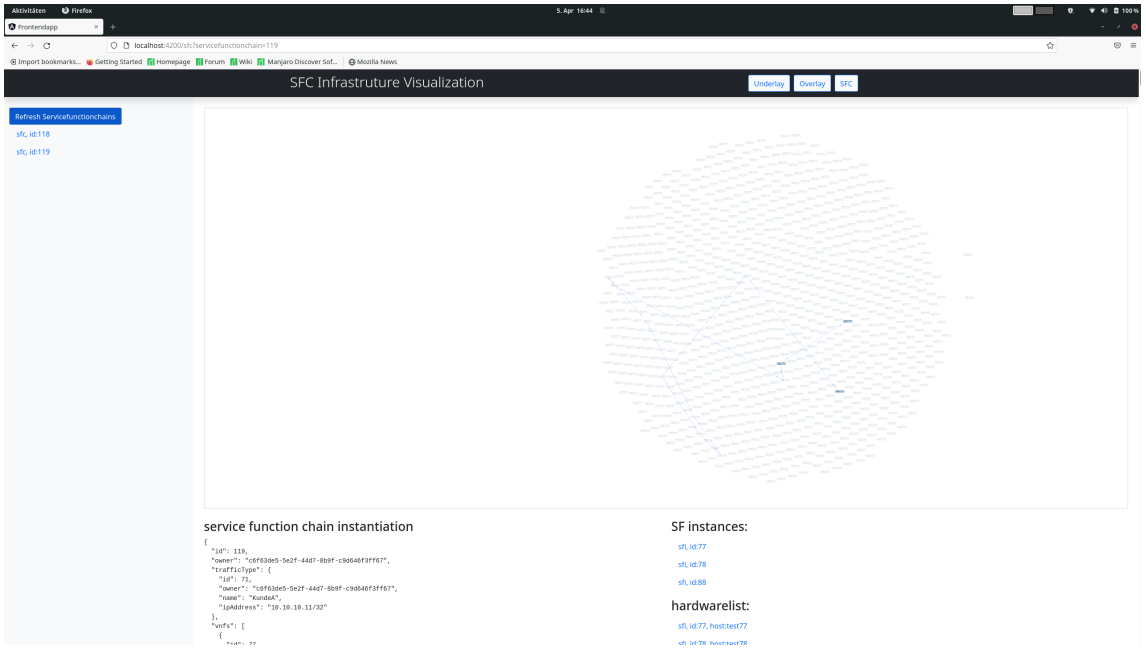


Figure E.2.: SFC visualization test with n=1000 hosts