University of Passau
Faculty of Computer Science and Mathematics

Chair of IT Security
Prof. Dr. Joachim Posegga

Bachelor's Thesis
in
Computer Science

Exploring Visualization Techniques for Wi-Fi Networks in Smart Homes

Jan Pfeifer
105363

Date: September 20, 2023
Supervisors: Prof. Dr. Joachim Posegga
Advisor: Korbinian Spielvogel MSc.Info.-Sec.

Pfeifer, Jan
Weinleitenweg 36
94036, Passau

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel „Exploring Visualization Techniques for Wi-Fi Networks in Smart Homes“ selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind.

Mit der aktuell geltenden Fassung der Satzung der Universität Passau zur Sicherung guter wissenschaftlicher Praxis und für den Umgang mit wissenschaftlichem Fehlverhalten vom 31. Juli 2008 (vABlUP Seite 283) bin ich vertraut.

Ich erkläre mich einverstanden mit einer Überprüfung der Arbeit unter Zuhilfenahme von Dienstleistungen Dritter (z.B. Anti-Plagiatssoftware) zur Gewährleistung der einwandfreien Kennzeichnung übernommener Ausführungen ohne Verletzung geistigen Eigentums an einem von anderen geschaffenen urheberrechtlich geschützten Werk oder von anderen stammenden wesentlichen wissenschaftlichen Erkenntnissen, Hypothesen, Lehren oder Forschungsansätzen.

.....
(Name, Vorname)

Translation of German text (notice: Only the German text is legally binding)

I hereby confirm that I have composed the present scientific work entitled “Exploring Visualization Techniques for Wi-Fi Networks in Smart Homes” independently without anybody else’s assistance and utilising no sources or resources other than those specified. I certify that any content adopted literally or in substance has been properly identified and attributed.

I have familiarised myself with the University of Passau’s most recent Guidelines for Good Scientific Practice and Scientific Misconduct Ramifications of 31 July 2008 (vABlUP Seite 283).

I declare my consent to the use of third-party services (e.g. anti-plagiarism software) for the examination of my work to verify the absence of impermissible representation of adopted content without adequate attribution, which would violate the intellectual property rights of others by claiming ownership of somebody else’s work, scientific findings, hypotheses, teachings or research approaches.

Supervisor contacts:

Prof. Dr. Joachim Posegga
Chair of IT Security
Universität Passau
Email: jp@sec.uni-passau.de
Web: https://www.sec.uni-passau.de

Advisor contacts:

Korbinian Spielvogel MSc.Info.-Sec.
Chair of Security in Information Systems
Universität Passau
Email: korbinian.spielvogel@uni-passau.de
Web: https://korbinian-spielvogel.de/

Abstract

The increasing adoption of smart home devices, fueled by the Internet of Things (IoT), has transformed home technology. While this offers benefits such as improved convenience, it also reveals complexities, especially in the field of network visualization. Current visualization tools are often either too intricate or expensive for most users, highlighting the need for simpler, user-focused solutions. This research introduces a straightforward Wi-Fi network visualization approach for smart homes. Within the Home Assistant platform, the system showcases a force-directed graph that represents both the hardware-based and software-defined network structures of the smart home. Accompanying this visualization is a table detailing key device attributes, such as name, MAC, and IP address. Updated at user-defined intervals, this visualization provides real-time insights, with an emphasis on user customization and ease of use. Implementing this visualization within a Software-Defined Network (SDN) allows not just for visualizing the network flow but also limited network control, letting users manage their devices through interactive visuals. An evaluation reviewed the visualization tool considering user-friendliness, efficiency, and adaptability. This tool offers the possibility to easily identify critical security indicators, like hidden devices or potential security breaches, enabling users to fortify their smart home defenses. Emphasizing user-friendliness and customizability, it appeals to a broad user base. In essence, this research presents a transformative approach to SDN visualization and control in smart homes.

Contents

1 Introduction

1.1 Motivation

In recent years, with the adoption of smart home devices, the landscape of home technology has witnessed a transformative shift. In 2020, global enterprise adoption of IoT stood at 43%, showing a significant increase from 13% in 2014 [1]. Additionally, predictions suggest that the IoT market is set to grow by 19.4%, reaching $483 billion between 2022 and 2027 [1]. Other sources project a significant surge especially in the number of smart home users by 2028. Compared to the 163.85 million users in 2018, estimates suggest a potential growth to 672.57 million, marking an impressive increase of over 410% [2].
This signals that an increasing number of households are incorporating smart home devices, recognizing their potential to enhance day-to-day life. While these advancements offer numerous conveniences, they also introduce complexities. Despite its promising growth, the surge in IoT devices has unveiled several challenges, most prominently in terms of security. According to Sagar Joshi [3], an average smart home could face over 12,000 attacks every week, underscoring the need for security measures in smart homes. What "average" means in this context is defined in chapter 2.2.1.
The roots of the IoT and smart home revolution can be traced back to the drive for home automation and the desire for more efficient, interconnected appliances and systems. Advancements in wireless communication technologies, cloud computing, and artificial intelligence have paved the way for the current IoT boom. Smart assistants like the Amazon Echo Dot [4], Google Home [5], Apple Home [6], and open-source options like Home Assistant [7] underscore the acceleration of this trend, hinting at a future where centralized IoT device control becomes the norm [2].
With the integration of smart devices into home networks, users are often met with a labyrinth of interconnected devices. This intricate web challenges one's ability to understand, and secure these connections, especially for those unfamiliar with IoT networks. Additionally, the niche and specialized nature of many current network visualization tools make them inaccessible or impractical for the average user.

1.2 Research Question and Objectives

The integration of a multitude of smart devices into home networks, as highlighted in the motivation, has presented both opportunities and challenges. These challenges are magnified when trying to understand the dynamics of device communication and potential security threats. One approach that holds promise in navigating these complexities is the concept

of SDNs. SDNs offer a dynamic, centralized, and adaptive means of managing networks, which is explored in-depth in section 2.3. Employing SDNs in smart home environments can provide a unified framework, necessary for visualizing and controlling device communication flow, leading to enhanced security and informed device management.
But why is this significant? Given the rapid growth of smart home devices, users are faced with a web of interconnected devices. Making this network understandable, manageable, and, most importantly, secure is vital. Yet, there's a gap in user-friendly solutions tailored for smart homes that can offer these capabilities. Addressing this challenge has the potential to impact smart home users worldwide, enhancing their security and overall experience.

From this perspective, the research is motivated by the following pivotal questions:

1. How can Wi-Fi network visualization be made more accessible for smart home users?
2. What adaptations are necessary to integrate Software-Defined Networks (SDN) into smart homes?
3. Given the diverse nature of smart homes, which visualization techniques are best suited for portraying an SDN effectively?

To tackle these questions a force-directed graph, a popular visualization method, will be implemented within the open-source application Home Assistant [7]. This representation is chosen to offer users a clear depiction of their network's communications. The capabilities of the Home Assistant platform will be harnessed to not only visualize but also monitor and manage the network's flow using the principles of SDN [8]. This approach aims to empower users, giving them control and understanding over their interconnected devices.
While this study primarily aims to simplify network visualization for smart home users, it recognizes the vastness of the IoT landscape. The focus will be on widely-used Wi-Fi devices rather than other popular options presented in chapter 3, as Wi-Fi is a comfortable and widely adopted protocol [9].

1.3 Thesis Structure

Chapter 2 delves into the background of the thesis. Here, current network visualization techniques are explained, leading to the selection of a fitting method for this study. This discussion aligns closely with chapter 3, titled "Related Work", which reviews existing software and tools. Chapter 4 presents the hardware and software architecture used within this project, followed by detailed implementation report in chapter 5. The penultimate chapter, chapter 6, evaluates the developed solution, analyzing its strengths, weaknesses, and potential improvements. Finally, Chapter 7 synthesizes the research findings, drawing conclusions and outlining avenues for future exploration.

2 Background

This chapter delves into the background of network visualization in the IoT era, the evolution and challenges of Smart Homes, and the transformative potential of Software-Defined Networks with an emphasis on the OpenFlow protocol.

2.1 Network Visualization

Data visualization plays a critical role in digital information processing. Its primary objective is to transform complex datasets into clear, visually accessible formats. This aids in the more consumable and actionable representation of data. By presenting data visually, stakeholders and analysts can discover patterns, trends, and gain insights more effectively [10]. Such representations are particularly useful when detecting hidden actors within raw data or identifying security threats, where anomalies and suspicious activities can be highlighted promptly [11], [12].

2.1.1 Network Visualization in the IoT Era

The surge in the IoT devices has led to denser networks [1]. Consequently, the need for network visualization has grown immensely, serving as a tool for monitoring performance, load, and spotting anomalies or bottlenecks [13].
While network visualization can trace its roots to the broader field of data visualization, the increasing challenges of networks created the need for specialized techniques. These vary from rudimentary link-node diagrams to more sophisticated three-dimensional visualization approaches [14]. Examples for both 2D and 3D visualization can be found in section 3 of this thesis.
When talking about visualization within networks there are two important distinctions that have to be made:

1. **Network visualization:** Representing the network and its connections. This is most often done by using a network graph of some sorts [15].
2. **Network data visualization:** Using the raw tabular network data, visualizations can be generated that provide insight into the communication, load, and performance of the network. Depending on the context, raw tabular network data might be represented using charts, plots, and other graphic formats [16].

2.1.2 Graphs

In the realm of dynamic visualization, several graph types stand out. The (un-)directed graph primarily depicts the direction of flow between nodes. Hierarchical graphs present data in layered structures, but they can have a rigid layout and get cluttered with numerous source-nodes, especially in larger networks. Weighted graphs emphasize the significance of connections, assigning a weight to each link, which can be a indicator of quality, reliability, speed or any other metric for this route [17]. Figure 2.1 shows several of those options:

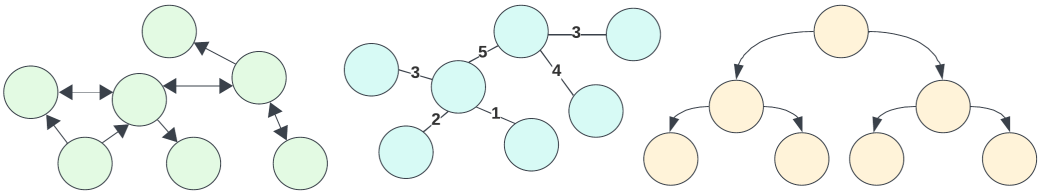


Figure 2.1: Classic Network Visualization Options, in form of a directed, a weighted, and a hirachical graph from left to right [15].

Here, the force-directed graph emerges as a notable option. Not only does it offer a flexible layout that dynamically adjusts to data connections and structures, but it also accommodates the implementation of features like direction and weight. This adaptability and the graph's inherent capability for automatic arrangement make it especially advantageous for network visualization, ensuring visual clarity and easy interpretation [18].

2.1.3 Challenges of visualizing Networks

It is not without a challenge to visualize networks in a suitable manner. Fundamentally, visualization involves showcasing nodes and links, each possibly carrying specific attributes. As networks evolve in complexity with varied attributes and multifaceted node/link features, the task becomes progressively more complex [19]. The hurdles intensify with larger networks where clarity must be maintained despite the multitude of nodes and links. Moreover, the contemporary demand extends beyond static displays, leaning towards interactive visualizations that offer users exploration and customization capabilities [19]. In addition, the question of how updates should be handled within the visualization has to be answered. Does the visualization update itself after a set time-interval, on change, or with a manual reload-button? How these challenges can be tackled is described in chapter 5 of this thesis.

2.2 Smart Homes

Smart homes consists of a bundle of technology and household management, aiming enhancing quality of living through automation and connectivity. Interconnected devices form the backbone of a smart home, providing essential security, comfort, and convenience. The essence of a smart home is to leverage technology in order to create a more efficient and responsive living environment [20].

2.2.1 Definitions

In order to better understand the landscape of smart homes and the boundaries of this thesis, it is crucial to define the terminologies and concepts central to this topic.

Lutolf et al. [20] define a *smart home* as following:

> "The integration of different services within a home by using a common communication system. It assures an economic, secure, and comfortable operation of the home and includes a high degree of intelligent functionality and flexibility."

In essence, a *smart home* manages *smart device*. But now the question arises what a smart device should be. Silverio et al. [21] use the following definition:

> "A smart device is a context-aware electronic device capable of performing autonomous computing and connecting to other devices [...] for data exchange."

Finally, a definition ot the *average smart home user* is needed. In this thesis this refers to an individual who incorporates and utilizes basic connected devices in their household primarily for convenience and efficiency without necessarily having advanced technical knowledge or expertise.

2.2.2 Smart Home Assistants

Since it is now clear what a smart home and smart devices are, the next step is to explore the software that implements the concept of a smart home, so called *smart home assistants*. These advanced platforms function as the primary control point for smart devices. They bridge device communication and introduce interaction methods for homeowners. Prominent voice-controlled assistants include Amazon's Alexa, Google Assistant, and Apple's Siri [22]. They are able to control temperature, adjust lights, play music, and provide information, therefore offering a new way of managing homes [23]. For the purpose of this thesis, open source software is used, namely the popular Home Assistant platform [7].
Controlling smart devices is one of the capabilities of these assistants. In this thesis, the focus is on smart devices connected via Wi-Fi. However, Wi-Fi is not the sole communication technique employed. Other popular protocols, as proposed by Salim Danbatta and Asaf Varol [9], include Bluetooth [24], Zigbee [25], and Z-Wave [26]. Furthermore, using a direct Ethernet connection [27] has its advantages, especially in terms of reliability.

2.2.3 Challenges in Smart Homes

Security

Difficulties in smart homes often arise from security risks. Since these smart devices usually have access to the internet and the local network, user data might be collected and sent to the cloud. In fact, many IoT device vendors like Tuya only support their use within a smart home if a cloud solution is implemented [28]. This means that every command is sent from the device to a server and then back to the home. This not only impacts the performance of the device but also creates a significant security vulnerability in the smart home [29]. A recent example is the Anker Eufy Scandal [30], where security cameras were reported to be uploading private data to the cloud without proper encryption. This highlights that even reputable brands have vulnerabilities which leads to the conclusion that security in smart homes is a critical concern.

Usability

Another challenge for smart homes is the usability in terms of ease of usage and ease of implementation. It is not a trivial task to get a smart home up and running especially for users new to the field. Smart home applications must maintain and improve user quality of life, taking into account user habits, legal and ethical considerations, as well as provider and user satisfaction [31].

2.3 Software-Defined Networks

SDNs represent a transformative approach to traditional network architecture. It primarily focuses on the separation of the control plane from the data plane, leading to a centralized control of the network while the hardware takes care of traffic forwarding [8].

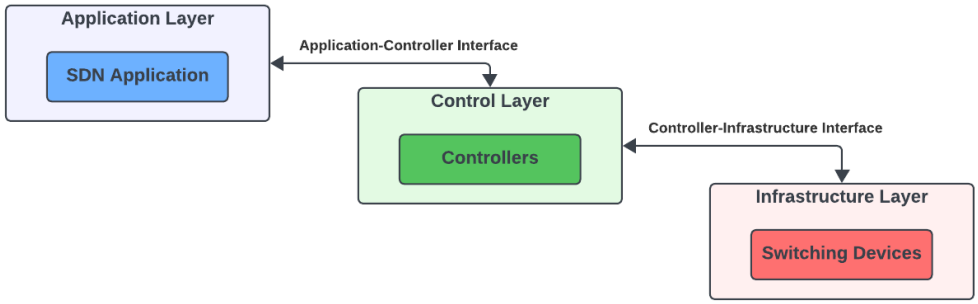


Figure 2.2: The SDN Reference Model with three layers: infrastructure at the base, control in the middle, and application at the top recreated from [32].

2.3.1 Evolution of SDN

At the heart of the SDN architecture lies the network controller, which comes in various forms, from open-source options to proprietary ones. Multiple examples are listed in chapter 3.2.1. Depending on organizational requirements, choices vary from customizable solutions to ready-made offerings with dedicated support [33].
The concept of programmable networks isn't new and predates well-known protocols like OpenFlow [34]. Early projects like Open Signaling in 1995 and Active Networking in the mid-1990s laid the groundwork. Subsequent endeavours, such as the 4D project in 2004 and Ethane in 2006, built on these foundations, ultimately leading to the SDN paradigm we recognize today. Regarding network device configuration, NETCONF emerged in 2006 as a competitor to SNMP. However, full network programmability was realized primarily through the SDN approach [33].

2.3.2 Challenges and Adoption

SDN's centralized design offers visibility into network operations. This complete view is useful for tasks like monitoring, troubleshooting, optimizing performance, enforcing security policies, and managing traffic flows [8]. Despite its promising capabilities, SDN the adoption faces hurdles. Issues range from standardization concerns, integration challenges with older equipment, to a shortage of technical expertise. The cautious approach to its adoption is evident, with many SDN implementations still confined to research prototypes [32].

2.3.3 OpenFlow

OpenFlow stands out in SDN's development, specifying the principle of the control and data forwarding planes. It brings precision to packet handling, defining how devices should manage both rule-based and non-matching packet traffic. OpenFlow is a network protocol designed to manage and direct traffic between routers and switches from multiple vendors [35]. At its core, it centralizes network management, allowing a controller to dictate traffic

routing across the entire network. The protocol achieves this by separating the control plane which makes decisions about where the traffic is sent from the data plane which actually forwards traffic to the selected destination [35].
The communication between the controller and the networking hardware employs what is termed the southbound API. This API is used by the controller to relay instructions to the networking hardware, essentially dictating how data should be forwarded. Conversely, the northbound API is used for the communication between the controller and network applications. It allows network applications to communicate their requirements and desired behaviors to the controller [36].
By standardizing these communications, OpenFlow enables greater flexibility, allowing for innovative network configurations, experiments, and security models without disrupting regular operations [35].
Switches and routers that operate using OpenFlow function by matching received packets with flow rules, which typically comprises a set of match fields (criteria) and a set of actions (instructions) to be executed on matching packets. The following overview presents the most important components [34]:

1. **Match Fields**: Criteria used to match incoming packets.
   - `in_port`: Ingress port of the received packet.
   - `dl_src` and `dl_dst`: Source and destination MAC addresses.
   - `nw_src` and `nw_dst`: Source and destination IP addresses.
   - `tp_src` and `tp_dst`: Source and destination transport layer ports (for TCP/UDP).
   - `nw_proto`: Network protocol, e.g., TCP, UDP, ICMP.
   - `dl_vlan`: VLAN ID.
2. **Actions**: Instructions for packets that match the criteria.
   - `output`: Forward the packet to a specific port.
   - `drop`: Discard the packet.
   - `mod_dl_src` and `mod_dl_dst`: Modify the source or destination MAC address.
   - `mod_nw_src` and `mod_nw_dst`: Modify the source or destination IP address.
   - `mod_vlan_vid`: Modify the VLAN ID.
3. **Priority**: A numeric value indicating the rule's priority. When multiple rules match, the rule with the highest priority is selected.
4. **Timeouts**: Time after which the flow is removed.

3 Related Work

This chapter provides an overview of the work relevant to the thesis topic, exploring both traditional and commercial visualization methods for networks, alongside with insights into SDNs.

3.1 Visualizations

Traditional methods for visualizing networks predominantly utilize a topology view, illustrating the structure as a graph. Pioneering tools such as Intra Vue [37] and Wi-Wiz [38] often present networks in two-dimensional representations. However, particular contexts, especially those merging networks with physical infrastructures like buildings, might find 3D visualizations more advantageous. The subsequent section introduces built-in visualization options for Home Assistant alongside additional choices available through the Home Assistant Community Store (HACS) [39]. These visualization methods include 2D, 3D, and live formats. Furthermore, commercial visualization tools are highlighted; while they might exceed the scope of typical smart home applications, they undeniably incorporate a range of interesting features.

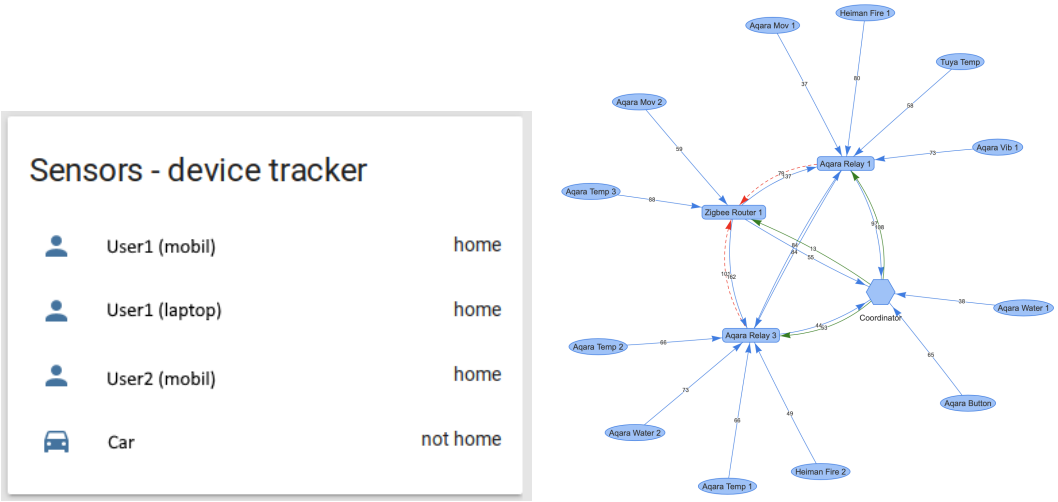
3.1.1 Visualization in Home Assistant

Built-In Visualization

Various software solutions, such as NMAP Tracker [40] and ZHA Map [41], have been developed to depict the IoT devices available in the network or the network itself, with a particular emphasis on smart home environments. Both of these options are natively integrated into Home Assistant, enabling immediate use without much additional configurations. Examples can be found in figure 3.1. However, the primary focus of most default tools remains on device location tracking or devices utilizing the Zigbee Protocol. A significant gap persists in tools specifically designed for visualizing Wi-Fi devices within smart homes.
The challenge amplifies when aiming for a comprehensive solution that integrates multiple communication protocols. The research conducted for this thesis did not find a holistic solution capable of visualizing all connected devices in a network graph.

Floor3D

The closest option to a complete visualization of a smart home is floor3D [44]. The floor3d-card facilitates a 3D representation of floor plans within Home Assistant, as depicted in



(a) Nmap Tracker example by Jeppe Ladefoged recreated from [42]. (b) ZHA Map example by Alexsandro Pires [43].

Figure 3.1: Visualization within Home Assistant.

figure 3.2.

Figure 3.2: floor3d-card: A 3D floor plan card for Home Assistant created by user dedi [45].

This visualization option offers a solution to both visualize and control the devices interconnected within the smart home environment. However, its current capabilities do not encompass the visualization of SDNs or their flow visualization. Additionally, potential users face an initial hurdle: integrating floor3D demands a familiarity with 3D modeling, a task that may be daunting for the average smart home user. Once set up, floor3D presents a detailed and aesthetically pleasing representation of a smart home.

3.1.2 Augmented Reality

Lumos

Within the domain of IoT, a notable application is the detection of concealed communication flows and devices of the Lumos project [11]. By visually representing communication paths, hidden IoT devices can be identified, thereby enhancing both system security and its overall management. This AR technique offers an exciting and new approach to network visualization.

Figure 3.3: Lumos: Identifying and localizing hidden IoT devices in an unfamiliar environment [11].

3.1.3 Commercial Network Visualization

Among the many tools available, certain commercial solutions stand out. This subsection explores these leading commercial network visualization tools. Here are a few exemplary options:

Grafana

Grafana offers highly customizable dashboards, enabling integration with a wide range of databases for versatile metric visualization. Its panel architecture accommodates a diverse array of visualization types within a single dashboard. While an open-source variant is available, Grafana's commercial versions, such as Grafana Enterprise and Grafana Cloud, offer additional features and support. Pricing structures depend on factors such as storage and user activity [46].

Prometheus

Prometheus, originating from SoundCloud, is an open-source system renowned for its monitoring and alerting capabilities. Distinguished by its multi-dimensional data model and potent query language, it seamlessly integrates with visualization tools, notably Grafana.

As an open-source tool, Prometheus does not entail direct costs, but integrations with commercial tools may introduce expenses [47].

Graphlytic

Graphlytic specializes in visual graph analytics. Its intuitive design facilitates the effortless exploration and manipulation of extensive datasets. A noteworthy integration with Neo4j empowers users to unearth patterns with ease. Their pricing model is tiered, with variations based on user count and features. Enterprise editions enhance the depth of offerings [48].

NetCrunch

NetCrunch by AdRem Software presents a comprehensive IT monitoring solution. It is particularly valued for its automatic network mapping capabilities, which transform intricate infrastructures into easily comprehensible visual representations. Licensing hinges on monitored nodes, with pricing contingent on node count and feature sets [49].

While each of the aforementioned solutions offers a unique set of capabilities, there are limitations to consider. Current commercial tools can be costly and with the open-source options currently available, one has to choose between a visualization tool and a flow control system. Moreover, these tools operate separately from the smart home environment. There is a lack of free, open-source tools that offer both network and flow visualization, as well as SDN control, integrated seamlessly into the smart home assistant of choice.

3.2 Software Defined Networking

3.2.1 SDN Controller

The domain of SDNs often utilizes visualization strategies, typically employing the traditional topology view. The importance of data visualization in comprehending network connectivity and security is emphasized in the literature [50].
SDN introduces a change in traditional networking architecture by decoupling network control from forwarding functions, enabling centralized control and manipulation of network behavior. Various solutions, both open-source and commercial, have been developed to address these requirements [51]. Below are some notable open-source and commercial SDN solutions:

Open Source Solutions:

1. **OpenDaylight (ODL)** [52]: Under the Linux Foundation, ODL provides a scalable, java-based SDN controller with a modular design, supporting protocols like OpenFlow.
2. **ONOS** [53]: Led by ON.Lab, ONOS is designed for service providers with a focus on performance and scalability. ONOS is Java-based.
3. **Floodlight** [54]: Spawned by Big Switch Networks, Floodlight is also a Java-based controller compatible with OpenFlow, known for its simplicity.
4. **Ryu** [55]: Ryu supports various protocols, including OpenFlow, and is recognized for its modularity and developer-friendly APIs. Ryu is based on Python.

Commercial Solutions:

1. **Cisco APIC** [56]: Cisco's APIC offers centralized automation with policy-driven application profiles as part of their Application Centric Infrastructure (ACI).
2. **VMware NSX** [57]: VMware's NSX emphasizes network virtualization and security, allowing dynamic network provisioning without the limitations of physical configurations.
3. **Juniper NorthStar** [58]: Juniper's NorthStar provides insights and management for IP/MPLS flows in expansive networks, excelling in traffic engineering.

The choice between open-source and commercial SDN platforms depends on specific organizational needs, scale, and budgets. While open-source options offer customization and innovation, commercial solutions often provide a comprehensive, integrated, and professionally maintained ecosystem.

3.2.2 SDN Visualization

SDN controllers, such as OpenDaylight (ODL), ONOS, Cisco APIC, and Juniper's Contrail, often feature Graphical User Interfaces (GUI) that present a topology view of the network. This visualization displays the arrangement of network elements, which include nodes such as switches, routers, and endpoints, as well as their interconnecting links. Such a depiction aids in understanding the network's structure, its interrelationships, and data flows. Figure 3.4 provides an example of a standard topology view within a GUI.

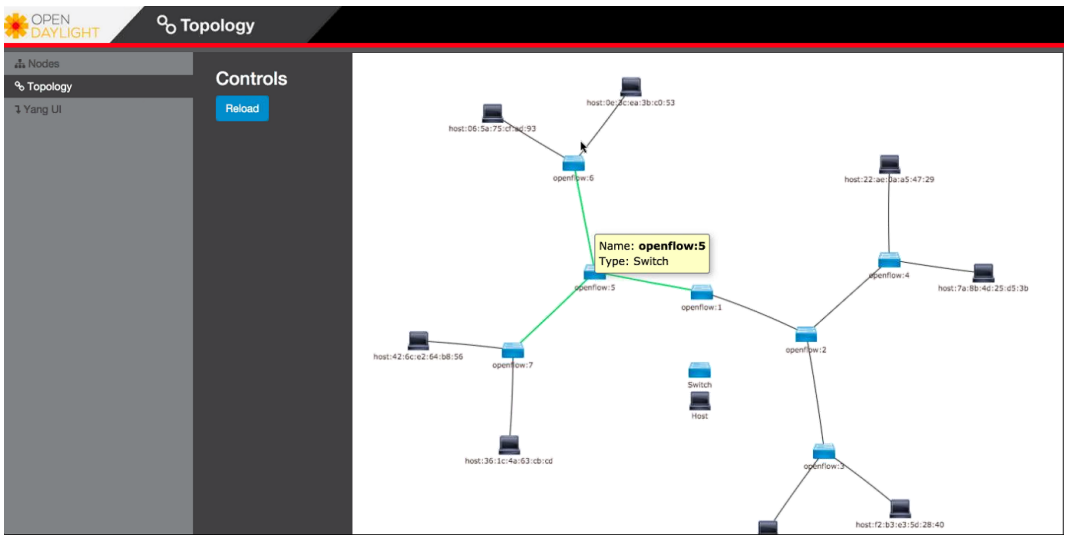


Figure 3.4: OpenDaylight Topology View [59].

4 System Architecture

This chapter depicts the system architecture, discussing both the hardware and software components pivotal to the network visualization project.

4.1 Hardware

The implementation of the network visualization will be carried out in a small-scale setup. The hardware infrastructure consists of the following devices:

- Raspberry Pi 4 Model B [60] as router using OpenWrt and Open vSwitch (OVS)
- Raspberry Pi 4 Model B as Home Assistant
- Raspberry Pi 4 Model B as Ryu SDN Controller
- IoT Devices, like Smart Bulbs, Plugs, etc.

4.1.1 Raspberry Pi 4 Model B and Alternatives

The Raspberry Pi 4 Model B is a compact single-board computer developed by the Raspberry Pi Foundation . Launched in June 2019, this model offers significant upgrades in terms of processing, memory, connectivity, and features. It's powered by a quad-core Cortex-A72 (ARM v8) 64-bit SoC clocked at 1.5GHz, with memory options available up to 8GB LPDDR4-3200 SDRAM - the version used in this thesis. The device supports both 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, Gigabit Ethernet, has two USB 3.0 and two USB 2.0 ports, and uses a microSD card slot for storage [60].
While the Pi's specs are sufficient for this test-bed if split onto two devices, for a full-fledged smart home environment, a more robust solution like the Intel Nuc [61] is recommended. Ideally, a real smart home setup would employ a dedicated OpenWrt router by default, connected to a computer - the Pi or Nuc - managing both the Home Assistant and the SDN controller. Given that every smart home has a router, this ideal setup requires only one additional device - the managing computer. Even a repurposed old PC or laptop can effectively serve as a managing device, reducing the initial hardware costs even more.

4.1.2 IoT Devices

A small array of IoT Devices are incorporated within this test network. Namely, Shelly LED DUO E27, Tapo LED L530E, Tapo Smart Plug Mini Smart Wi-Fi Socket, GW1100 weather

station. These devices are controlled by Home Assistant via the Display.

The physical layout is depicted in figure 4.1:

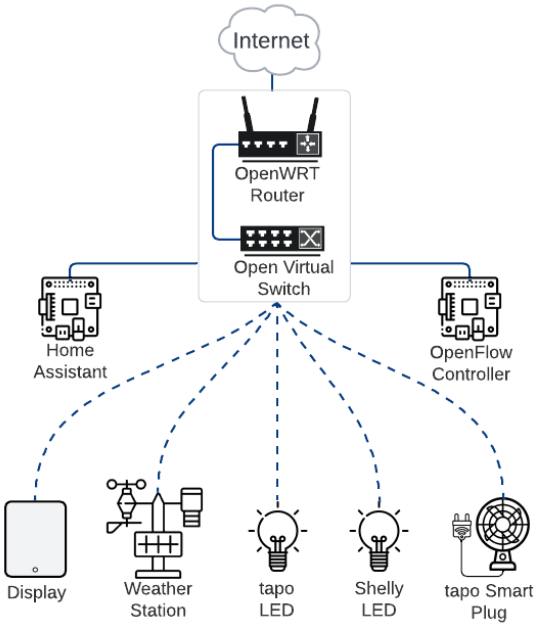


Figure 4.1: The physical layout of the test-bed.

Both the Home Assistant and the OpenFlow Controller interface with the OpenWrt router through USB 3.0 to Ethernet adapters, while all other devices establish a connection via Wi-Fi. The router's Ethernet port facilitates internet connectivity. In a typical smart home setting, this would connect to a router supplied by the Internet Service Provider (ISP) or use the OpenWrt router directly as a gateway to the ISP.

4.2 Software

The Software components, including Home Assistant [7], OpenWrt [62], and Open vSwitch (OVS) [63], will be installed and configured to facilitate network management and control. The IoT devices, such as smart bulbs, fans, and plugs, will be managed by Home Assistant and connected via Wi-Fi. To visualize the network topology, the d3.js library [64] is utilized. This concludes the most important software for this project. Additionally used software, which aids in streamlining the implementation process, is named in chapter 5.
The following versions of any software that was used in the implementation phase are listed in table 4.1:

Table 4.1: Software components and their versions.

Software	Description	Version
Home Assistant	Home automation platform	2023.8
Raspbian	Linux OS for Raspberry Pis	11
OpenWrt	Linux OS for routers	21.02.3
Open vSwitch (OVS)	Multilayer virtual switch	2.14.3
Ryu	Open-source SDN controller	4.34
d3.js	JavaScript library for visualization	7.8.5
Node.js	JavaScript runtime	18.12.0
Parcel	Web application bundler	2.9.3
Visual Studio Code	Source-code editor by Microsoft	1.81.1
Docker Desktop	Platform for running apps in containers	4.22.1

4.2.1 Home Assistant

Home Assistant is an open-source home automation platform emphasizing local control and privacy. It consolidates various smart home devices, irrespective of their manufacturers, under a single interface. Due to its vast array of community-driven integrations and plugins, coupled with robust customizability, Home Assistant has emerged as a go-to solution for smart home beginners and enthusiasts [7]. The ultimate goal is to have Home Assistant running on the Pi. During the implementation process, a Docker image of Home Assistant will be used to simplify the process.

4.2.2 OpenWRT and OVS

OpenWrt is an open-source firmware tailored for embedded devices, enhancing standard routers with extensive customization capabilities. In tandem, Open vSwitch (OVS) is a multilayer, open-source virtual switch, optimized for network automation through programmatic extensions, all while accommodating standard management interfaces and protocols. Together, they offer a powerful suite for precise network traffic management and control [62], [63]. The LuCi API of the OpenWrt router facilitates the extraction of information about connected devices [65].

4.2.3 Ryu SDN Controller

Ryu is a Python-based, open-source SDN controller. As a centralized platform, it governs network devices, like switches and routers. Supporting multiple management protocols, such as OpenFlow, Ryu's well-defined APIs facilitate the creation and management of network applications [55].

5 Implementation

This section documents the implementation process and decisions made during the implementation phase, offering transparency into the development process.

5.1 Configuration of the Network

For commencing the network configuration it is necessary to establish a typical smart home's architecture. It is reasonable to assume most homes possess a singular access point, primarily the router provided by the ISP. Given this study's reliance on OpenWrt firmware, a rudimentary network configuration is assumed, that contains one router with a single Wi-Fi access point.
As mentioned in chapter 2.2.2, smart devices employ wireless protocols like Wi-Fi, Bluetooth, Zigbee, and Z-Wave for communication. Given the thesis's emphasis on Wi-Fi networks, other protocols are set aside for the scope of this thesis and there is no need to configure any kind of gateway usually used in the context of other wireless protocols.

Wi-Fi

With OpenWrt, the router's Wi-Fi can sometimes be disabled by default, depending on the specific firmware version. For this project, the default firmware was replaced with the latest version available on the OpenWrt website, version 21.02.3 [62]. After installing the firmware, essential configurations - including setting the root password, enabling Wi-Fi, adjusting network settings, and tweaking DHCP services - are required. While these adjustments can be made using either the CLI or the web interface (usually available at 192.168.1.1), it's recommended to opt for a direct command line connection. Although SSH connections are feasible, a command line connection offers more reliability, especially during modifications that could interrupt network access. Should the configuration be executed on a dedicated router that lacks display output and support for mouse and keyboard, it's recommended to create a backup after each configuration step.
Configuration adjustments were focused on the following files located in the `etc/conf/` directory:

1. `/network:` Manages interfaces, routes, and gateways.
2. `/wireless:` Contains settings for wireless interfaces, including NATs, encryption, etc.
3. `/firewall:` Port forwards, input/output rules, and NAT configurations, are defined.

4. `/dhcp`: DHCP configurations and IP address assignments are set here.

Comprehensive documentation, such as Seppo Hätönen guide [66], provides best-practice configurations. The complete configuration utilized is available in the project repository. An overview of the content in this repository can be found in section 5.4.

API

For acquiring information regarding connected devices, an interface between the Home Assistant and the OpenWrt router is necessitated. The LuCi client-side API facilitates this connection [67]. To use this API, the router needs the installation of the LuCi.rpc package:

```
opkg update
opkg install luci-mod-rpc
```

Post-installation, API calls to the router are feasible via a simple Python script [68]:

```
from openwrt_luci_rpc import OpenWrtRpc

router = OpenWrtRpc('192.168.1.1', 'user', '******')
result = router.get_all_connected_devices(only_reachable=
    True)

for device in result:
    mac = device.mac
    name = device.hostname

    # convert class to a dict
    device_dict = device._asdict()
```

However, hardcoding credentials is a security concern. Therefore, utilizing ‘secrets.yaml‘ to store sensitive information is a recommended measure [69].

Sadly the current version of LuCi [68] has a bug since:

> “Newer OpenWrt releases (18.06+) do not use ‘reachable‘ or ‘stale‘ values as they flap constantly even when the device is inside the network. The very existence of the mac in the results is enough to determine the ”device is home”.“

It is assumed that this issue will be addressed in future versions, necessitating updates to the code provided in this thesis. For the time being, the existence of the MAC address is checked, to see, if a device is online.

SDN

The next step involves activating the SDN functionalities. The mentioned guide by Seppo Hätönen [66] is still in parts useful for this procedure, though adjustments are needed to ensure compatibility with OpenWrt. First and foremost, the OVS package must be installed on the OpenWrt router:

```
opkg install openvswitch
```

Subsequently, the default Linux bridge, which does not support OpenFlow features, is replaced with the OVS bridge:

```
# Find and remove default interfaces
brctl show
```

```
brctl delif <bridge> <interface>

# Create new bridge and add interfaces
ovs-vsctl add-br br0
ovs-vsctl add-port br0 wlan0
ovs-vsctl add-port br0 eth0
```

Due to the use of Wi-Fi client isolation, it becomes crucial to activate pvlan. This ensures that incoming packets can be routed out from their original port [70]:

```
echo 1 > /proc/sys/net/ipv4/conf/br0/proxy_arp_pvlan
```

With the bridge replaced and pvlan activated, the router should be functional after a restart. At this point, it's essential to verify that the DHCP service is running, devices can connect to the network, and the internet is accessible. The easiest test is to connect a Laptop or smartphone to the network.
In the context of this setup, WPA2 encryption was not utilized due to the known compatibility issues between hostapd, the user space daemon for wireless access points and authentication servers, and OVS in the current OpenWrt setup. A patch developed by Helmut Jacob "hschaa" addresses this limitation, allowing hostapd to interface with OVS bridges.
However, it necessitates a separate compilation of hostapd and was not applied within this testbed, though the patch can be accessed at the provided GitHub link [71].
Upon transitioning from a testbed to a real-world environment, it is strongly recommended to use WPA2 encryption to protect the network and its connected devices.
To initiate the SDN, a second Pi is necessary. For this thesis, Raspbian was chosen as the Linux distribution to be installed on it. After installing Raspbian [72], the ryu package needs to be installed and started and the switch as well as the RESTful Interface have to be activated.

```
pip3 install ryu
ryu-manager ryu.app.simple_switch_rest_13 ryu.app.ofctl_rest
```

5.2 Configuration of Home Assistant

Home Assistant and its supervisor are available in four distinct versions [7]:

- **Home Assistant OS**: Home Assistant and its supervisor for embedded devices.
- **Home Assistant Container**: Home Assistant packaged as a Docker container.
- **Home Assistant Supervised**: Docker for Home Assistant and its supervisor.
- **Home Assistant Core**: A lightweight version of Home Assistant without supervisor.

The supervisor handles tasks such as updating Home Assistant, managing add-ons, and overseeing the system's configuration.

Development Environment

For active development, the installation of Home Assistant in Docker is recommended. However, Home Assistant OS is typically most apt for standard smart home setups. For this project, the Home Assistant Container was used within Docker, as detailed in Elmar Hinz's guide [73]. Prerequisites include a GitHub account, and installation of VS Code and Docker Desktop.
If one strictly follows the guide, it results in an installation of Home Assistant Container and

an empty git repository. This repository then has to be added to Home Assistant. In the context of this thesis, the project repository already contains the complete code.

Setting up Home Assistant

To integrate the project into Home Assistant, start Home Assistant using the "Run Home Assistant" command in VS Code, accessing it via `http://localhost:8124` for configuration. The Home Assistant Community Store (HACS) needs installation next. Instructions are provided in the official guide [74]. The installation over the command line was chosen in this setup.

```
wget -O - https://get.hacs.xyz | bash -
sudo reboot
```

The next step is integrating a custom card into Home Assistant. This is done by navigating into the folder `config/www/community/` on the machine where Home assistant is installed and cloning the project repository:

```
git clone git@github.com:pfeifer-j/visualization.git
```

Within the Home Assistant filesystem, the custom card is now being imported via git. Post-cloning, it has to be registered within Home Assistant at `http://localhost:8124/config/lovelace/resources`:

Figure 5.1: Adding resources in Home Assistant.

The custom card is then added to a dashboard by selecting "Custom Card: Network Visualization" during card selection. With the router and OVS configured properly, the graph should appear, presenting the editor interface. At this juncture, the graph will close to empty as no devices have been added to the network yet.
As the final step, the Python script for API management has to be started:

```
python3 python/server.py
```

5.3 Adding IoT Devices

Before delving into the visualization software, it's useful to introduce a hand full of devices to the network. As highlighted in chapter 4, a small selection of smart devices will be integrated into the network, primarily serving as dummies to test visualization features.

Incorporating supported devices into Home Assistant is generally straightforward. By navigating to the Devices tab, many devices can be integrated without extensive configurations, given Home Assistant's rich set of default integrations. However, some devices, like the Tapo LED L530E and Tapo Smart Plug Mini Smart Wi-Fi Socket utilized in this thesis, do not natively interface with Home Assistant. Thankfully, Home Assistant's active community often fills these gaps, providing open-source solutions for a broad spectrum of devices and services. Through HACS, an integration is implemented to support the Tapo Smart Plug and LED [75]. The remaining devices specified in figure 4.1 are effortlessly added using the auto-discover function. With the network populated with Wi-Fi devices, the implementation of the visualization system can be tested out.

5.4 Implementation of the Network Visualization

5.4.1 Project Overview

This section explains the creation of the custom card, presenting its functionalities, code structures, and decisions made during the implementation phase. The architectural foundation is outlined in chapter 4. As mentioned before, Elmar Hinz's guide on starting the development served as the foundation for the implementation [73]. Visual Studio Code serves as the primary IDE for development, utilizing the Remote-SSH extension to connect the Home Assistant Container. The build process uses Parcel within a Node.js environment. TypeScript, a JavaScript superset, is the main language of choice, with parts of the project written in Python. The project's directory general structure is provided below, highlighting key components:

```
\network-visualization
├── backup
└── src
    ├── demo
    ├── python
    │   └── server.py
    ├── card.ts
    ├── editor.ts
    ├── index.ts
    ├── network.ts
    ├── view.ts
    └── css.ts
```

Visualization endeavors are primarily undertaken within the `view.js` file, through the use of d3.js [64]. The `card.ts` defines the Home Assistant card's layout, while `index.ts` bridges parameter exchanges between Home Assistant and the current implementation. `network.ts` orchestrates client-side API calls, and `python/server.py` manages server-side API interactions, interfacing with both OVS and the OpenWrt router. Customization functionalities are defined in `editor.ts`, and `css.ts` encapsulates the design specifics. Hardcoded data for a demonstrative network is located in the `demo` folder. The `backup` folder contains the configurations used for Home Assistant and the network components.
To keep it brief the following sequence diagram shows how the code functions:

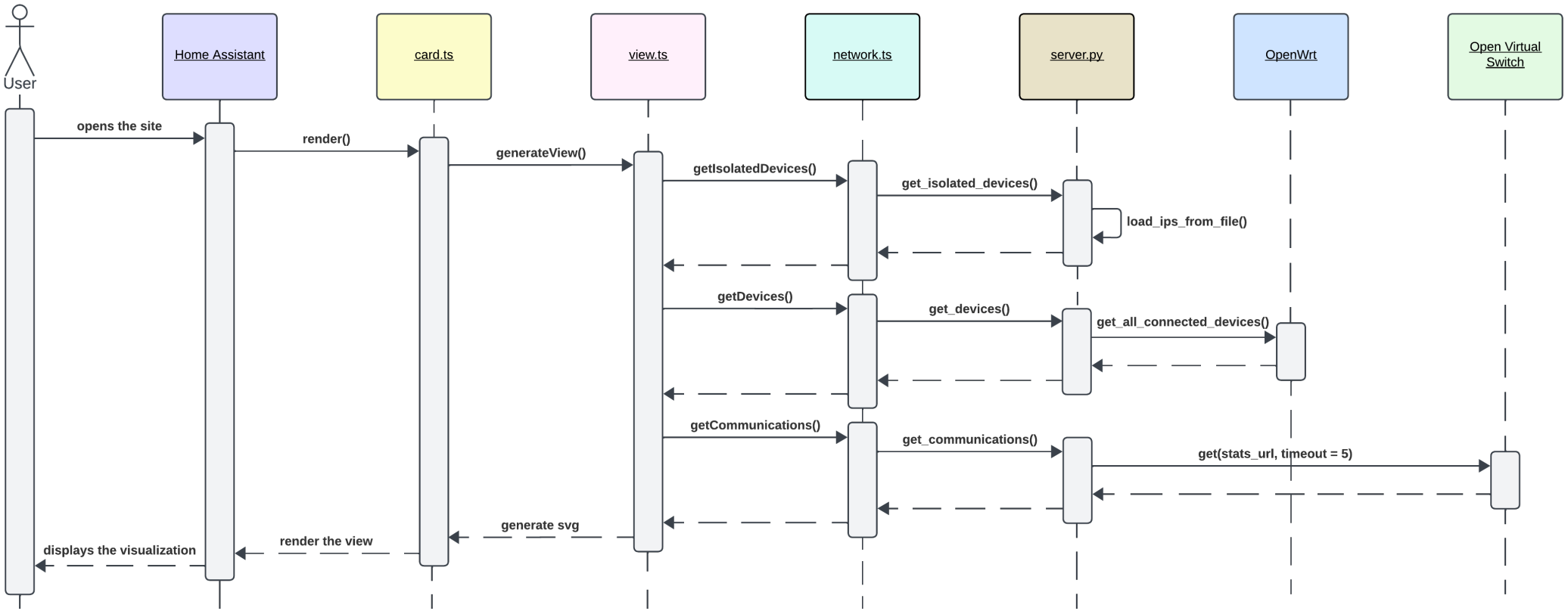


Figure 5.2: Sequence Diagram of the visualization.

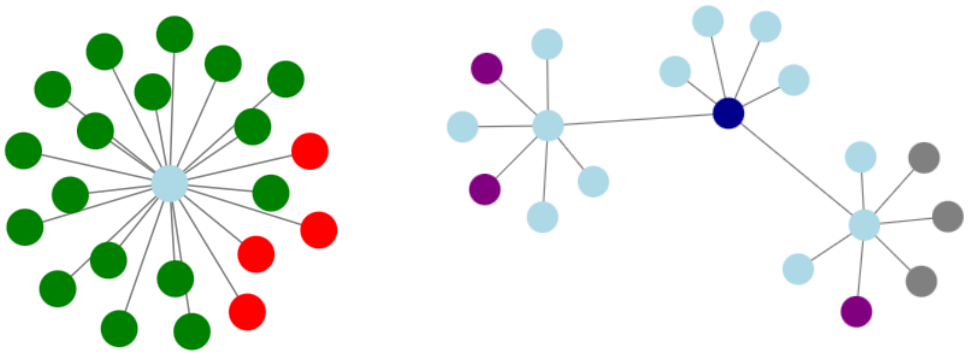
5.4.2 Implemented Features

This section delves into the features integrated into the project. Specifically, four primary functionalities were incorporated:

1. Visualization of the network and its communication flow.
2. Periodic automatic updates.
3. Preliminary network flow control, facilitating device isolation.
4. User customization options.

Feature 1: Network and Communication Flow Visualization

The network visualization is achieved by creating circle objects in d3.js, which are sourced from the JSON data obtained from API calls. These circle objects represent the nodes or devices in the network. Each entry in the JSON data contains details such as name, IP address, MAC address, host, and information on whether the device is currently reachable. However, as discussed in chapter 5.1, the "reachable" flag is not accurately set in the latest LuCi.rpc release. As a workaround, the MAC address's presence is verified instead of relying on the "reachable" flag. These details are subsequently added as attributes to the nodes. Based on the host information, a link connects the nodes to a central circle, which symbolizes the OpenWrt router by default. This represents the first of two visualization modes, called the *Physical Visualization*.
The second mode, *Software-Defined Visualization*, portrays device interconnections based on OVS switch statistics. The API call to OVS results in a specific JSON format for each flow rule. The information gained from the flow rule can be utilized to visualize the communications between devices. When a switch receives a packet, a flow rule is generated, signifying active communication between the source and destination devices specified in the rule. Depending on the configuration, flow rules can be retained for a variable duration, enabling the visualization to depict either recent or extended communication periods, depending on the configuration of the OVS. Here, broadcasts can be filtered out by ignoring the flows with broadcast addresses. Upon clicking a node, flow rules with the device's IP address as the source highlight the respective communication endpoints, the destination address. Additionally, flow rules can influence network structure, by grouping devices in the same Software-Defined subnet together. This distinction is determined by the vlan-tag within the flow rule.
Figure 5.3b serves as an example of what the visualization can look like:



(a) Physical Mode: Online nodes are colored green, offline nodes are colored red.

(b) Software-Defined Mode: Online nodes are colored lightblue, offline are colored purple, and isolated nodes are gray.

Figure 5.3: Both visualization modes showcasing different setting in the editor.

And Figure 5.4 depicts a selected node, its communication partners and the stats in the table.

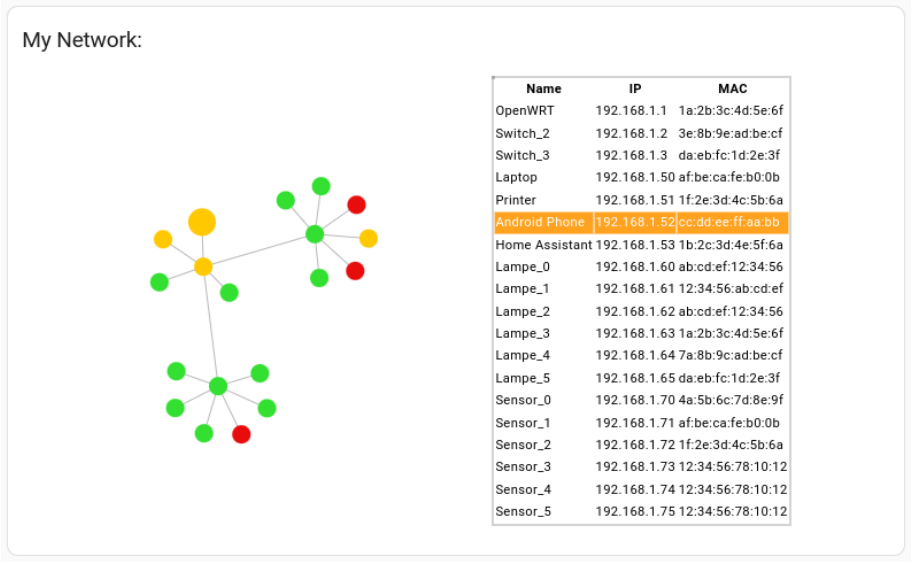


Figure 5.4: A node is selected in the visualization, with the corresponding table view displayed next to it. This visualization represents the demo network with randomized MAC addresses. The selected node appears slightly larger, and the corresponding row in the table is highlighted.

Feature 2 - Updates

While the concept behind this feature seems straightforward, its practical application poses challenges. Simply reloading the graph after a specified time interval is the obvious solution, but such an approach leads to the undesirable consequence of resetting the graph's structure on every update. Utilizing d3.js's physics engine allows nodes to be manually arranged. However, these arrangements are lost upon a full reload. For now, it was not feasible to find an effective solution to preserve the positioning of the nodes during updates. Consequently, while users have the choice to set an update interval or manually reload for updates, the nodes' positions are unfortunately not retained post-update.
Nonetheless, the basic update feature provides the user with the option to set a time interval, after which the visualization is reloaded. In the background, new API calls are made and the SVG created by d3.js is re-rendered.
In summary, while Feature 2 achieves its goal of updating the graph, it compromises on preserving node positioning. Future iterations would benefit from addressing this limitation and implementing on-change updates.

Feature 3 - Network Flow Control

The initial steps into network flow control are undertaken with this implementation. Although it's currently not possible to block specific ports or relocate devices into subnets using the interactive visualization, foundational groundwork has been established, facilitating relatively easy future updates.
At present, complete isolation of devices is achievable. To do this, a device must first be selected in the visualization, as shown in Figure 5.4. Following this, the user can press the *DELETE* key to activate isolation. It's worth noting that switches, routers, and Home Assistant itself cannot be isolated due to potential unforeseen consequences for the network.

Addresses that cant be isolated have to be specified by the user. If no further addresses are specified at least the OpenWrt router cant be isolated. Before a device is isolated, a popup notifies the user about the upcoming action, ensuring they fully comprehend its implications. Once confirmed, the node changes to a grey color (by default), signifying that a flow rule has been dispatched to the OVS in the background. The structure of a flow rule, without the fields that are not used, is illustrated as follows:

```
dl_src=11:22:33:44:55:66, dl_dst=aa:bb:cc:dd:ee:ff, actions=DROP
```

After a brief period, the rule is applied, allowing the isolated device's to solely communicate with the router.
To revoke this flow rule and reintegrate the device back into the network, the user needs to select the isolated device and press the *ENTER* key. Subsequently, the node resumes its original color, indicating its reactivated status.
Isolated devices are consistently stored in the blacklist, located under `/data/` within the project's directory structure.
This feature harbors significant potential. It's conceivable to introduce subnetting features, more nuanced isolation options including port-blocking, and direct control of smart devices, such as activating lights, directly from the graph. Further discussion on these possibilities can be found in chapter 7.

Feature 4 - Customization

Visualization can often be subjective, making customization options crucial. With this feature, users have the flexibility to adjust various elements, including size, shape, color, and movement, as shown below in Figure 5.5:

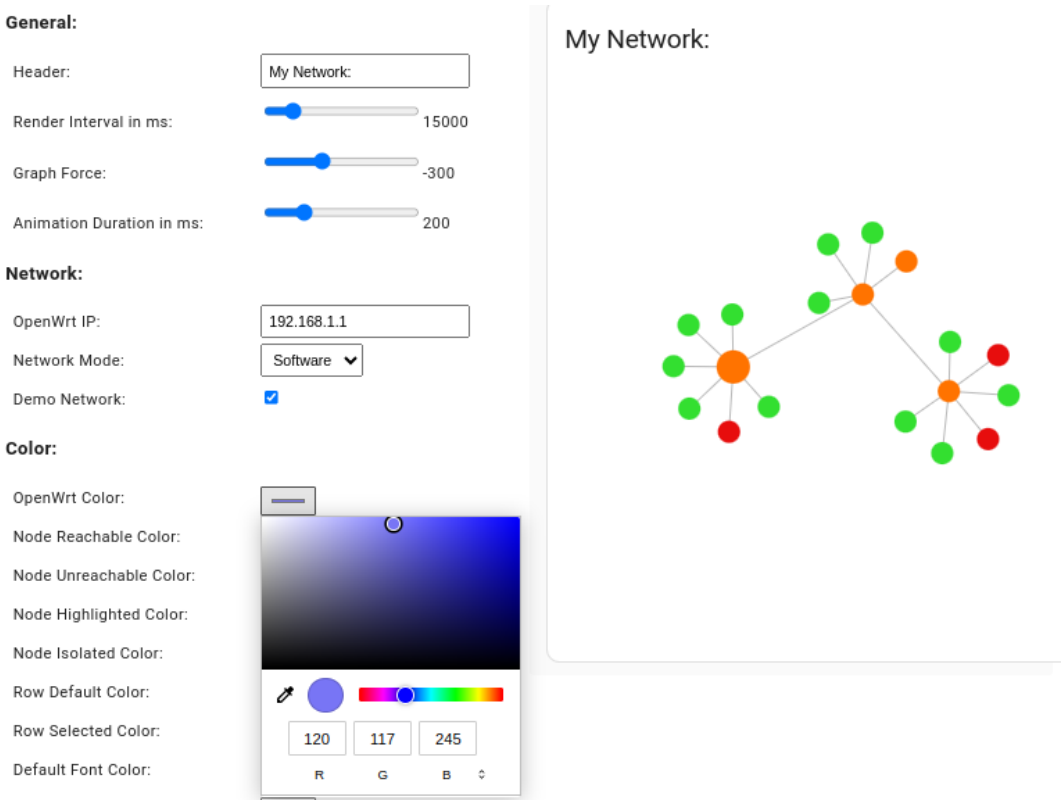


Figure 5.5: Editor of the visualization with a node selected.

Whenever a user modifies a parameter, the visualization immediately reflects the change, showcasing the updated appearance.
The following table lists all parameters that are editable:

General	
Header	Name of the network.
Render Interval	Time until an update happens.
Graph Force	Strenght of the physics engine.
Animation Duration	Animation time for events.
Network	
OpenWrt IP	IP address of the source router.
Network Mode	Network visualization mode.
Demo Network	Demo network for development.
Color	
Colors	Colors for fonts, nodes, and links in each state.
Shape	
Shape	Size for nodes and links in each state.

Table 5.1: Description of Various Parameters.

To elevate the user experience and streamline customization, color wheels and sliders have been integrated for certain parameters within the editor. These additions enhance the intuitive nature of the interface, making the customization process more user-friendly.

6 Evaluation and Discussion

In the following section the effectiveness of the system will be evaluated and discussed.

6.1 Evaluation

This evaluation emphasizes the installation, customization, and specific measurements central to this implementation. While several public software solutions exist, a direct comparison was not conducted due to the significant time and financial constraints.

6.1.1 Installation

The ease of installation is an important aspect of this implementation. Unfortunately, while HACS requires a GitHub account, not all users have one. Therefore, two scenarios have to be considered:

1. **The user has or is willing to create a GitHub account:** This is the ideal scenario. Users can utilize HACS to install custom repositories and then refer to the project repository's documentation and configuration guidelines to set up the visualization system easily. The installation process is documented and straightforward.
2. **The user prefers not to engage with GitHub:** This presents a challenge since the code must be transferred to the machine in some way. A solution would be if Home Assistant had this card integrated by default. However, achieving such integration seems distant given that this is an independent project. A significant amount of additional effort would be needed before Home Assistant would adopt the network visualization as a default feature. A more optimistic outlook might be its integration into HACS if the feature gains popularity. But even then a GitHub account would be required.

As of this thesis's writing date, the recommended approach is to set up a user account on GitHub and use the existing configuration files in the project repository. While it might not be the perfect solution compared to a native integration into Home Assistant, it remains a solid option.

6.1.2 Customization

In the context of customization, two primary topics need to be addressed. First, the editor is examined. Throughout the development of this thesis, a significant emphasis was placed on the customizability of this work. With the aid of the editor, users can adjust the font, size, layout, physics, animations, update intervals, and essentially any parameter involved in the visualization. Consequently, the design can be readily adjusted to individual preferences using the editor. For instance, users can easily switch between dark and light modes. Additionally, the application is likely suitable for those with color vision deficiencies such as color blindness. The second aspect is related to the open-source nature of this project. Admittedly, making changes beyond what the editor allows might be challenging for individuals without programming knowledge. However, it's worth noting that if users wish to alter an unavailable option in the editor or introduce a new feature, this thesis offers guidance for setting up the development environment and provides access to the open-source code and configuration.

6.1.3 Visualization Time

Due to constraints in hardware costs, performance was only measurable in a relatively small network. As illustrated in figure 4.1, the network comprises eight active participants plus a computer used for performance measurement. Within this scale, the average time taken to visualize consistently registered below half a second for page load and graph rendering without waiting for the arrangement of the graph. Specifically, the duration fluctuated between 250 ms and 475 ms across five measurements.
The force-directed graph leads to a situation where nodes continue to move after the initial render. The time they take to stabilize varies based on the strength setting of the physics engine. This strength determines node proximity: a high value, such as 100, pulls the nodes together quickly, whereas a lower value, like the default of -300, spaces the nodes further apart. At an extreme low of -1000, visualization time significantly increases. With a strength of 100, the stabilization takes 150 ms, but the nodes are very closely packed. At the default strength of -300, it takes around 350 ms, with the nodes comfortably spaced. In contrast, a strength of -1000 results in a duration of approximately 1.3 seconds, and the nodes are very distant. These numbers represent averages from five trials.
In summary, for this small-scale setup and using default settings, the graph requires between 600 ms and 825 ms to fully load and stabilize.
To assess visualization time on a larger scale, demo networks with 20, 100, and 250 devices were constructed. Here, API calls were replaced with static data, allowing the graph to load instantly and no further hardware to be needed. With the default physics strength of -300, the stabilization times were slightly longer than the small-scale setup: around 0.653, 1.320, and 3.664 seconds, respectively. However, it's worth noting that in larger networks, API calls might introduce additional delays, although graph rendering remains fairly consistent as long as the rendering hardware is not saturated.
The page load time measurements were taken using Google Chrome Version 116.0.5845.142 on a Windows 10 machine, equipped with an Intel Core i7-4790k at 4.0 GHz and 16 GB of DDR3 RAM. The Raspberry Pi Model 4B managed the API calls. Measurement of the graphs stabilization duration was achieved by logging event times in JavaScript to the Development Console in Google Chrome.

6.1.4 Isolation Time

The time taken for the API call to update the flow is almost immediate. However, the more important metric is the duration until the new rule effectively takes action. Using a stopwatch, a computer, and a laptop, measurements were taken five times to determine the time until isolation. Initially, a ping was sent to Google's DNS server from the laptop:

```
ping -t 8.8.8.8
```

After several successful pings, the laptop was isolated using the visualization, simultaneously starting the stopwatch. The interval until the ping was lost ranged from 3 to 7 seconds. This method of measurement, while not the most accurate, was chosen due to the lack of a better alternative. Reintegrating the isolated laptop back into the network took slightly less time, with durations between 2 to 5 seconds.
For faster application of flow rules, it's recommended to use a dedicated OpenWrt router instead of a Raspberry Pi running OpenWrt. A direct ethernet connection is preferable over a USB 3.0 to ethernet adapter, as used in these tests. Further optimization might be achieved with improvements in API management.

6.1.5 Network Delay

Upon measuring the performance of the network, no noticeable difference was observed between an OpenWrt router using its default configuration and the same router using OVS. Further testing, especially in larger-scale networks, is necessary to draw more comprehensive conclusions.

6.2 Discussion

Following the evaluation of the implementation's features, the next phase is to address the initially posed research questions.

6.2.1 Question 1: Accessibility

A primary step in addressing the question of how to make Wi-Fi network visualization more accessible for smart home users is to implement an intuitive interface. This interface should effortlessly integrate user-friendly features like drag-and-drop functionalities, distinct animations, and clearly labeled icons." This ensures users, regardless of their technical background, can seamlessly comprehend the connections within their network. But before the visualization can be used, it has to be configured, which is, in itself, an arguably more important step. Embedding the visualization directly within commonly used platforms, such as the Home Assistant dashboard, is a strategic move. It places the network information right at the users' fingertips, allowing them to gain insights about their network without delving into complex tools. To further smooth the learning curve, introducing users to this tool with comprehensive documentation can be a deciding factor. These documentations have to include how to configure and use the visualization system to provide an entry point for inexperienced users. After a user has installed the software, it is crucial to provide personalization options. Recognizing the diverse needs of smart home users means granting them the freedom to modify visual aspects, tweak view settings, and ensure that their visualization reflects their unique requirements. Coupled with this is the necessity for regular updates of the view to mirror the state of the network, empowering the user to promptly detect and mitigate potential issues.
Moreover, as the world of smart homes continues to expand, the visualization tool should stay up to date. This means routine updates ensuring compatibility with new devices and versions of the software used for implementation, like OpenWrt, OVS, Ryu, and so on. Beyond the tool itself, accessibility also encapsulates the support ecosystem around it. This means providing answers to struggling users who want to try the system. Extending beyond platforms like GitHub to dedicated websites with help sections would be a valuable addition. In conclusion, enhancing Wi-Fi network visualization's accessibility revolves around intuitive design, seamless integration, user empowerment through personalization, and continuous

support. Adopting these strategies can transform this project into a valuable tool for smart home users, ensuring more insight into their connected environments.

6.2.2 Question 2: Introducing SDNs into Smart Homes

To introduce SDNs into smart homes, a change in perspective and hardware is necessary.
The first and foremost change required is in mindset. Unfortunately, given the hype around IoT devices [1], many of these devices lack robust security standards. Even average smart home users should be made aware of the risks associated with using IoT devices at home. Raising this awareness can be achieved through media, and this responsibility falls on journalists, publications, and others.
Once there's increased awareness and interest in the topic, the next step is hardware modification. To utilize SDN in smart homes, dedicated hardware that supports SDN functionalities is essential. As mentioned in chapters 4 and 5, such hardware exists and can be integrated into current networks. Considering that not all network devices support SDN, it's imperative to upgrade to SDN-compatible routers and switches. These upgrades will ensure centralized control, which allows the features SDNs provide.
Given the complexity of networking, routing, and applying flow rules, offering an intuitive option for SDN control is vital. This thesis aims to provide an entry point for such a solution. But it's not only about providing tools; educating the user is equally significant, especially for the less tech-savvy. Offering resources or training can empower users to use the full potential of their SDN-enabled smart home. In a rapidly evolving SDN landscape, staying updated is crucial.
Integration with mainstream smart home systems can significantly improve user experience by offering unified control over smart devices and the network. By addressing these foundational aspects, the shift to SDN can be streamlined and made efficient. However, it may still be some time before software-defined networks become commonplace in households.

6.2.3 Question 3: Visualization Techniques

To determine which visualization techniques are most suitable for depicting an SDN, one can draw upon commonly used visualization techniques. If a basic framework exists, a seamless transition between the physical and Software-Defined views can be achieved, by implementing a force-directed graph to represent the network itself. Network data has been effectively visualized using established software such as Grafana [46] and Prometheus [47] for years, with tools like charts, plots, and heatmaps. For an insightful representation of SDN, a robust API, like the Ryu API, proves to be very useful. This facilitates the retrieval of statistics from the switch, which can then be displayed to the user, offering a clearer understanding of smart home connections.
To depict communication flows, it's a viable option to just represent both the source and destination. For larger networks, the SDN would need to store path information for visualization purposes, or otherwise shortest path algorithms would need to be employed to represent the route a packet takes. To depict the path would be a valuable addition. For now only endpoints are highlighted, which is a more reliable method.
As outlined in chapter 3, a 3D visualization of the SDN could provide a comprehensive perspective. However, given its complexity and the resources it demands, it wasn't explored in depth in this thesis.

7 Conclusion

In this thesis, a specialized tool was developed for network visualization and management in smart homes, enabling informed user decisions within a SDN and spreading awareness in terms of IT security. Upon reviewing various visualization techniques, it was demonstrated that networks are best visualized using graphs, especially force-directed graphs, which are effective due to seamlessly adapt to network changes. This visualization tool integrates the open-source software OpenWRT, Ryu, and Home Assistant, utilizing the d3.js framework for visualization to make the software accessible to a broad user range. To simplify data extraction from the OVS bridge and the OpenWRT router the tool leverages the LuCi.rpc and Ryu's RESTful API. Ryu was selected for its extensive features, cost-effectiveness, and robust Python support. The integration with OpenWRT and OVS was motivated by their open-source nature and synergy with Ryu. Incorporating the tool into Home Assistant, a well-known open-source software, facilitates user adoption. Beyond visualization, this interactive platform enables users to monitor network communication and manage devices by selectively isolating them, thereby enhancing overall network control.
Evaluation of the tool highlighted its relatively easy installation process and numerous customization options. Tested in a small network environment with nine devices, its performance was notably effective, emphasizing its utility for Wi-Fi network visualization. Even in simulated larger networks, the tool proved to be smoothly operable.
When talking about what this thesis achieves, it has to be mentioned that this integration promises to amplify understanding and security in the smart home domain. Beyond the technical achievements, the practical implications of this thesis are significant. In times where a smart home is filled with IoT devices - from smart fridges to intelligent lighting systems - security isn't just a technical concern but a matter of personal well-being. A breach or hidden actor in a smart home network isn't just about data but could impact the physical security of inhabitants. By making advanced technologies accessible and elevating IoT security, this thesis provides an option to increase a homeowner's peace of mind.
Yet, as with all research, limitations exist. The tool was primarily tested in smaller network settings, leaving questions regarding scalability in networks with over 50 devices. Its exclusive focus on the Home Assistant platform means other platforms are overlooked. Future endeavors could explore extending its application to other smart home assistants or even developing it into standalone software. Improvements in its user interface, particularly in refining the editor's intuitiveness, would enhance user experience. Challenges, like issues with LuCi.rpc and node positioning, pinpoint areas for software improvement. There's also potential in exploring compatibility with popular routers, such as Fritz!Box and Speedport.

The evolution of smart home networking is accelerating. As these homes evolve, so do the complexities of the supporting networks. This research is a foundational step in the

domain of SDN visualization and management. With opportunities spanning from technical adjustments to compatibility extensions, the future signals promising advances aimed at better serving users in the dynamic realm of smart home networking.

Glossary

ACI Application Centric Infrastructure.

API Application Programming Interface.

APIC Application Policy Infrastructure Controller.

CLI Command Line Interface.

d3.js Data-Driven Documents - Java Script.

DHCP Dynamic Host Configuration Protocol .

GUI Graphical User Interface.

HACS Home Assistant Community Store.

hostapd Host Access Point Daemon.

IDE Integrated Development Environment.

IoT Internet of Things.

IP Internet Protocol.

ISP Internet Service Provider.

IT Information Technology.

JSON JavaScript Object Notation.

LED Light-emitting diode.

LuCi Network address translation.

MAC Media Access Control.

MPLS Multiprotocol Label Switching.

NAT Network address translation.

Neo4j graph database management system.

NMAP Network Mapper.

ODL OpenDayLight.

ONOS Open Network Operating System.

OpenWrt OpenWRT stands for Open Wireless RouTer.

OVS Open vSwitch.

rpc Remote Procedure Call.

Ryu The opensource SDN controller.

SDN Software-Defined Network.

SNMP Simple Network Management Protocol.

SSH Secure Socket Shell.

Wi-Fi Wireless Fidelity.

WPA2 Wi-Fi Protected Access 2.

ZHA Zigbee Home Automation.

Bibliography

[1] Kolya Hnatyuk. Internet of things (iot) statistics: 2022/2023. https://marketsplash.com/internet-of-things-statistics, 2023. Accessed: 30 August 2023.

[2] Statista Research Department. Number of users of smart homes worldwide 2018-2027. https://www.statista.com/forecasts/887613, 2023. Accessed: 17 August 2023.

[3] Sagar Joshi. 70 iot statistics to unveil the past, present, and future of iot. https://learn.g2.com/iot-statistics, 2023. Accessed: 15 June 2023.

[4] Amazon alexa. https://alexa.amazon.com/, 2023. Accessed: 25 July 2023.

[5] Google home. https://home.google.com/welcome/, 2023. Accessed: 25 July 2023.

[6] Apple home. https://www.apple.com/de/home-app/, 2023. Accessed: 25 July 2023.

[7] Home Assistant Developers. Home Assistant. https://www.home-assistant.io/, 2023. Accessed: 25 July 2023.

[8] Rahim Masoudi and Ali Ghaffari. Software defined networks: A survey. *Journal of Network and Computer Applications*, 67:1–25, 2016. Accessed: 25 July 2023.

[9] Salim Danbatta and Asaf Varol. Comparison of zigbee, z-wave, wi-fi, and bluetooth wireless technologies used in home automation, 06 2019. Accessed: 15 June 2023.

[10] Parul Gandhi and Jyoti Pruthi. *Data Visualization Techniques: Traditional Data to Big Data*, pages 53–74. Springer Singapore, Singapore, 2020.

[11] Rahul Anand Sharma, Elahe Soltanaghaei, Anthony Rowe, et al. Lumos: Identifying and localizing diverse hidden {IoT} devices in an unfamiliar environment. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1095–1112, 2022.

[12] John Goodall. Introduction to visualization for computer security, 01 2007.

[13] Mohammed Asif Khan, Bhargavi Goswami, and Saleh Asadollahi. Data visualization of software-defined networks during load balancing experiment using floodlight controller. *Data Visualization: Trends and Challenges Toward Multidisciplinary Perception*, pages 161–179, 2020.

[14] Matthias Kraus, Johannes Fuchs, Björn Sommer, et al. Immersive analytics with abstract 3d visualizations: A survey. In *Computer Graphics Forum*, volume 41, pages 201–229. Wiley Online Library, 2022.

[15] Peter Rodgers, Gem Stapleton, Bilal Alsallakh, et al. A task-based evaluation of combined set and network visualization. *Information Sciences*, 367, 06 2016.

[16] S Margret Anouncia, Hardik A Gohel, and Subbiah Vairamuthu. *Data Visualization*. Springer, 2020.

[17] V. K. Balakrishnan. *Schaum's Outline of Theory and Problems of Graph Theory*. The McGraw-Hill Companies, Inc., United States of America, 1997. Sponsoring Editor: Barbora Gilson; Production Supervisor: Suzanne Rapcavage; Editing Supervisor: Maureen B. Walker.

[18] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.

[19] Ben Shneiderman and Aleks Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006.

[20] Muhammad Raisul Alam, Mamun Bin Ibne Reaz, and Mohd Alauddin Mohd Ali. A review of smart homes—past, present, and future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1190–1203, 2012.

[21] Manuel Silverio-Fernández, Suresh Renukappa, and Subashini Suresh. What is a smart device?-a conceptualisation within the paradigm of the internet of things. *Visualization in Engineering*, 6(1):1–10, 2018.

[22] Bergur Thormundsson. Main devices used with voice assistants in the u.s. 2021, by brand, 2023. Accessed: 30 August 2023.

[23] Cchangedev. Smart tech in the american home: Distrust or trust?, Jun 2022.

[24] Bluetooth SIG. Bluetooth Technology Website. `https://www.bluetooth.com/`, 2023. Accessed: 25 July 2023.

[25] Zigbee Alliance. Zigbee Official Website. `https://www.zigbee.org/`, 2023. Accessed: 25 July 2023.

[26] Z-Wave Alliance. Z-Wave Official Website. `https://z-wavealliance.org/`, 2023. Accessed: 25 July 2023.

[27] IEEE. Ethernet Technology. `https://www.ieee802.org/3/`, 2023. Accessed: 25 July 2023.

[28] Tuya Support. Can smart devices be controlled only through the lan without using cloud services? `https://support.tuya.com/en/help/_detail/K9tjtiy33x3qf`, 2023. Accessed: 01 September 2023.

[29] Wei Zhou, Yan Jia, Yao Yao, et al. Discovering and understanding the security hazards in the interactions between {IoT} devices, mobile apps, and clouds on smart home platforms. In *28th USENIX security symposium (USENIX security 19)*, pages 1133–1150, 2019.

[30] Charlie Sorrel. Why anker's eufy cameras upload scandal shows the dangers of home automation. `https://www.bit.ly/Ankers-Eufy-Cameras`, 2022. Accessed: 30 August 2023.

[31] Marie Chan, Daniel Estève, Christophe Escriba, et al. A review of smart homes—present state and future challenges. *Computer Methods and Programs in Biomedicine*, 91(1):55–81, 2008. Accessed: 25 July 2023.

[32] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, et al. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.

[33] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, et al. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[34] Open Networking Foundation. Openflow switch specification (version 1.5.1). Technical report, Open Networking Foundation, 2015. Accessed: 25 July 2023.

[35] Nick McKeown, Tom Anderson, Hari Balakrishnan, et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[36] Open Networking Foundation. Onf sdn evolution. Technical Recommendation TR-535, Open Networking Foundation, 2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303, 09 2016. Disclaimer: THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER. All rights reserved. Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation.

[37] Intravue. `http://www.i-vue.com/iv3/help/IntravueHelp.htm`, 2023. Accessed: 25 July 2023.

[38] Wiviz. https://wiki.dd-wrt.com/wiki/index.php/Wiviz, 2023. Accessed: 13 August 2023.

[39] HACS Developers. Home Assistant Community Store (HACS). `https://hacs.xyz/`, 2023. Accessed: 30 August 2023.

[40] Home Assistant. Nmap tracker. `https://www.home-assistant.io/integrations/nmap_tracker/`, 2023. Accessed: 25 July 2023.

[41] zha ng. zha-map: A visualization tool for zigbee home automation networks in home assistant. `https://github.com/zha-ng/zha-map`, 2023. Accessed: 25 July 2023.

[42] Jeppe Ladefoged (ladefoged81). Device tracker - first home / last home. `https://community.home-assistant.io/t/device-tracker-first-home-last-home/30036`, 2017. Accessed: 30 August 2023.

[43] Alexsandro Pires (alexspires). Zigbee network map - red dashed lines. `https://community.home-assistant.io/t/zigbee-network-map-red-dashed-lines/216670`, 2020. Accessed: 30 August 2023.

[44] Adizanni. floor3d-card: A 3d floor plan card for home assistant. `https://github.com/adizanni/floor3d-card`, 2023. Accessed: 25 July 2023.

[45] dedi. Community highlights - 4th edition. https://community.home-assistant.io/t/community-highlights-4th-edition/175261/2, March 2020. Accessed: 30 August 2023.

[46] Grafana. `https://grafana.com/`, 2023. Accessed: 25 July 2023.

[47] Prometheus. `https://prometheus.io/`, 2023. Accessed: 25 July 2023.

[48] Graphlytic pricing. `https://graphlytic.biz/pricing/`, 2023. Accessed: 25 July 2023.

[49] Netcrunch pricing. `https://www.adremsoft.com/pricing/`, 2023. Accessed: 25 July 2023.

[50] Pang-Wei Tsai, Chun-Wei Tsai, Chia-Wei Hsu, et al. Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, 12(4):3958–3969, 2018.

[51] Chung-Sheng Li and Wanjiun Liao. Software defined networks. *IEEE Communications*

Magazine, 51(2):113–113, 2013.

[52] Opendaylight. https://www.opendaylight.org/, 2023. Accessed: 13 August 2023.

[53] Onos: Open network operating system. https://onosproject.org/, 2023. Accessed: 13 August 2023.

[54] Floodlight openflow controller. https://www.projectfloodlight.org/floodlight/, 2023. Accessed: 13 August 2023.

[55] Ryu sdn framework. https://ryu.readthedocs.io/en/latest/, 2023. Accessed: 13 August 2023.

[56] Cisco apic. https://www.cisco.com/c/en/us/products/cloud-systems-management/application-policy-infrastructure-controller-apic/index.html, 2023. Accessed: 13 August 2023.

[57] Vmware nsx. https://www.vmware.com/products/nsx.html, 2023. Accessed: 13 August 2023.

[58] Juniper northstar controller. https://www.juniper.net/us/en/products-services/sdn/northstar-controller/, 2023. Accessed: 13 August 2023.

[59] OpenDaylight Project. Viewing network topology. https://nexus.opendaylight.org/content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/_viewing_network_topology.html, 2023. Accessed: 30 August 2023.

[60] Raspberry pi 4 model b. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, 2023. Accessed: 13 August 2023.

[61] Intel. What is a nuc? https://www.intel.com/content/www/us/en/products/docs/boards-kits/nuc/what-is-nuc-article.html, 2023. Accessed: 01 September 2023.

[62] Openwrt project. https://openwrt.org/, 2023. Accessed: 13 August 2023.

[63] Open vswitch. https://www.openvswitch.org/, 2023. Accessed: 13 August 2023.

[64] D3.js - data-driven documents. https://d3js.org/, 2023. Accessed: 13 August 2023.

[65] Finbarr Brady. openwrt-luci-rpc documentation release 1.1.16. https://readthedocs.org/projects/openwrt-luci-rpc/downloads/pdf/stable/, March 2023. Accessed: 25 July 2023.

[66] Seppo Hätönen. Wifisdn - university of helsinki wiki. https://wiki.helsinki.fi/display/WiFiSDN, 2023. Accessed: 13 June 2023.

[67] luci-mod-rpc - openwrt packages. https://openwrt.org/packages/pkgdata/luci-mod-rpc, 2023. Accessed: 13 August 2023.

[68] Openwrt luci rpc documentation. https://readthedocs.org/projects/openwrt-luci-rpc/downloads/pdf/stable/, 2023. Accessed: 13 August 2023.

[69] Secrets - home assistant. https://www.home-assistant.io/docs/configuration/secrets/, 2023. Accessed: 13 August 2023.

[70] Juniper Networks. Understanding private vlans. https://www.juniper.net/documentation/us/en/software/junos/multicast-l2/topics/topic-map/private-vlans-qfx-series.html, 2023. Accessed: 16 June 2023.

[71] Helmut Jacob "hschaa". Hostapd patch for open vswitch. https://github.com/hschaa/hostapd/commit/c89daaeca4ee90c8bc158e37acb1b679c823d7ab#diff-165dd5a1681d9394993972f6923fddf8R153, 2023. Accessed: 13 August 2023.

[72] Raspberry pi os. `https://www.raspberrypi.com/software/`, 2023. Accessed: 03 June 2023.

[73] Elmar Hinz. Home assistant tutorials. `https://github.com/home-assistant-tutorials/`, 2023. Accessed: 13 August 2023.

[74] Hacs: Download and installation. `https://hacs.xyz/docs/setup/download/`, 2023. Accessed: 13 August 2023.

[75] Andrea Petreti. Home assistant tapo p100 integration. `https://github.com/petretiandrea/home-assistant-tapo-p100`, 2023. GitHub repository.

List of Figures

List of Tables