



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Vergleich und Evaluation von Containerlab und Kathara zur Simulation von Netzwerken sowie Validierung anhand realer Netzwerkexperimente im Labor

Bachelorarbeit

von

Brinda Olivia Ngangya Ndoutchou

Matrikelnummer: s0586746

Fachbereich 1 – Energie und Information –
der Hochschule für Technik und Wirtschaft Berlin

zur Erlangung des akademischen Grades

Bachelor of Engineering (B. Eng.)

im Studiengang

Informations- und Kommunikationstechnik

Tag der Abgabe: 07.09.2026

Erstgutachten: Prof. Dr. Thomas Scheffler

Zweitgutachten: Herr Detlev Schäfer

Inhaltsverzeichnis

1	Einleitung	1
1.1	Aufgabenbeschreibung	1
1.2	Problemstellung und Motivation	1
1.3	Forschungsfragen	1
2	Grundlagen	3
2.1	Begriffserklärungen	3
2.1.1	Netzwerkvirtualisierung und Netzwerkemulation	3
2.1.2	Docker	3
2.1.3	Container	4
2.1.4	Image	4
2.1.5	YAML-Datei	4
2.1.6	Startup-Datei	5
2.1.7	Dynamic Host Configuration Protocol (DHCP)	5
2.1.8	DHCP Option 121 (RFC 3442)	5
2.1.9	Apache HTTP Server	5
2.1.10	WebDAV	6
2.2	Containerlab	6
2.2.1	Architektur und Funktionsweise	6
2.2.2	Unterstützte Kinds-Arten	6
2.3	Kathara	7
2.3.1	Architektur und Funktionsweise	7
2.3.2	Unterstützte Gerätetypen	7
3	Konzeption / Architektur	8
3.1	Beschreibung der drei Szenarien	8
3.2	Auswahl der Netzwerkgeräte	9
3.2.1	Containerlab	9
3.2.2	Kathara	9
3.2.3	Physischen Labor	10
4	Umsetzung im physischen Cisco-Labor	11
4.1	Vorbereitung der Cisco Labor Testumgebung	11
4.2	Umsetzung Szenario 1	12
4.2.1	Verwendete Hardware und Software	12
4.2.2	Topologie	12

4.2.3	Konfiguration	13
4.2.4	Routerkonfiguration	13
4.2.5	Switch-Konfiguration	15
4.2.6	Durchführung und Testergebnisse	15
4.3	Umsetzung Szenario 2	17
4.3.1	Topologie	17
4.3.2	Konfiguration	19
4.3.3	Durchführung und Testergebnisse	23
4.3.4	Testergebnisse	25
4.4	Umsetzung Szenario 3	25
4.4.1	Topologie	25
4.4.2	Konfiguration	27
4.4.3	Durchführung	29
4.4.4	Testergebnisse	31
5	Umsetzung in Containerlab	32
5.1	Installation	32
5.1.1	Installation von Docker Engine	32
5.1.2	Installation von Containerlab	33
5.1.3	Installation des Arista-cEOS-Images	33
5.2	Umsetzung Szenario 1	34
5.2.1	Topologie	34
5.2.2	Konfiguration	36
5.2.3	Durchführung und Testergebnisse	40
5.3	Umsetzung Szenario 2	45
5.3.1	Topologie	45
5.3.2	Konfiguration	47
5.3.3	Durchführung	51
5.3.4	Testergebnisse	55
5.4	Umsetzung Szenario 3	55
5.4.1	Topologie	55
5.4.2	Konfiguration	57
5.4.3	Durchführung	61
5.4.4	Testergebnisse	65
5.4.5	Grafische Benutzeroberfläche von Containerlab	65
5.4.6	Bewertung der praktischen Umsetzung mit Containerlab	67
6	Umsetzung in Kathara	68
6.1	Installation und Vorbereitung	68
6.1.1	Installation von Kathara	68
6.1.2	Verwendete Docker-Images	69
6.2	Umsetzung Szenario 1	70
6.2.1	Topologie	70

6.2.2	Konfiguration	70
6.2.3	Durchführung und Testergebnisse	74
6.3	Umsetzung Szenario 2	79
6.3.1	Topologie	79
6.3.2	Konfiguration	80
6.3.3	Durchführung	89
6.3.4	Testergebnisse	92
6.4	Umsetzung Szenario 3	92
6.4.1	Topologie	92
6.4.2	Konfiguration	93
6.4.3	Konfiguration der Topologiedatei	93
6.4.4	Durchführung	97
6.4.5	Testergebnisse	101
7	Vergleich und Evaluation der drei Umgebungen	102
7.1	Benutzerfreundlichkeit und Implementierungsaufwand	102
7.2	Ressourcenverbrauch	103
7.3	Realitätsnahe der Simulationsumgebungen	103
7.4	Stärken und Schwächen im Vergleich	104
8	Fazit	107
	Anlage	109
	Abbildungsverzeichnis	110
	Tabellenverzeichnis	111
	Quelltextverzeichnis	112
	Literaturverzeichnis	115
	Eigenständigkeitserklärung und Erklärung zur Nutzung von KI	119
	KI-Verzeichnis	119

Kurzfassung

Diese Arbeit untersucht die Eignung der Netzwerkemulationsumgebungen Containerlab und Kathara für die Erstellung und Durchführung praxisnaher Netzwerkszenarien. Ziel ist es, beide Emulatoren hinsichtlich Funktionsumfang, Benutzerfreundlichkeit, Performance und Realitätsnähe zu vergleichen sowie ihre Eignung für den Einsatz in der Hochschullehre zu bewerten.

Hierzu wurden vergleichbare Netzwerktopologien in Containerlab und Kathara in Containerlab und Kathara aufgebaut und verschiedene Netzwerkfunktionen praktisch untersucht. Dazu gehören unter anderem Routing mit OSPF, VLANs, Firewall-Regeln, DHCP, Multicast. Zusätzlich wurde der Datenverkehr mit Wireshark bzw. tcpdump aufgezeichnet. Ergänzend erfolgte ein Vergleich der in Containerlab und Kathara erzielten Ergebnisse mit einer realen Netzwerkumgebung.

Die Untersuchungen zeigen, dass mit beiden Werkzeugen unterschiedliche Netzwerkszenarien umgesetzt und typische Netzwerkfunktionen praxisnah getestet werden können. Gleichzeitig wurden Unterschiede bei der Konfiguration, der verwendeten Netzwerksoftware, der Performance und dem Verhalten einzelner Protokolle festgestellt.

Insgesamt bieten Containerlab und Kathara geeignete Möglichkeiten zur Durchführung virtueller Netzwerklabore. Die Untersuchungen zeigen jedoch, dass die eingesetzte Netzwerkemulationsumgebung einen Einfluss auf die Messergebnisse haben kann und bei deren Auswertung berücksichtigt werden sollte.

Abstract

This thesis investigates the suitability of the network emulation environments Containerlab and Kathara for creating and running practical network scenarios. The aim is to compare both emulators in terms of functionality, usability, performance, and realism, as well as to evaluate their suitability for use in higher education.

For this purpose, comparable network topologies were implemented in Containerlab and Kathara, and various network functions were practically investigated. These include, among others, OSPF routing, VLANs, firewall rules, DHCP, and multicast. In addition, network traffic was captured using Wireshark and tcpdump. The result obtained with Containerlab and Kathara were also compared with those from a real network environment.

The result show that both tools can be used to implement different network scenarios and to test typical network functions under practical conditions. At the same time, differences were identified in configuration, the network software used, performance, and the behavior of individual protocols.

Overall, Containerlab and Kathara provide suitable solutions for conducting virtual network labs. However, the results also show that the network emulation environment used can influence measurement results and should therefore be taken into account when evaluating and interpreting them.

Kapitel 1: Einleitung

1.1 Aufgabenbeschreibung

Im Rahmen dieser Bachelorarbeit werden die beiden Netzwerkvirtualisierungsumgebungen Containerlab und Kathara untersucht und miteinander verglichen. Zu diesem Zweck werden verschiedene Netzwerkszenarien in beiden Umgebungen aufgebaut und getestet.

Die Ergebnisse werden anschließend mit einer realen Netzwerkumgebung der Hochschule verglichen, um die Realitätsnähe der beiden Software zu bewerten.

1.2 Problemstellung und Motivation

Studierende in Studiengängen im Bereich Netzwerktechnik sind häufig auf physische Netzwerklabore angewiesen, die zeitlich und räumlich eingeschränkt zugänglich sind. Dies erschwert das selbstständige Üben und Studierendenexperimente außerhalb der Hochschule erheblich.

Software wie Containerlab und Kathara könnte das Problem lösen, indem sie es ermöglicht, einfache und komplexe Netzwerktopologien auf einem lokalen Rechner aufzubauen und zu testen. Solche Software bieten große Vorteile in der Lehre, da Studierende unabhängig von Laborzeiten und physischer Infrastruktur arbeiten können.

Allerdings stellt sich die Frage, inwiefern diese virtuellen Umgebungen das Verhalten realer Netzwerke zuverlässig und realitätsnah widerspiegeln. Für den Einsatz in der Ausbildung ist es entscheidend, dass Simulationen möglichst nah an realen Bedingungen liegen. Vor diesem Hintergrund besteht die Notwendigkeit, den Funktionsumfang, die Benutzerfreundlichkeit, die Performance und die Realitätsnähe solcher Werkzeuge systematisch zu untersuchen und mit realen Netzwerkumgebungen zu vergleichen.

1.3 Forschungsfragen

Die Arbeit orientiert sich an den folgenden Forschungsfragen:

- Wie realitätsnah sind die mit Containerlab und Kathara simulierten Netzwerkumgebungen im Vergleich zu einer realen Netzwerkumgebung?

- Worin unterscheiden sich Containerlab und Kathara hinsichtlich Funktionsumfang, Benutzerfreundlichkeit und Performance?
- Wie gut funktioniert die Aufzeichnung des Datenverkehrs mittels Wireshark an unterschiedlichen Punkten in den simulierten Netzen?
- Können auf den Docker-Containern auch konkrete Anwendungen, beispielsweise Webserver und Browser, gestartet und ausgeführt werden?
- Welches der beiden Netzwerkemulatoren eignet sich besser für den selbstständigen Einsatz in der Hochschullehre?

Kapitel 2: Grundlagen

2.1 Begriffserklärungen

2.1.1 Netzwerkvirtualisierung und Netzwerkemulation

Bei der Netzwerkvirtualisierung werden physische Netzwerkkomponenten durch Software virtuell bereitgestellt. Dadurch können mehrere virtuelle Netzwerke auf derselben physischen Hardware aufgebaut und betrieben werden, ohne dass für jede Komponente ein eigenes physisches Gerät benötigt wird.

Die Netzwerkemulation geht einen Schritt weiter. Dabei wird versucht, das Verhalten realer Netzwerkgeräte in einer virtuellen Umgebung nachzubilden. Die emulierten Geräte können dabei reale Netzwerkprotokolle und Anwendungen ausführen. Die Virtualisierung kann dabei als technische Grundlage für eine Netzwerkemulation dienen. Dieser Zusammenhang ist sowohl bei Kathara als auch bei Containerlab erkennbar. Kathara verwendet Container, um Netzwerkgeräte zu emulieren und miteinander zu verbinden. Containerlab ermöglicht ebenfalls den Aufbau containerbasierter Netzwerklabore, bei denen die einzelnen Knoten über virtuelle Verbindungen zu einer Topologie verbunden werden.

2.1.2 Docker

Docker ist eine Plattform zur Erstellung und Ausführung von Containern. Anwendungen können zusammen mit den benötigten Abhängigkeiten in einem Container bereitgestellt und isoliert ausgeführt werden[1].

Docker stellt außerdem Netzwerkfunktionen bereit, mit denen Container über virtuelle Netzwerke miteinander verbunden werden können. Damit bildet Docker eine technische Grundlage für Kathara und Containerlab, indem die Netzwerkknoten als Container ausgeführt werden. Der Aufbau und die Organisation der jeweiligen Netzwerktopologie werden von Kathara bzw. Containerlab übernommen.

2.1.3 Container

Ein Container ist eine isolierte Umgebung, in der eine Anwendung zusammen mit den dafür benötigten Dateien und Abhängigkeiten ausgeführt werden kann. Jeder Container kann eigene Prozesse, Dateien und Netzwerkeinstellungen besitzen.

Im Gegensatz zu einer virtuellen Maschine benötigt ein Container kein eigenes vollständiges Betriebssystem. Mehrere Container auf einem Host teilen sich den Kernel des Host-Betriebssystems. Dadurch benötigen Container in der Regel weniger Ressourcen und können schneller gestartet werden als virtuelle Maschinen[2].

2.1.4 Image

Ein Docker-Image bildet die Grundlage für einen Container. Es enthält die Programme, Dateien, Bibliotheken und Konfigurationen, die für die Ausführung des Containers erforderlich sind. Aus demselben Image können mehrere voneinander getrennte Container erstellt werden [3].

Images können selbst erstellt oder aus einer Container-Registry wie Docker Hub heruntergeladen werden. Docker Hub ist der Standard-Marktplatz zum Speichern und Verteilen von Images [4].

2.1.5 YAML-Datei

YAML(ein rekursives Akronym für „YAML Ain’t Markup Language“) ist eine menschenfreundlich ausgerichtete Daten-Serialisierungssprache, die sich gut mit modernen Programmiersprachen für gemeinsame Alltagsaufgaben versteht[5].

```
1 # Products purchased
2 - item : Super Hoop
3   quantity: 1
4 - item : Basketball
5   quantity: 4
6 - item : Big Shoes
7   quantity: 1
```

Dieses Beispiel zeigt eine kurze Beschreibung der Syntax.

In Containerlab wird eine YAML-Datei zur Beschreibung der Netzwerktopologie verwendet. In dieser Datei werden beispielsweise die Knoten des Labs, die verwendeten Images und die Verbindungen zwischen den Knoten festgelegt[6].

2.1.6 Startup-Datei

In Kathara kann für jeden Knoten eine Startup-Datei angelegt werden. Diese Datei enthält Befehle, die bei Start des jeweiligen Knotens automatisch ausgeführt werden. Die Startup-Datei wird nach dem Namen des Knotens benannt, beispielsweise `router1.startup` für den Knoten `router1`.

Mit einer Startup-Datei können wiederkehrende Konfigurationsschritte automatisiert werden. So können beispielsweise IP-Adressen und Routen konfiguriert oder benötigte Dienste gestartet werden. Dadurch müssen diese Befehle nach einem erneuten Start des Labs nicht jedes Mal manuell eingegeben werden[7].

2.1.7 Dynamic Host Configuration Protocol (DHCP)

Das Dynamic Host Configuration Protocol (DHCP) ist ein Client-Server-Protokoll, mit dessen Hilfe Netzwerkgeräte automatisch Netzwerkkonfigurationen erhalten können. Ein DHCP-Server kann einem Client beispielsweise eine IP-Adresse und weitere Konfigurationsparameter bereitstellen[8].

2.1.8 DHCP Option 121 (RFC 3442)

Die DHCP Option 121 wird als *Classless Static Route Option* bezeichnet. Sie ermöglicht es einem DHCP-Server, einem Client zusätzlich zur normalen Netzwerkkonfiguration statische Routen zu übermitteln. Über DHCP Option 121 kann dem Client mitgeteilt werden, über welchen Router ein bestimmtes Zielnetz erreicht werden kann [9]. Die entsprechende Route kann automatisch in die Routingtabelle des Clients übernommen werden. Eine manuelle Konfiguration der Route auf jedem einzelnen Client ist dadurch nicht erforderlich.

2.1.9 Apache HTTP Server

Der Apache HTTP Server ist ein kostenloser Open-Source-Webserver, der zahlreiche Websites und Webanwendungen unterstützt. Er verarbeitet Anfragen von Clients, beispielsweise Webbrowsern, und stellt die angeforderten Inhalte wie HTML-Seiten, Bilder oder andere Dateien bereit.

Apache kann durch verschiedene Module um zusätzliche Funktionen erweitert werden. In dieser Arbeit wurde Apache innerhalb eines Containers eingesetzt, um einen Webserver innerhalb der virtuellen Netzwerktopologie bereitzustellen.

2.1.10 WebDAV

WebDAV steht für *Web Distributed Authoring and Versioning* und ist eine Erweiterung des HTTP-Protokolls. Es ermöglicht, Dateien oder komplette Verzeichnisse über das Internet bereitzustellen und an verschiedene Geräte zu übertragen. Es ergänzt das Hypertext Transfer Protocol (HTTP) und unterstützt auch dessen Nachfolgeversionen HTTP/2 und HTTP/3, die primär für das Ausliefern und Anzeigen von Webinhalten zuständig sind[10].

Durch WebDAV können Dateien auf einem Webserver über das Netzwerk verwaltet werden. Dazu gehören beispielsweise das Speichern, Bearbeiten Löschen von Dateien.

2.2 Containerlab

2.2.1 Architektur und Funktionsweise

Containerlab ist eine Umgebung zum Aufbau von Netzwerklaboren, die auf Container basiert sind. Die Netzwerktopologie wird in einer **YAML-Datei** konfiguriert. In dieser Datei wird festgelegt, welche Geräte im Netzwerk vorhanden sind, welche Images sie verwenden und wie die Geräte miteinander verbunden sind [11].

Die einzelnen Geräte werden als Container gestartet und über virtuelle Netzwerkschnittstellen miteinander verbunden. Containerlab übernimmt dabei die Erstellung und Verwaltung der benötigten Container und Netzwerkverbindungen. Ein definiertes Lab kann mit dem Befehl `containerlab deploy -t <Name des definierten Labs>` gestartet und mit dem Befehl `containerlab destroy -t <Name des definierten Labs>` wieder entfernt werden.

Die Konfiguration der Geräte hängt vom verwendeten Node-Kind ab. Dadurch können sowohl normale Linux-Container als auch verschiedene Netzwerkbetriebssysteme wie Arista cEOS, Cisco IOL und Nokia SR Linux in einer Topologie verwendet werden [12].

2.2.2 Unterstützte Kinds-Arten

Containerlab unterstützt verschiedene Arten von Kinds. Mit dem Parameter `kind` kann festgelegt werden, welche Art von Geräten verwendet wird. Dabei können zum Beispiel Linux-Container, verschiedene Netzwerkbetriebssysteme wie Arista cEOS, Cisco IOL und Nokia SR Linux eingesetzt werden.

In dieser Arbeit werden hauptsächlich die Node-Kinds `linux` und `arista_ceos` verwendet. Die Linux-Container werden als Endgeräte eingesetzt, um zum Beispiel die Kommunikation zwischen den Geräten zu testen. Arista cEOS wird für die Router- und Switching-Funktionen verwendet.

2.3 Kathara

2.3.1 Architektur und Funktionsweise

Kathara ist ein Netzwerkemulator, mit dem Netzwerke virtuell aufgebaut und getestet werden können. Die einzelnen Geräte, wie PCs oder Router, werden dabei in Containern ausgeführt. Für die Verwaltung der Container kann Kathara Docker verwenden[13].

Der Aufbau eines Netzwerks wird hauptsächlich in der Datei `lab.conf` festgelegt. Dort werden die Geräte und ihre Schnittstellen definiert. Die Verbindung zwischen den Geräten erfolgt über die sogenannten **Kollisionsdomäne**. Geräte, die mit derselben Kollisionsdomäne verbunden sind, befinden sich im gleichen virtuellen Netzwerksegment.

Für jedes Gerät kann zusätzlich eine `startup`-Datei erstellt werden. Darin stehen Befehle, die beim Start automatisch ausgeführt werden. So können zum Beispiel Routen und IP-Adressen eingerichtet werden [7].

Mit dem Befehl `kathara lstart` wird das Lab gestartet. Kathara erstellt dann die benötigten Container und die in der `lab.conf` festgelegten Verbindungen. Mit `kathara lclean` kann das Lab anschließend wieder beendet werden.

2.3.2 Unterstützte Gerätetypen

Kathara verwendet keine fest vorgegebenen Gerätetypen wie physische Netzwerkgeräte. Die Funktion eines virtuellen Geräts hängt hauptsächlich von dem verwendeten Docker-Image und der darauf ausgeführten Software ab. Dadurch können verschiedene Netzwerkgeräte wie Endgeräte, Router oder Webserver nachgebildet werden.

Kathara stellt dafür unterschiedliche Docker-Images zur Verfügung [14].

Das Image `kathara/base` enthält eine grundlegende Linux-Umgebung und kann beispielsweise für PCs oder Webserver verwendet werden. Für Routing-Aufgaben steht das Image `kathara/frr` zur Verfügung, das FRRouting enthält. Damit kann ein Container als Router eingesetzt und für verschiedene Routing-Protokolle konfiguriert werden.

Mithilfe der Auswahl der Images und der jeweiligen Konfiguration können die virtuellen Geräte an das gewünschte Netzwerkszenario angepasst werden.

Kapitel 3: Konzeption / Architektur

3.1 Beschreibung der drei Szenarien

Für den Vergleich der Netzwerkvirtualisierungsumgebung wurden drei Testszenarien mit unterschiedlichem Funktionsumfang konzipiert. Dabei werden sowohl grundlegende Netzwerkfunktionen als auch komplexere Mechanismen und Anwendungen berücksichtigt.

Szenario 1: Grundlegende Netzwerkkommunikation und Anwendungen

Das erste Szenario untersucht die grundlegende Kommunikation zwischen mehreren Endgeräten und Netzwerkkomponenten. Zusätzlich werden Linux-basierte Anwendungen und Netzwerkdienste integriert, um zu prüfen, ob neben der Netzwerkkommunikation auch konkrete Anwendungen innerhalb der virtuellen Umgebung ausgeführt werden können.

Szenario 2: VLANs, Routing und Netzwerkdienste

Im zweiten Szenario wird das Netzwerk durch VLANs logisch segmentiert. Die Kommunikation zwischen den verschiedenen Teilnetzen erfolgt über Inter-VLAN-Routing. Zusätzlich wird OSPF für den dynamischen Austausch von Routinginformationen eingesetzt. Damit werden sowohl Layer-2- als auch Layer-3-Funktionen untersucht.

Szenario 3: Redundanz, Zugriffskontrolle und Multicast

Das dritte Szenario erweitert die Topologie um die redundante Übertragungswege, Zugriffskontrollen und Multicast-Kommunikation. Durch den Einsatz mehrerer Router kann im Falle des Ausfalls eines Übertragungsweges ein alternativer Pfad verwendet werden. Zusätzlich wird untersucht, ob bestimmte Kommunikationsbeziehungen gezielt eingeschränkt und Multicast-Daten zwischen den Netzwerken übertragen werden können.

3.2 Auswahl der Netzwerkgeräte

Für die Umsetzung der Testszenarien wurden Netzwerkkomponenten ausgewählt, die benötigten Routing-, Switching- und Anwendungsfunktionen unterstützen. Gleichzeitig wurde darauf geachtet, dass die verwendeten Images für die praktische Umsetzung verfügbar sind.

3.2.1 Containerlab

Für die Router und Switches wurden in Containerlab Arista cEOS-Image verwendet. Ursprünglich wurden virtuelle Cisco-Geräte wie Cisco IOL und IOL-L2 geplant. Diese Images sind zwar in der kostenlosen Version von Cisco Modeling Labs (CML-Free) verfügbar, ihre Nutzung ist jedoch auf Cisco Modeling Labs beschränkt. Eine Verwendung der bereitgestellten Cisco-Images außerhalb von CML ist ohne entsprechende Lizenz nicht gestattet. Zudem können mit CML-Free maximal fünf Nodes gleichzeitig betrieben werden [15]. Arista stellte daher eine geeignete Alternative für die virtuelle Testumgebung dar.

Das für Containerlab benötigte cEOS-Image kann über die Arista-Webseite bezogen werden. Die Beschaffung des Images erfolgt in folgenden Schritten:

1. Zunächst ist eine Registrierung bzw. Anmeldung auf der Arista-Webseite erforderlich [16].
2. Nach der Anmeldung wird im Software-Downloadbereich die Produktkategorie *cEOS Lab* ausgewählt.
3. Anschließend wird die gewünschte cEOS-Version ausgewählt und die entsprechende Archiv heruntergeladen. Für diese Arbeit wurde die Version 4.32.0F verwendet. Die heruntergeladene Datei lautet:

```
cEOS64-lab-4.32.0F.tar.xz ceos:4.32.0F
```

Für die Endgeräte und Server wurden Linux-Container eingesetzt. Diese ermöglichten die Ausführung verschiedener Anwendungen und Netzwerktools, beispielsweise Apache, iPerf3 und Wireshark.

3.2.2 Kathara

In Kathara wird das Image `kathara/frr` für den Router verwendet. Dieses Image enthält das FRRouting [17]. FRRouting stellt verschiedene dynamische Routingprotokolle, darunter RIP, OSPF und BGP, zur Verfügung [18].

3.2.3 Physischen Labor

Im Physischen Labor werden die vorhandene Cisco-Netzwerkgeräte eingesetzt. Dazu gehören ein Cisco ISR 4221 als Router und ein Cisco Catalyst 2960 als Switch.

Kapitel 4: Umsetzung im physischen Cisco-Labor

Dieses Kapitel beschreibt die vollständige Umsetzung von den drei Szenarien im physischen Cisco-Labor. Das physische Labor dient als Referenzumgebung für den Vergleich mit den virtuellen Simulationsumgebungen. Im Gegensatz zu den virtuellen Testumgebung wie Containerlab und Kathara kommt hier reale Netzwerkhardware zum Einsatz.

4.1 Vorbereitung der Cisco Labor Testumgebung

Vor der Umsetzung der drei Szenarien wurde zunächst die physische Testumgebung im Cisco-Labor der Hochschule vorbereitet. Die benötigten Router, Switches und Computer standen im Labor bereits zur Verfügung. Die für die Konfiguration und die grundlegende Überprüfung benötigten Netzwerktools waren auf den Laborcomputern bereits vorhanden. Lediglich `iperf3` wurde zusätzlich installiert, um den Datendurchsatz zwischen den Endgeräten messen zu können.

Anders als bei den virtuellen Testumgebungen Containerlab und Kathara wurden die Geräte im physischen Labor mit echten Ethernet-Kabeln miteinander verbunden. Die PCs wurden mit den jeweiligen Switches verbunden und die Switches anschließend an die zugehörigen Router angeschlossen. Auch die Verbindung zwischen den Router wurde über ein Ethernet-Kabel hergestellt.

Für den Zugriff auf die Kommandozeile der Cisco-Geräte wurde PuTTY verwendet. Nach der Verkabelung wurde überprüft, ob die benötigten Schnittstellen aktiv waren. Außerdem wurden Firewall-Einstellungen der Computer angepasst, da die Windows- Firewall den Zugriff auf lokale Dienste blockiert. Durch die Definition der Netzwerkschnittstelle als privates Netz, kann die Blockade umgangen werden. Anschließend konnten die einzelnen Szenarien konfiguriert und getestet werden.

4.2 Umsetzung Szenario 1

4.2.1 Verwendete Hardware und Software

Die Tabelle 4.1 zeigt die im Labor verwendete Hardware und deren jeweilige Aufgabe.

Tabelle 4.1: Verwendete Hardware und Software im physischen Cisco-Labor

Gerät/Software	Modell/Version	Rolle
Router	Cisco ISR 4221	Routing zwischen den Netzwerken
Switch	Cisco Catalyst 2960	Verbindung der Endgeräte
Computer	Windows-PC	Durchführung der Netzwerktests
PuTTY	Release 0.83	Konfiguration der Cisco-Geräte

4.2.2 Topologie

Die Topologie besteht aus zwei lokalen Netzwerken, die über zwei Cisco-Router miteinander verbunden sind. In jedem lokalen Netzwerk befinden sich ein Switch und zwei PCs als Endgeräte.

Router1 ist über die Schnittstelle `GigabitEthernet0/0/0` mit dem ersten lokalen Netzwerk verbunden und verwendet dort die IP-Adresse `10.1.0.1`. Router2 stellt mit der Adresse `10.2.0.1` die Verbindung zum zweiten lokalen Netzwerk her. Die direkte Verbindung zwischen den beiden Routern erfolgt über das Netz `192.168.100.0`. Dabei wurde der Interface `GigabitEthernet0/0/1` von Router1 die Adresse `192.168.100.1` und der Interface `GigabitEthernet0/0/1` von Router2 die Adresse `192.168.100.2` zugewiesen.

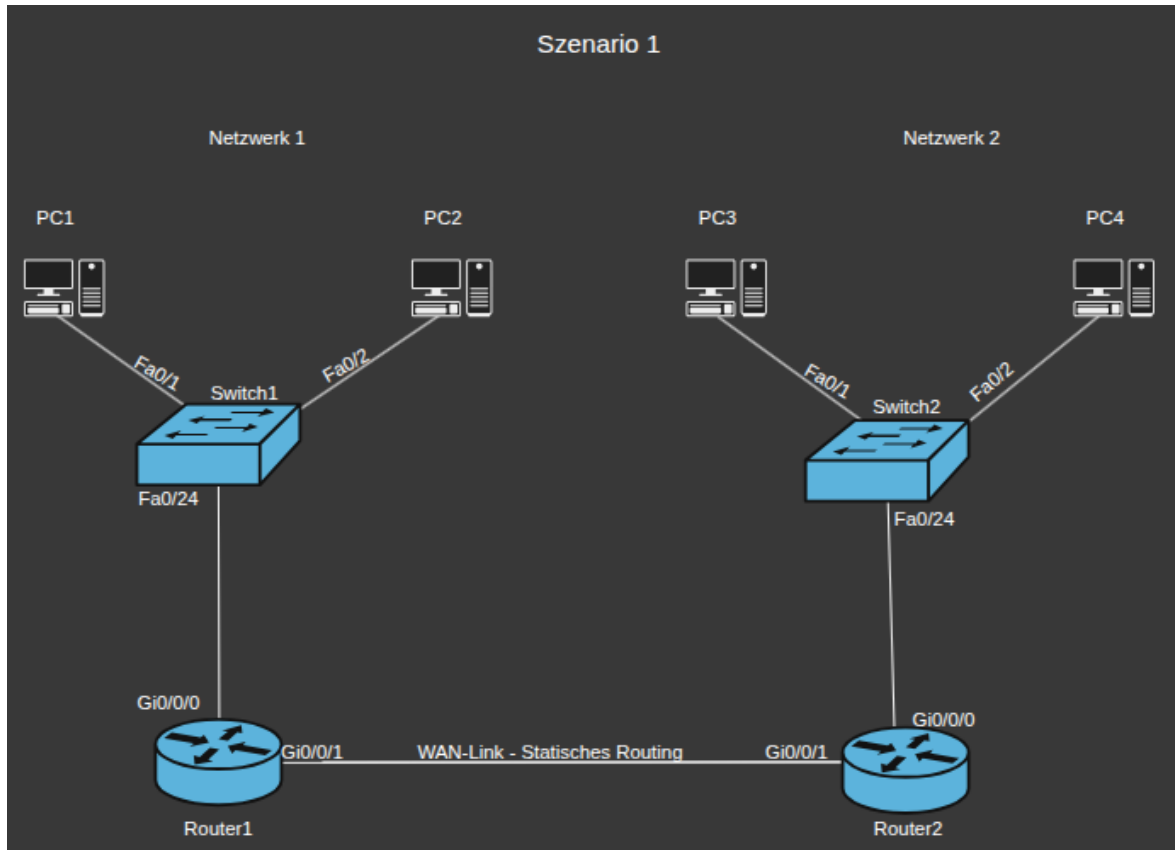


Abbildung 4.1: Netzwerktopologie des ersten Szenarios im Labor

Die verwendete Adressierung ist in Tabelle 4.2 zusammengefasst.

Tabelle 4.2: Adressierung der Geräte im physischen Labor für Szenario 1

Gerät	Interface	IP-Adresse	Funktion
Router1	Gi0/0/0	10.1.0.1	Gateway für Netzwerk 1
Router1	Gi0/0/1	192.168.100.1	Verbindung zu Router2
Router2	Gi0/0/0	10.2.0.1	Gateway für Netzwerk 2
Router2	Gi0/0/1	192.168.100.2	Verbindung zu Router1
PC1 und PC2	Ethernet	10.1.0.50-10.1.0.100(Per DHCP)	Clients für Netzwerk 1
PC3 und PC4	Ethernet	10.2.0.50-10.2.0.100(Per DHCP)	Clients für Netzwerk 2

4.2.3 Konfiguration

4.2.4 Routerkonfiguration

Nach dem Aufbau der Topologie wurden zunächst die Schnittstellen der beiden Router konfiguriert. Die Konfiguration erfolgte direkt über die Cisco-Kommandozeile. Router1

erhielt im ersten lokalen Netzwerk die Adresse 10.1.0.1, während Router2 im zweiten Netzwerk die Adresse 10.2.0.1 verwendet. Für die direkte Verbindung zwischen den Routern wurden die Adressen 192.168.100.1 und 192.168.100.2 vergeben.

Codeauszug 4.1: Interface-Konfiguration der Router in Szenario 1

```
1  ! Router1
2  interface GigabitEthernet0/0/0
3      ip address 10.1.0.1 255.255.255.0
4      no shutdown
5
6  interface GigabitEthernet0/0/1
7      ip address 192.168.100.1 255.255.255.252
8      no shutdown
9
10 !Router2
11 interface GigabitEthernet0/0/0
12     ip address 10.2.0.1 255.255.255.0
13     no shutdown
14
15 interface GigabitEthernet0/0/1
16     ip address 192.168.100.2 255.255.255.252
17     no shutdown
```

Damit die beiden lokalen Netzwerke miteinander kommunizieren können, wurde auf jedem Router eine statische Route zum jeweils entfernten Netzwerk eingerichtet. Router1 erreicht das Netz 10.2.0.0/24 über Router2 mit der Next-Hop-Adresse 192.168.100.2. Auf Router2 wurde entsprechend eine Route zum Netz 10.1.0.0/24 über 192.168.100.1 eingerichtet.

Codeauszug 4.2: Konfiguration der statischen Routen

```
1  !Router1
2  ip route 10.2.0.0 255.255.255.0 192.168.100.2
3
4  !Router2
5  ip route 10.1.0.0 255.255.255.0 192.168.100.1
```

Außerdem wurde auf jeweiligen Router ein DHCP-Server für die angeschlossenen Endgeräte eingerichtet. Router1 übernimmt die Adressvergabe im Netz 10.1.0.0/24 und Router2 im Netz 10.2.0.0/24.

Codeauszug 4.3: DHCP-Konfiguration auf der Router

```
1  !Router1
2  ip dhcp excluded-address 10.1.0.1 10.1.0.49
3  ip dhcp pool NETZWERK1
4      network 10.1.0.0 255.255.255.0
5      dns-server 8.8.8.8
```

```

6  option 121 hex 080a.0a01.0001
7  lease 210
8
9  !Router2
10 ip dhcp excluded-address 10.1.0.1 10.1.0.49
11 ip dhcp pool NETZWERK2
12 network 10.2.0.0 255.255.255.0
13 dns-server 8.8.8.8
14 option 121 hex 080a.0a02.0001
15 lease 210

```

Der Befehl `option 121 hex 080a.0a02.0001` wird verwendet, um den Clients eine statistische Route zum Netz 10.0.0.0/8 über das Gateway 10.2.0.1 zu übermitteln. Der Hexadezimalzahl setzt sich dabei aus drei Bestandteilen zusammen. Der Wert 08 gibt die Länge des Zielnetzes an und entspricht /8. Dann folgt 0a, die hexadezimale Darstellung der Dezimalzahl 10. Damit wird das Zielnetz 10.0.0.0/8 beschrieben. Die anschließenden vier Bytes 0a 02 00 01 stellen die Gateway-Adresse dar und entsprechen dezimal der IP-Adresse 10.2.0.1. Der Client kann dadurch Datenverkehr für das entsprechende Zielnetz über dieses Gateway weiterleiten[9].

Tabelle 4.3: Aufbau des Hexadezimalwertes der DHCP-Option 121

Hexadezimalwert	Bedeutung	Wert
08	Länge	/8
0a	Zielnetz	10.0.0.0/8
0a 02 00 01	Gateway	10.2.0.1

4.2.5 Switch-Konfiguration

Die beiden Cisco Catalyst 2960 wurden in Szenario 1 als Layer-2-Switch eingesetzt. Sie verbinden die PCs des jeweiligen lokalen Netzwerks mit dem zugehörigen Router. Eine besondere Konfiguration der Switches war für dieses Szenario nicht erforderlich.

4.2.6 Durchführung und Testergebnisse

DHCP-Adressvergabe

Nach der Konfiguration wurde geprüft, ob die Endgeräte erfolgreich eine IP-Adresse über DHCP erhalten haben. Für PC1 und PC2 wurden Adressen aus dem DHCP-Bereich des ersten LANs erwartet. Für PC3 und PC4 wurden Adressen aus dem DHCP-Bereich des zweiten LANs erwartet. Die Vergebenen IP-Adressen sind in Tabelle 4.4 dargestellt.

Tabelle 4.4: Vergebene IP-Adressen in Szenario 1

Gerät	IP-Adresse	Netzwerk
PC1	10.1.0.54	10.1.0.0/24
PC2	10.1.0.55	10.1.0.0/24
PC3	10.2.0.50	10.2.0.0/24
PC4	10.2.0.51	10.2.0.0/24

Ergebnis: Alle vier PCs erhielten automatisch eine IP-Adresse. Der DHCP-Server im Router funktioniert korrekt.

Ping-Test

Mit den Ping-Tests wurde die Erreichbarkeit innerhalb der Topologie überprüft. Von PC1 und PC2 wurden zunächst Router1 und Router2 und anschließend die Endgeräte PC3 und PC4 getestet.

Tabelle 4.5: Ergebnisse der Ping-Tests von PC1

Ziel	IP-Adresse	Paketverlust	Antwortzeit
Router1	10.1.0.1	0 %	ca. 1 ms
Router2	10.2.0.1	0 %	ca. 1 ms
PC3	10.2.0.50	0 %	ca. 2,33 ms
PC4	10.2.0.51	0 %	ca. 2,33 ms

Ergebnis: 0 % Paketverlust in allen Testfällen. Die gleichen Tests wurden zusätzlich von PC3 und PC4 durchgeführt. Auch hier waren Router1, Router2 sowie PC1 und PC2 ohne Paketverlust erreichbar.

Traceroute

Nach der erfolgreichen Überprüfung der Konnektivität wurde mit `tracert` überprüft, welchen Weg die Pakete zwischen den beiden Netzwerken nehmen. Der Test wurde von PC1 zum Endgerät PC4 mit der Adresse 10.2.0.51 durchgeführt.

Tabelle 4.6:

Hop	IP-Adresse	Gerät
1	10.1.0.1	Router1
2	192.168.100.2	Router2
3	10.2.0.51	PC3

Ergebnis: 3 Hops - der erwartete Netzwerkpfad wurde bestätigt.

Datendurchsatz-iperf3-Test

Abschließend wurde der TCP-Datendurchsatz zwischen den beiden Netzwerken mit `iperf3` gemessen. PC3 wurde dabei als Server und PC1 als Client verwendet. Die Messung wurde über einen Zeitraum von 30 Sekunden durchgeführt.

Codeauszug 4.4: Durchsatzmessung zwischen PC1 und PC3

```
1 PC3:
2 iperf3 -s
3
4 PC1:
5 iperf3.exe -c 10.2.0.50 -t 30
6
7 Ergebnis:
8 Transfer: 119 MBytes
9 Sender: 33.2 Mbit/s
10 Receiver: 33.2 Mbit/s
```

Während der 30-sekündigen Messung wurden insgesamt 119 MByte übertragen. Dabei wurde sowohl auf der Sender- als auch auf der Empfängerseite ein durchschnittlicher Datendurchsatz von 33,2 Mbit/s erzielt.

4.3 Umsetzung Szenario 2

4.3.1 Topologie

Die Topologie besteht aus zwei Routern, einem Switch, vier Windows-Computer und einem Linux-Laptop, der als Webserver verwendet wurde.

Am Switch wurden zwei VLANs eingerichtet. Die Dozenten-PCs befinden sich in VLAN 10 und die Studenten-PCs in VLAN 20. Der Switch ist über eine Trunk-Verbindung mit Router1 verbunden.

Router1 übernimmt das Routing zwischen den beiden VLANs und ist mit Router2 über das Netz 10.0.1.0/30 verbunden. An Router2 ist der Webserver angeschlossen. Dieser befindet sich im Netzwerk 10.0.30.0/24.

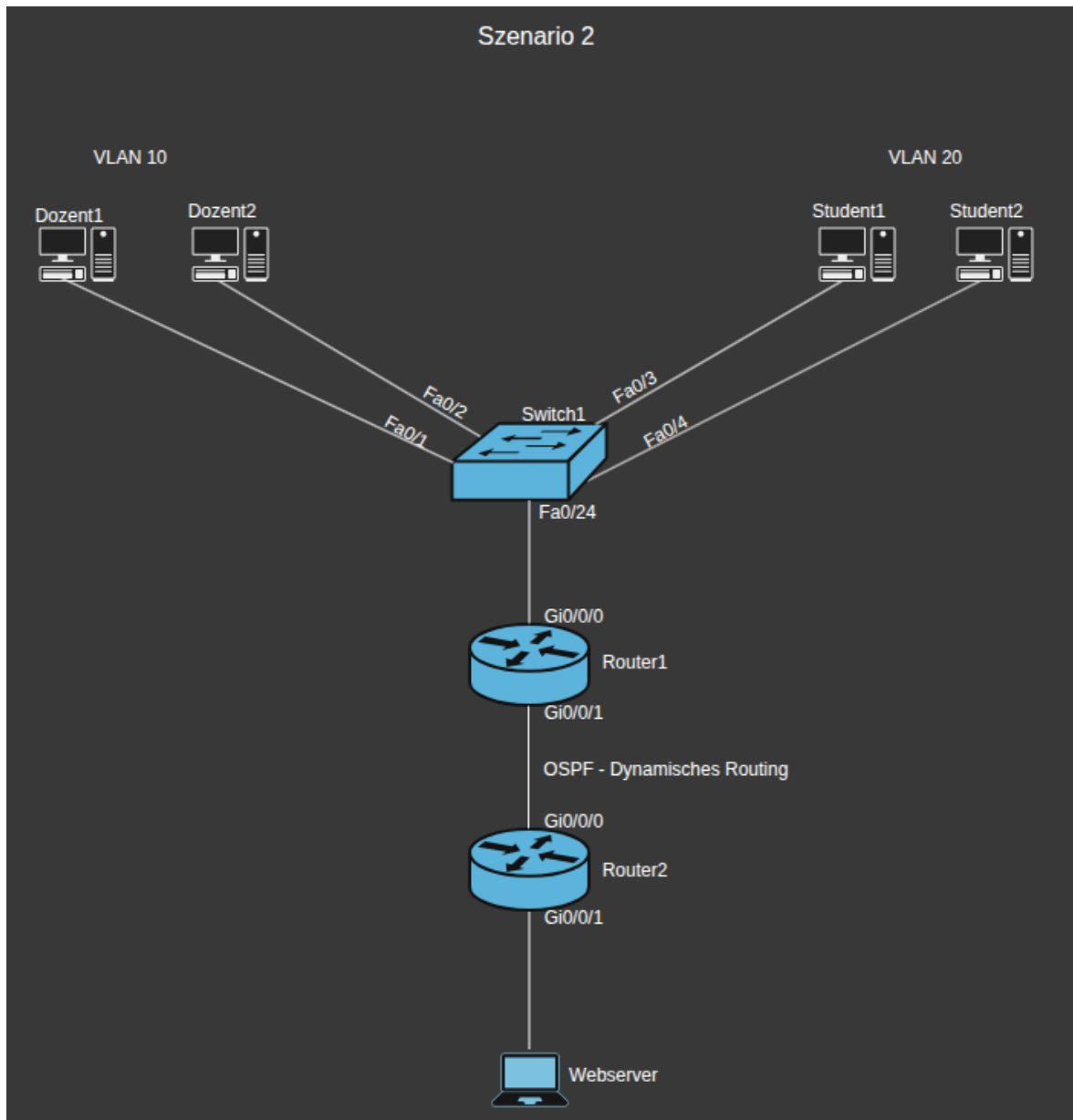


Abbildung 4.2: Netzwerktopologie des zweiten Szenarios im Labor

Die verwendete Adressierung ist in Tabelle 4.7 zusammengefasst.

Tabelle 4.7: Adressierung des zweiten Containerlab-Szenarios

Geräte	IP-Adresse	Beschreibung
Router1 (VLAN 10)	10.0.10.1/24	Gateway für VLAN 10 (Dozenten)
Router1 (VLAN 20)	10.0.20.1/24	Gateway für VLAN 20 (Studenten)
Router1 (WAN)	10.0.1.1/30	Verbindung zu Router2
Router2 (WAN)	10.0.1.2/30	Verbindung zu Router1
Router2 (Servernetz)	10.0.30.1/24	Gateway des Webservernetzes
DHCP-Bereich VLAN 10	10.0.10.10 – 10.0.10.50	Automatische Adressvergabe für die Dozenten-PCs
DHCP-Bereich VLAN 20	10.0.20.10 – 10.0.20.50	Automatische Adressvergabe für die Studenten-PCs
WebDAV-Server	10.0.30.2/24	Zentraler Dateiserver im Servernetz

4.3.2 Konfiguration

Routerkonfiguration

Auf Router1 wurden zwei Subinterfaces auf `GigabitEthernet0/0/0` eingerichtet.

Das Subinterface `GigabitEthernet0/0/0.10` gehört zu VLAN 10 und verwendet die IP-Adresse `10.0.10.1`.

Das Subinterface `GigabitEthernet0/0/0.20` gehört zu VLAN 20 und verwendet die IP-Adresse `10.0.20.1`. Router1 übernimmt dadurch das Routing zwischen den beiden VLANs. Für den Austausch der Routen zwischen den beiden Routern wurde OSPF verwendet.

Codeauszug 4.5: Konfiguration auf Router1

```

1
2 interface GigabitEthernet0/0/0
3   no shutdown
4
5 interface GigabitEthernet0/0/0.10
6   encapsulation dot1Q 10
7   ip address 10.0.10.1 255.255.255.0
8   no shutdown
9
10 interface GigabitEthernet0/0/0.20
11   encapsulation dot1Q 20
12   ip address 10.0.20.1 255.255.255.0
13   no shutdown

```

```
14
15 ! WAN-Interface zu Router2
16 interface GigabitEthernet0/0/1
17   ip address 10.0.1.1 255.255.255.252
18   no shutdown
19
20 router ospf 1
21   network 10.0.10.0 0.0.0.255 area 0
22   network 10.0.20.0 0.0.0.255 area 0
23   network 10.0.1.0 0.0.0.3 area 0
```

Zusätzlich wurde auf Router1 einen DHCP-Server für beide VLANs konfiguriert. Dadurch erhalten die angeschlossenen PCs ihre IP-Adressen automatisch.

Codeauszug 4.6: DHCP-Konfiguration auf Router1

```
1 ip dhcp excluded-address 10.0.10.1 10.0.10.9
2 ip dhcp pool VLAN10-Dozenten
3   network 10.0.10.0 255.255.255.0
4   option 121 hex 080a.0000.0a00.1001
5   dns-server 8.8.8.8
6   lease 210
7
8 ip dhcp excluded-address 10.0.20.1 10.0.20.9
9 ip dhcp pool VLAN20-Studenten
10  network 10.0.20.0 255.255.255.0
11  option 121 hex 080a.0000.0a00.1401
12  dns-server 8.8.8.8
13  lease 210
```

Router1 und Router2 wurden über das Netz 10.0.1.0/30 miteinander verbunden. Router1 verwendet dabei die Adresse 10.0.1.1 und Router2 die Adresse 10.0.1.2

Auf beiden Routern wurde anschließend OSPF eingerichtet. Dadurch werden die VLAN-Netze sowie das Netz des Webserver zwischen den beiden Routern bekannt gemacht. Der Webserver befindet sich hinter Router2 im Netz 10.0.30.0/24.

Codeauszug 4.7: Konfiguration auf Router2

```
1 ! WAN Interface zu Router1
2 interface GigabitEthernet0/0/0
3   ip address 10.0.1.2 255.255.255.252
4   no shutdown
5
6 ! Interface zu Webserver
7 interface GigabitEthernet0/0/1
8   ip address 10.0.30.1 255.255.255.0
9   no shutdown
```

```
10
11 router ospf 1
12   network 10.0.1.0 0.0.0.3 area 0
13   network 10.0.30.0 0.0.0.255 area 0
```

VLAN-Konfiguration des Switches

Auf Switch1 wurden VLAN 10 für die Dozenten und VLAN 20 für die Studenten eingerichtet. Die Ports `FastEthernet0/1` und `FastEthernet0/2` wurden VLAN 10 zugeordnet. Die Ports `FastEthernet0/3` und `FastEthernet0/4` wurden VLAN 20 zugeordnet. Der Port `FastEthernet0/24` wurde als Trunk konfiguriert und mit Router1 verbunden. Über diesen Trunk Port werden beide VLANs zwischen dem Switch und Router1 übertragen.

Codeauszug 4.8: VLAN-Konfiguration auf Switch1

```
1  vlan 10
2    name Dozenten
3
4  vlan 20
5    name Studenten
6
7  ! Access Ports fuer Dozenten (VLAN 10)
8  interface FastEthernet0/1
9    switchport access vlan 10
10   switchport mode access
11
12 interface FastEthernet0/2
13   switchport access vlan 10
14   switchport mode access
15
16
17 ! Access Ports fuer Studenten (VLAN 20)
18 interface FastEthernet0/3
19   switchport access vlan 20
20   switchport mode access
21
22 interface FastEthernet0/4
23   switchport access vlan 20
24   switchport mode access
25
26 ! Trunk Port zu Router1
27 interface FastEthernet0/24
28   switchport trunk allowed vlan 10,20
29   switchport mode trunk
```

Konfiguration des Webservers

Der Webserver wurde auf einem Laptop mit Ubuntu eingerichtet. Zuerst wurde die Netzwerkschnittstelle des Laptops konfiguriert.

Der Webserver erhielt die IP-Adresse 10.0.30.2/24. Als Gateway wurde der Router2 mit der IP-Adresse 10.0.30.1 verwendet. Zusätzlich wurde eine Route zu den anderen Netzen des Szenarios eingerichtet.

Codeauszug 4.9: Netzwerkkonfiguration des physischen Webservers

```
1 sudo ip addr add 10.0.30.2/24 dev enp0s31f6
2 sudo ip route add 10.0.0.0/8 via 10.0.30.1 dev enp0s31f6
```

Nach der Netzwerkkonfiguration wurde zunächst die Erreichbarkeit des Gateways und der anderen Netze mit `ping` überprüft. Dadurch konnte sichergestellt werden, dass die grundlegende IP-Kommunikation zwischen dem Webserver und den angeschlossenen Netzen funktioniert, bevor der WebDAV-Dienst konfiguriert wurde.

Für die Bereitstellung des WebDAV-Dienstes wurde auf einem Ubuntu-PC Apache in der Version 2.4.66(Ubuntu) verwendet, genau die gleiche Version wie im Kathara. Daher erfolgte die Konfiguration ähnlich wie bei der Umsetzung im Kathara. Zunächst wurde ein eigenes Verzeichnis für die WebDAV-Dateien eingerichtet und dem Apache-Benutzer `www-data` zugeordnet. Anschließend wurden die benötigten WebDAV-Module aktiviert.

Codeauszug 4.10: Vorbereitung des WebDAV-Servers im physischen Labor

```
1
2 sudo apt-get install apache2
3
4 sudo mkdir -p /var/www/webdav
5 sudo chown -R www-data:www-data /var/www/webdav
6
7 sudo a2enmod dav
8 sudo a2enmod dav_fs
9
10 sudo a2dissite 000-default
11 sudo a2ensite webdav
12 sudo service apache2 start
```

Die Module `dav` und `dav_fs` stellen die benötigten WebDAV-Funktionalitäten bereit. Die standardmäßig aktive Apache-Site `000-default` von Apache wurde mit `sudo a2dissite 000-default` deaktiviert, damit für das Szenario nur der eingerichtete WebDAV-VirtualHost auf Port 80 verwendet wird. Anschließend wurde die eigene Konfiguration `webdav.conf` mit `sudo a2ensite webdav` aktiviert und der Apache gestartet.

Die eigentliche Konfiguration des WebDAV-Dienstes wurde in der Datei `/etc/apache2/sites-available/webdav.conf` vorgenommen.

Codeauszug 4.11: WebDAV-Konfiguration im physischen Labor

```
1
2 <VirtualHost *:80>
3     DocumentRoot /var/www/webdav
4     <Directory /var/www/webdav>
5         DAV On
6         <RequireAny>
7             Require method GET POST OPTIONS PROPFIND HEAD
8             Require ip 10.0.10.0/24
9         </RequireAny>
10    </Directory>
11 </VirtualHost>
```

Als `DocumentRoot` wurde das Verzeichnis `/var/www/webdav` verwendet. Mit `DAV On` wird WebDAV für dieses Verzeichnis aktiviert.

Die Zugriffsrechte wurden über `RequireAny` festgelegt. Die Methoden `GET`, `POST`, `OPTIONS`, `PROPFIND` und `HEAD` sind für alle Clients erlaubt.

Geräte aus dem Dozenten-Netz `10.0.10.0/24` erhalten zusätzlich Zugriff auf weitere WebDAV-Methoden. Dadurch können Dozenten hochladen, lesen und löschen, während Studenten die bereitgestellten Dateien nur lesen können.

4.3.3 Durchführung und Testergebnisse

Überprüfung der VLAN-Trennung

Nach der Konfiguration wurde zunächst auf den Switch überprüft, ob die VLANs richtig eingerichtet und die entsprechenden Ports den VLANs zugeordnet wurden. Dafür wurde der Befehl `show vlan brief` verwendet.

Anschließend wurde die Kommunikation zwischen den Geräten mit Ping-Tests überprüft. Dadurch konnte kontrolliert werden, ob die eingerichteten VLANs und die Netzwerkverbindungen wie vorgesehen funktionieren.

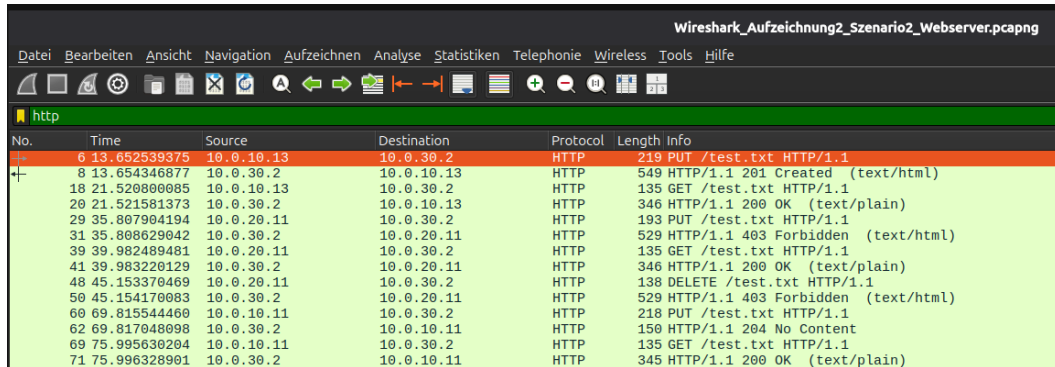
OSPF-Test

Nach der VLAN-Konfiguration wurde die Funktion von OSPF überprüft. Dazu wurde zunächst mit `show ip ospf neighbor` kontrolliert, ob die OSPF-Nachbarschaften zwischen den Routern erfolgreich aufgebaut wurden. Anschließend wurde mit `show ip route ospf` geprüft, ob die entfernten Netze über OSPF in die Routingtabellen übernommen wurden.

Zum Schluss wurde die Erreichbarkeit der verschiedenen Netze mit Ping-Tests überprüft.

WebDAV-Dienst

Zum Schluss wurde der WebDAV-Dienst getestet. Dabei wurde geprüft, ob die unterschiedlichen Zugriffsrechte für Dozenten und Studenten wie vorgesehen funktionieren. Die Kommunikation mit dem Webserver wurde gleichzeitig mit Wireshark aufgezeichnet.



Wireshark_Aufzeichnung2_Szenario2_Webserver.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
6	13.652539375	10.0.10.13	10.0.30.2	HTTP	219	PUT /test.txt HTTP/1.1
8	13.654346877	10.0.30.2	10.0.10.13	HTTP	549	HTTP/1.1 201 Created (text/html)
18	21.520800085	10.0.10.13	10.0.30.2	HTTP	135	GET /test.txt HTTP/1.1
20	21.521581373	10.0.30.2	10.0.10.13	HTTP	346	HTTP/1.1 200 OK (text/plain)
29	35.807904194	10.0.20.11	10.0.30.2	HTTP	193	PUT /test.txt HTTP/1.1
31	35.808629042	10.0.30.2	10.0.20.11	HTTP	529	HTTP/1.1 403 Forbidden (text/html)
39	39.982489481	10.0.20.11	10.0.30.2	HTTP	135	GET /test.txt HTTP/1.1
41	39.983220129	10.0.30.2	10.0.20.11	HTTP	346	HTTP/1.1 200 OK (text/plain)
48	45.153370469	10.0.20.11	10.0.30.2	HTTP	138	DELETE /test.txt HTTP/1.1
50	45.154170083	10.0.30.2	10.0.20.11	HTTP	529	HTTP/1.1 403 Forbidden (text/html)
60	69.815544460	10.0.10.11	10.0.30.2	HTTP	218	PUT /test.txt HTTP/1.1
62	69.817048098	10.0.30.2	10.0.10.11	HTTP	150	HTTP/1.1 204 No Content
69	75.995630204	10.0.10.11	10.0.30.2	HTTP	135	GET /test.txt HTTP/1.1
71	75.996328901	10.0.30.2	10.0.10.11	HTTP	345	HTTP/1.1 200 OK (text/plain)

Abbildung 4.3: Wireshark-Mitschnitt von HTTP-Methoden

Abbildung 4.3 zeigt die Kommunikation mit dem Webserver. Der Dozenten-PC mit der IP-Adresse 10.0.10.13 konnte die Datei `test.txt` hochladen. Der Server bestätigte dies mit 201 Created. Auch das Lesen der Datei mit GET war möglich.

Beim Studenten-PC mit der IP-Adresse 10.0.20.11 wurde das Hochladen und löschen einer Datei dagegen mit 403 Forbidden abgelehnt. Das Lesen der Datei war möglich und wurde mit 200 OK bestätigt. Damit wurden die festgelegten Zugriffsrechte erfolgreich umgesetzt.

4.3.4 Testergebnisse

Tabelle 4.8: Zusammenfassung der Testergebnisse des Szenario2

Test	Status	Ergebnis
VLAN-Zuordnung (show vlan)	✓	Alle Endgeräte wurden den vorgesehenen VLANs zugeordnet.
Kommunikation innerhalb VLAN 10	✓	Ping zwischen den Clients erfolgreich.
Kommunikation innerhalb VLAN 20	✓	Ping zwischen den Clients erfolgreich.
Inter-VLAN-Routing	✓	Kommunikation zwischen VLAN 10 und VLAN 20 über Router1 erfolgreich.
OSPF-Nachbarschaft	✓	Nachbarschaft im Zustand FULL ; Routinginformationen wurden erfolgreich ausgetauscht.
Erreichbarkeit des Webservers	✓	Webserver über Router2 erfolgreich erreichbar.
WebDAV-Zugriff (Dozent)	✓	Datei konnte erfolgreich hochgeladen, heruntergeladen und gelöscht werden 4.11.
WebDAV-Zugriff (Student)	✓	Herunterladen möglich; Schreib- und Löschzugriffe wurden mit 403 Forbidden verweigert.

4.4 Umsetzung Szenario 3

4.4.1 Topologie

Die Topologie besteht aus drei Cisco-Routern, zwei Layer-2-Switches und vier PCs. Router1 verbindet Netzwerk 1 (PC1, PC2) über Switch 1, Router2 verbindet Netzwerk 2 (PC3, PC4) über Switch2. Router1 und Router2 sind sowohl direkt verbunden (Primärpfad) als auch über Router3 (Backuppfad) miteinander verbunden. Dadurch steht bei einem Ausfall der direkten Verbindung ein alternativer Weg zwischen beiden Netzwerken zur Verfügung.

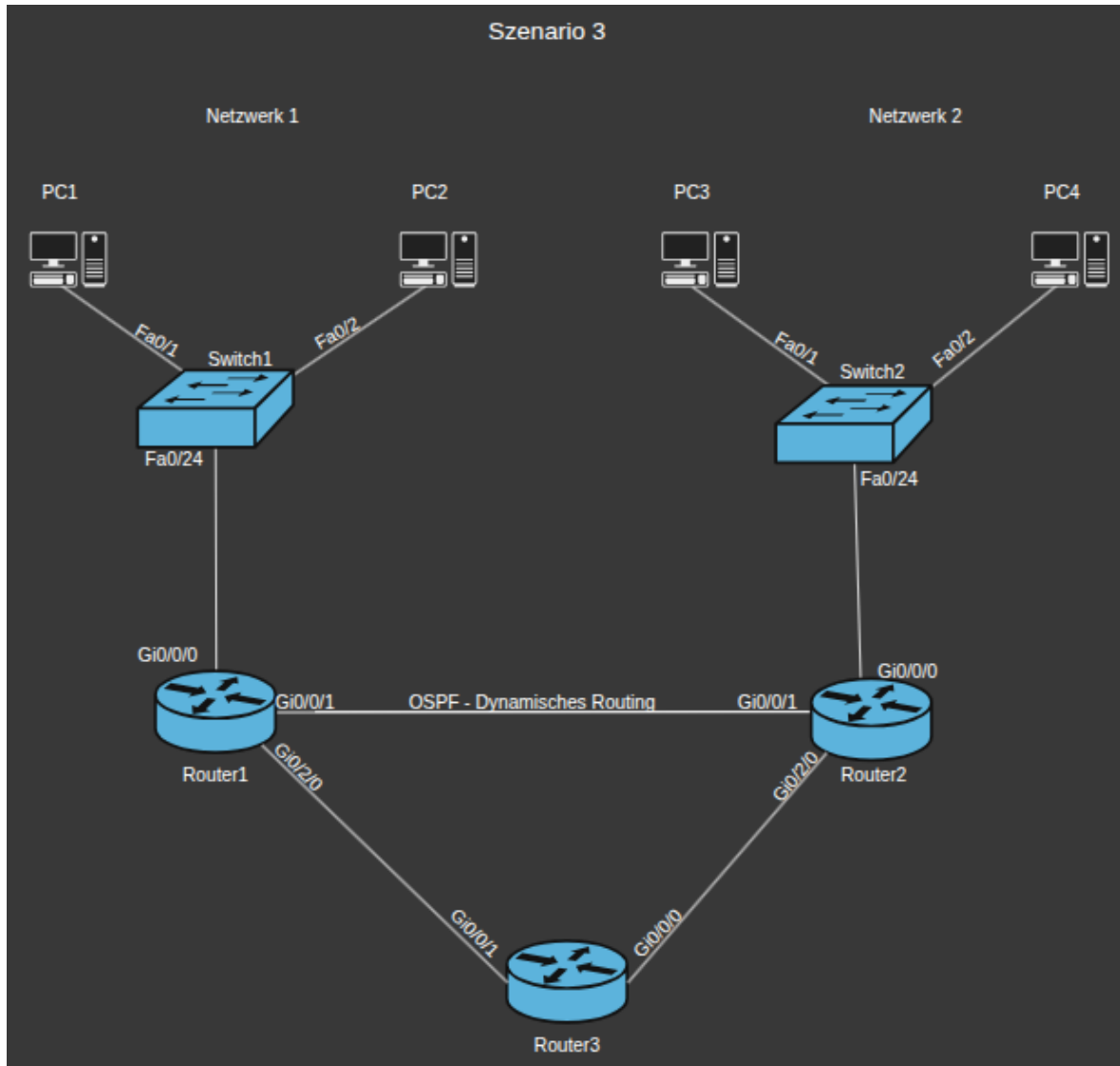


Abbildung 4.4: Netzwerktopologie des dritten Szenarios im Labor

Die verwendete Adressierung ist in Tabelle 4.9 zusammengefasst.

Tabelle 4.9: Adressierung des dritten Szenarios

Komponente	IP-Adresse/Netzwerk	Beschreibung
Router1 (LAN)	10.0.1.1/24	Standardgateway für Netzwerk 1 mit PC1 und PC2
Router1 (Primärpfad)	10.0.12.1/30	Verbindung zu Router2
Router2 (Primärpfad)	10.0.12.2/30	Verbindung zu Router1
Router1 (Backup-Pfad)	10.0.13.1/30	Verbindung zu Router3
Router3 (zu Router1)	10.0.13.2/30	Verbindung zu Router1
Router3 (zu Router2)	10.0.23.1/30	Verbindung zu Router2
Router2 (zu Router3)	10.0.23.2/30	Verbindung zu Router3
Router2 (LAN)	10.0.2.1/24	Gateway für Netzwerk 2 mit PC3 und PC4
DHCP-Bereich Netzwerk 1	10.0.1.20 – 10.0.1.100	Automatische IPv4-Adressvergabe für die Endgeräte im Netzwerk 1
PC2	10.0.1.11	Per DHCP-Reservierung fest zugewiesene IPv4-Adresse
DHCP-Bereich Netzwerk 2	10.0.2.20 – 10.0.2.100	Automatische IPv4-Adressvergabe für die Endgeräte im Netzwerk 2
PC4	10.0.2.11	Per DHCP-Reservierung fest zugewiesene IPv4-Adresse
Multicast-Bereich	224.0.0.0/4	Adressbereich für den Multicast-Verkehr

4.4.2 Konfiguration

Routerkonfiguration

Die Grundkonfiguration der Router entspricht weitgehend den vorherigen Szenarien. Dazu gehören die Konfiguration der Layer-3-Schnittstellen, sowie die Verwendung von OSPF für den dynamischen Austausch der Routinginformationen.

Codeauszug 4.12: OSPF-Konfiguration auf Router1

```

1 router ospf 1
2 network 10.0.1.0 0.0.0.255 area 0
3 network 10.0.12.0 0.0.0.3 area 0
4 network 10.0.13.0 0.0.0.3 area 0

```

Konfiguration der ACL-Regeln

Für die Zugriffskontrolle wurde auf Router1 und Router2 Extended Access Control Lists (ACL) eingerichtet. Dabei sollte nur die Kommunikation zwischen PC2 und PC4 blockiert

werden. Die Kommunikation zwischen den anderen PCs sollte weiterhin möglich sein.

Auf Router1 wurde folgende ACL konfiguriert:

Codeauszug 4.13: ACL-Konfiguration auf Router1

```
1 interface GigabitEthernet0/0/0
2   no ip unreachable
3   ip access-group FW_INBOUND_PC2_PC4 in
4
5 ip access-list extended FW_INBOUND_PC2_PC4
6   deny ip host 10.0.1.11 host 10.0.2.11
7   permit ip any any
```

Die ACL-Regel `ip access-list extended FW_INBOUND_PC2_PC4` blockiert Pakete von PC2 mit der IP-Adresse 10.0.1.11 zu PC4 mit der IP-Adresse 10.0.2.11. Mit `permit ip any any` wird der restliche IP-Verkehr weiterhin erlaubt.

Die ACL-Regel wird mit `ip access-group FW_INBOUND_PC2_PC4 in` eingehend auf dem Interface `interface GigabitEthernet0/0/0` angewendet. Mit `no ip unreachable` wurde die blockierte Verbindung mit 100 % Paketverlust angezeigt. Ohne diesen Befehl funktionierte die ACL-Regel zwar, beim Test wurde jedoch kein Paketverlust angezeigt.

Auf Router2 wurde das gleiche in der Gegenrichtung gemacht.

Codeauszug 4.14: ACL-Konfiguration auf Router2

```
1 interface GigabitEthernet0/0/0
2   no ip unreachable
3   ip access-group FW_INBOUND_PC2_PC4 in
4
5 ip access-list extended FW_INBOUND_PC2_PC4
6   deny ip host 10.0.2.11 host 10.0.1.11
7   permit ip any any
```

Multicast-Konfiguration

Für die Multicast-Kommunikation wurde auf den Routern zunächst mit `ip multicast-routing distributed` das Multicast-Routing aktiviert. Anschließend wurde mit `ip pim dense-mode` PIM Dense Mode auf den Interfaces eingerichtet, über die Multicast-Daten übertragen werden. Dabei geht ein Router zunächst davon aus, dass die Multicast-Daten auch in den anderen Bereichen des Netzwerks benötigt werden. Die Pakete werden deshalb zuerst über die vorhandenen Pfade weitergeleitet. Wenn ein Router auf einem Pfad keine Empfänger oder keine weitere PIM-Nachbarn für die Multicast-Gruppe hat, kann er eine Prune-Nachricht senden. Dadurch werden weitere Multicast-Pakete nicht mehr über diesen Zweig weitergeleitet.

Codeauszug 4.15: PIM-Konfiguration auf Router1

```
1 ip multicast-routing distributed
2
3 interface GigabitEthernet0/0/1
4   ip pim dense-mode
5
6 interface GigabitEthernet0/0/1
7   ip pim dense-mode
8
9 interface GigabitEthernet0/2/0
10  ip pim dense-mode
```

Auf den Layer-2-Switches wurde keine zusätzliche Konfiguration für IGMP Snooping vorgenommen. Denn auf die verwendeten Cisco-Switches war IGMP Snooping standardmäßig aktiviert. Dies wird von Cisco dokumentiert[19]. Eine globale Aktivierung von IGMP Snooping wäre auf einem Cisco-Switch mit dem Befehl `ip igmp snooping` möglich.

```
1 Switch2# configure terminal
2 Switch2(config)# ip igmp snooping
3 Switch2(config)# end
```

4.4.3 Durchführung

Überprüfung der Firewall-Regeln

Nach der Konfiguration wurde überprüft, ob die eingerichteten ACL-Regeln die vorgesehen funktionieren. Dazu wurden Ping-Tests zwischen den verschiedenen PCs durchgeführt.

Die Verbindung zwischen PC2 (10.0.1.11) und PC4 (10.0.2.11) wurde in beide Richtungen blockiert. Die Ping-Tests zeigten dabei einen Paketverlust von 100 %. Die Kommunikation zwischen den anderen PCs war weiterhin möglich. Damit wurde bestätigt, dass nur die gewünschte Verbindung durch die ACL-Regeln blockiert wurde.

Überprüfung des Failovers

Vor dem Ausfall war die direkte Verbindung zwischen Router1 und Router2 aktiv und wurde als Primärpfad verwendet. Der zusätzliche Router3 stellte den Backup-Pfad bereit. Die Kommunikation zwischen den beiden Netzwerken funktionierte zu diesem Zeitpunkt wie vorgesehen.

Für den Failover-test wurde das Kabel, das Router1 und Router2 direkt verbindet, entfernt. Nach dem Ausfall wurde der Verkehr automatisch über den Backup-Pfad Router1–Router3–Router2 weitergeleitet. Die Kommunikation zwischen den beiden Netzwerken blieb dadurch

weiterhin möglich. Damit konnte gezeigt werden, dass die konfigurierte OSPF-Redundanz wie vorgesehen funktioniert.

Überprüfung der Multicast-Kommunikation

Für den Multicast-Test wurde ein Raspberry Pi als Sender verwendet, der einen Video-Stream an die Multicast-Gruppe 239.1.2.3 sendete. Auf diesem Raspberry Pi wurde für den Multicast-Stream ein TTL-Wert von 10 eingestellt. Auf PC3 wurde VLC als Empfänger gestartet. Unter "Netzwerkstream öffnen" wurde die folgende Adresse eingegeben:

```
1 | udp://@239.1.2.3
```

Dadurch trat PC3 der Multicast-Gruppe bei und konnte den gesendeten Video-Stream empfangen.

Auf den Routern wurde die Multicast-Kommunikation anschließend mit `show ip igmp groups`, `show ip pim neighbor`, `show ip mroute` und `show ip mroute active` überprüft. Mit `show ip igmp groups` wurde überprüft, bei welchen Multicast-Gruppen die angeschlossenen Empfänger angemeldet sind. Mit `show ip pim neighbor` konnten die aufgebauten PIM-Nachbarschaften zwischen den Routern kontrolliert werden.

Die Multicast-Routingtabelle wurde mit `show ip mroute` überprüft. Dabei konnte unter anderem erkannt werden, über welche Interfaces die Multicast-Pakete weitergeleitet wurden. Der (S, G) -Eintrags zeigte außerdem die Quelle des Multicast-Streams und die zugehörige Multicast-Gruppe.

4.4.4 Testergebnisse

Tabelle 4.10: Testergebnisse von Szenario 3

Test	Status	Beobachtung
Netzwerkcommunication	✓	Kommunikation zwischen den beiden Netzen Über OSPF möglich.
Firewall-Filterung	✓	Kommunikation zwischen PC2 und PC4 erfolgreich blockiert.
OSPF-Failover	✓	Nach Ausfall des Primärpfades erfolgte die Umschaltung über Router3.
Firewall nach Failover	✓	Die Filterung zwischen PC2 und PC4 blieb auch auf dem Backup-Pfad wirksam.
Multicast-Routing	✓	Der (S,G)-Eintrag wurde aufgebaut und der Multicast-Verkehr weitergeleitet.
Multicast-Datenübertragung	✓	Die Multicast-Übertragung mit wurde erfolgreich nachgewiesen.

Kapitel 5: Umsetzung in Containerlab

5.1 Installation

5.1.1 Installation von Docker Engine

Containerlab basiert auf Docker und verwendet Docker-Container zur Virtualisierung der Netzwerkgeräte. Daher musste zunächst Docker Engine auf dem Ubuntu-20.04-LTS-System installiert werden. Für Ubuntu stehen verschiedene Installationsmethoden zur Verfügung. Docker kann beispielsweise über das offizielle APT-Repository, über heruntergeladene Pakete oder mithilfe des von Docker bereitgestellten Convenience-Skripts installiert werden [20]. Für diese Bachelorarbeit wurde das offizielle Convenience-Skript gewählt. Diese Methode wird von Docker insbesondere für Test- und Entwicklungsumgebung empfohlen und automatisiert die Installation aller benötigten Docker-Komponenten. Da die praktische Umsetzung dieser Arbeit ausschließlich in einer lokalen Testumgebung erfolgt, stellt diese Installationsmethode eine geeignete Lösung dar.

Codeauszug 5.1: Download des Docker-Installationsskripts

```
1 | curl -fsSL https://get.docker.com -o get-docker.sh
```

Mit diesem Befehl wird das offizielle Installationsskript von der Docker-Webseite heruntergeladen und lokal gespeichert.

Codeauszug 5.2: Ausführen des Docker-Installationsskripts

```
1 | sudo sh get-docker.sh
```

Das Installationsskript wird mit Administratorrechten ausgeführt und installiert die benötigten Docker-Komponenten.

Codeauszug 5.3: Überprüfung der Docker-Installation

```
1 | docker --version
```

Nach der Installation wurde mit dem Befehl `docker --version` überprüft, ob Docker Engine erfolgreich installiert wurde. Die Ausgabe bestätigt die installierte Docker-Version. Damit

stand Docker Engine als Container-Laufzeitumgebung für die Installation und Ausführung von Containerlab zur Verfügung.

5.1.2 Installation von Containerlab

In dieser Arbeit wurde das offizielle Installationsskript verwendet [21].

Codeauszug 5.4: Download und Installation von Containerlab

```
1 | bash -c "$(curl -sL https://get.containerlab.dev)"
```

Mit diesem Befehl wird das offizielle Installationsskript von der Webseite von Containerlab heruntergeladen und ausgeführt. Das Skript installiert Containerlab automatisch und richtet die erforderlichen Komponenten auf dem System ein.

Codeauszug 5.5: Überprüfung der Containerlab-Installation

```
1 | containerlab version
```

5.1.3 Installation des Arista-cEOS-Images

Im Gegensatz zu Docker Engine und Containerlab sind die cEOS-Images nicht öffentlich verfügbar und es muss zunächst ein Benutzerkonto über das Arista Customer Portal erstellt werden. Nach der Anmeldung wurde das zur verwendeten cEOS-Version passende `ceos64 tar`-Archiv heruntergeladen. Anschließend wurde das heruntergeladene Archiv in Docker importiert. Dadurch stehen die Images Containerlab für die Erstellung virtueller Netzwerkgeräte zur Verfügung [22].

Codeauszug 5.6: Import des Arista-cEOS-Images

```
1 | # import container image and save it under ceos:4.32.0F name
2 | docker import cEOS64-lab-4.32.0F.tar.xz ceos:4.32.0F
```

Codeauszug 5.7: Überprüfung des importierten Images

```
1 | docker images | grep ceos
```

Tabelle 5.1: Verwendete Software für die Containerlab-Testumgebung

Softwarekomponente	Version	Verwendungszweck
Ubuntu	26.04 LTS	Hostbetriebssystem für die Testumgebung
Docker Engine	29.5.3	Container-Laufzeitumgebung zur Ausführung der Netzwerkgeräte
Containerlab	0.76.1	Erstellung und Verwaltung der Netzwerktopologie
Arista cEOS	4.32.0F	Netzwerkbetriebssystem zur Simulation virtueller Router und Layer-2-Switches

5.2 Umsetzung Szenario 1

5.2.1 Topologie

Die Topologie besteht aus zwei lokalen Netzwerken, die jeweils über einen virtuellen Arista-cEOS-Switch mit zwei Linux-Containern verbunden sind. Die beiden Netzwerke werden über zwei virtuelle Router miteinander verbunden. Als Endgeräte wurden vier Linux-Container verwendet. Diese repräsentieren Arbeitsplatzrechner innerhalb der beiden lokalen Netzwerke. Die Switching-Funktion wird durch zwei virtuelle Arista-cEOS-Switches übernommen, während zwei weitere Arista-cEOS-Geräte als Router eingesetzt werden und den Datenaustausch zwischen den beiden Netzwerken ermöglichen. Abbildung 5.1 zeigt den vollständigen Aufbau des ersten Netzwerkszenarios.

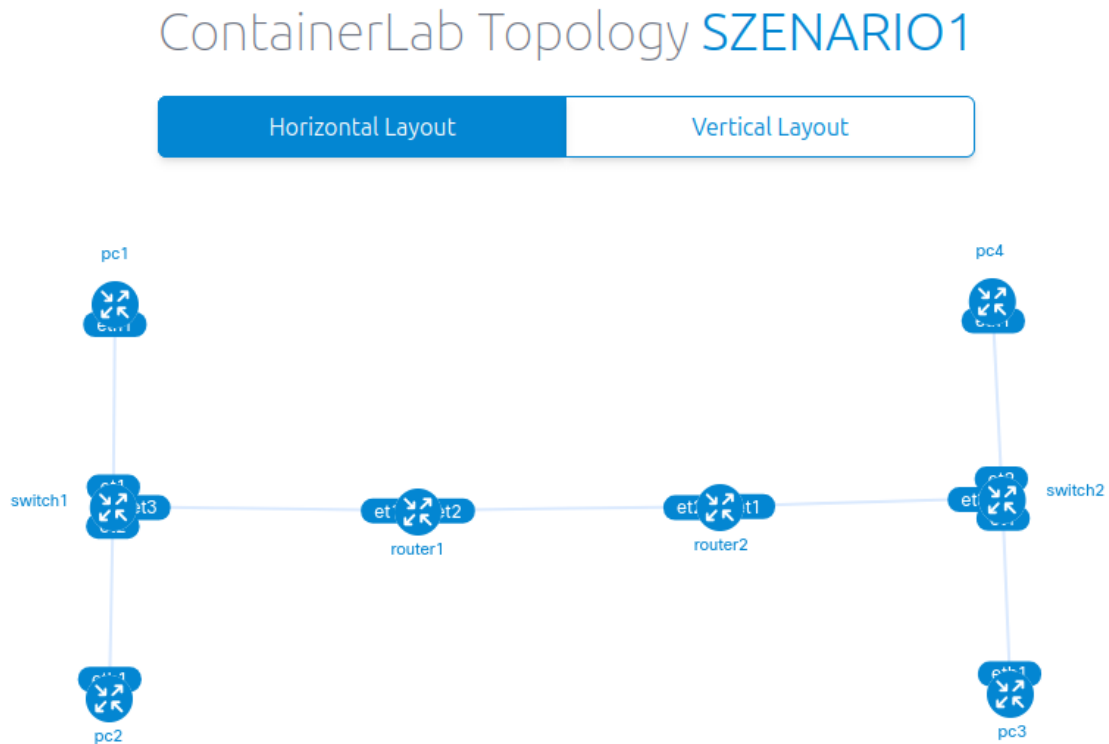


Abbildung 5.1: Netzwerktopologie des ersten Containerlab-Szenarios

Die verwendete Adressierung ist in Tabelle 5.2 zusammengefasst. Die Router verwenden statische IP-Adressen.

Tabelle 5.2: Adressierung der Geräte in Containerlab – Szenario 1

Gerät	IP-Adresse	Beschreibung
Router1 (LAN)	10.1.0.1/24	Gateway und DHCP-Server für LAN-1
Router1 (WAN)	192.168.100.1/30	Verbindung zu Router2
Router2 (LAN)	10.2.0.1/24	Gateway und DHCP-Server für LAN-2
Router2 (WAN)	192.168.100.2/30	Verbindung zu Router1
DHCP-Bereich	10.1.0.50 – 10.1.0.100	Automatische Adressvergabe für PC1 und PC2
DHCP-Bereich	10.2.0.50 – 10.2.0.100	Automatische Adressvergabe für PC3 und PC4

5.2.2 Konfiguration

Containerlab-Topologiedatei

Die Netzwerktopologie wurde in einer YAML-Datei (`topology.clab.yml`) definiert. Diese Datei bildet die zentrale Konfigurationsgrundlage von Containerlab und beschreibt den vollständigen Aufbau der virtuellen Netzwerkumgebung.

In der Topologiedatei werden zunächst der Name der Topologie sowie die Standardparameter festgelegt. Anschließend werden alle verwendeten Netzwerkgeräte mit ihrem jeweiligen Gerätetyp, dem zugehörigen Container-Image sowie den Startkonfigurationsdateien definiert. Darüber hinaus enthält diese YAML-Datei die Beschreibung aller virtuellen Verbindungen zwischen den einzelnen Geräten.

Die Router werden als Layer-3-Geräte eingesetzt. Sie besitzen statisch konfigurierte IPv4-Adressen, übernehmen das Routing zwischen den beiden lokalen Netzwerken und stellen für die angeschlossenen Endgeräte jeweils einen DHCP-Dienst bereit. Die Switches arbeiten auf Layer-2 und verbinden die Endgeräte mit jeweiligen Router. Die Linux-Container (Endgeräte) beziehen ihre IPv4-Adressen beim Start automatisch über DHCP.

Tabelle 5.3: Bestandteile der Containerlab-Topologie

Abschnitt	Beschreibung
<code>name</code>	Legt den Namen der Netzwerktopologie fest.
<code>defaults</code>	Definiert Standardparameter für alle Netzwerkgeräte.
<code>kinds</code>	Ordnet den verwendeten Gerätetypen das entsprechende Container-Image zu.
<code>nodes</code>	Beschreibt alle Router, Switches und Linux-Container der Topologie.
<code>links</code>	Definiert die virtuellen Verbindungen zwischen den einzelnen Netzwerkgeräten.

Tabelle 5.3 zeigt die wesentlichen Bestandteile der verwendeten Containerlab-Topologiedatei sowie deren Funktion innerhalb der Netzwerktopologie [6].

Codeauszug 5.8 zeigt den allgemeinen Aufbau der verwendeten Containerlab-Topologiedatei. Zunächst werden der Name der Topologie sowie allgemeine Standardeinstellungen definiert. Anschließend wird das für die virtuellen Router und Switches verwendete Arista-cEOS-Container-Image festgelegt.

Codeauszug 5.8: Grundstruktur der Containerlab-Topologiedatei

```

1 name: szenario1
2
3 topology:
```

```
4 defaults:
5   env:
6     INTFTYPE: et
7
8   kinds:
9     arista_ceos:
10      image: ceos:4.32.0F
```

Die Umgebungsvariable `INTFTYPE` legt fest, dass für Arista-cEOS die Kurzbeschreibung `et` für die Ethernet-Schnittstellen verwendet wird[23]. Im Abschnitt `kinds` wird die verwendete Version der Container-Image festgelegt, die für alle Arista cEOS Geräte eingesetzt wird.

Der Codeauszug 5.9 zeigt beispielhaft die Definition der Netzwerkgeräte innerhalb der Containerlab-Topologie.

Im ersten Szenario werden Linux-Container als Endgeräte eingesetzt.

Codeauszug 5.9: Beispiel der definition eines Gerätes in Containerlab

```
1 nodes:
2
3   pc1:
4     kind: linux
5     image: ghcr.io/srl-labs/network-multitool
6
7   switch1:
8     kind: arista_ceos
9     startup-config: switch1/switch1.cfg
10
11   router1:
12     kind: arista_ceos
13     startup-config: router1/router1.cfg
```

Im Abschnitt `nodes` werden sämtliche Netzwerkgeräte der Topologie definiert. Für die Endgeräte werden Linux-Container mit dem Image `network-multitool` verwendet [24]. Dieses Image stellt verschiedene Netzwerkwerkzeuge wie beispielsweise `ping`, `tracerroute` oder `tcpdump` zur Verfügung und eignet sich daher zur Durchführung der Netzwerktests[25]. Die Router und Layer-2-Switches basieren auf dem Gerätetyp `arista_ceos`. Über den Parameter `startup-config` wird jedem Gerät beim Start automatisch die entsprechende Konfigurationsdatei zugewiesen [26]. Dadurch stehen alle benötigten Einstellungen unmittelbar nach dem Start der Topologie zur Verfügung. Codeauszug 5.10 zeigt die Definition der virtuellen Netzwerkverbindungen innerhalb der Containerlab-Topologie. Über den Abschnitt `links` werden die einzelnen Netzwerkgeräte miteinander verbunden. Dabei wird jeweils angegeben, welche Schnittstellen zweier Geräte über virtuellen Link verbunden werden.

Codeauszug 5.10: Definition der Netzwerkverbindungen

```
1  links:
2    - endpoints: ["pc1:eth1", "switch1:et1"]
3    - endpoints: ["pc2:eth1", "switch1:et2"]
4    - endpoints: ["switch1:et3", "router1:et1"]
5    - endpoints: ["router1:et2", "router2:et2"]
6    - endpoints: ["router2:et1", "switch2:et3"]
7    - endpoints: ["switch2:et1", "pc3:eth1"]
8    - endpoints: ["switch2:et2", "pc4:eth1"]
```

Routerkonfiguration

Die Konfiguration der beiden Arista-cEOS-Router erfolgte über Startkonfigurationsdateien `router1.cfg` und `router2.cfg`, die beim Start der Container automatisch geladen werden. Ziel der Konfiguration war die Einrichtung der LAN- und WAN-Schnittstellen, die Aktivierung des Routings sowie die Bereitstellung eines DHCP-Dienstes für die Endgeräte.

Mit dem Befehl `ip routing` wird die IPv4-Paketweiterleitung aktiviert. Erst dadurch kann der Router Datenpakete zwischen den beiden lokalen Netzwerken weiterleiten. Die LAN-Schnittstelle wurde mit einer statischen IPv4-Adresse konfiguriert und dient als Standardgateway für die Endgeräte. Die WAN-Schnittstelle verbindet beide Router. Es wurden statische Routen eingerichtet, damit beide Router die jeweils entfernten Netzwerke erreichen können. Zusätzlich wurde auf jedem Router ein DHCP-Server eingerichtet. Dieser vergibt den Linux-Container automatisch IPv4-Adressen.

Switchkonfiguration

Für die beiden lokalen Netzwerke wurden zwei virtuelle Arista-cEOS-Switches eingesetzt. Die Switches wurden ausschließlich als Layer-2-Switches betrieben. Die Konfiguration beschränkte sich daher auf grundlegende Einstellungen wie die Vergabe eines Hostnamen, die Aktivierung der Schnittstellen.

Konfiguration der Endgeräte

Als Endgeräte wurden Linux-Container des Images `ghcr.io/srl-labs/network-multitool` eingesetzt. Dieses Image wurde gewählt, da es bereits zahlreiche Netzwerkzeuge wie `ping`, `traceroute`, `iperf3`, `tcpdump`, und `curl` enthält und somit eine geeignete Grundlage für die Durchführung der Netzwerktests bildet. Jeder Linux-Container wurde identisch konfiguriert. Die Konfiguration der Endgeräte erfolgte direkt in der Topologiedatei.

Codeauszug 5.11: Konfiguration von PC1

```
1  pc1:
2  kind: linux
3  image: ghcr.io/srl-labs/network-multitool
4  dns:
5    servers:
6      - 8.8.8.8
7  env:
8    DISPLAY: ${DISPLAY} # X11-Display fuer Wireshark GUI
9  binds:
10    - /tmp/.X11-unix:/tmp/.X11-unix # X11-Socket
11  exec:
12    - udhcpc -i eth1 # DHCP-Client auf eth1 starten
13    - sh -c "ip route add 10.0.0.0/8 via 10.1.0.1 dev eth1 || true"
14    - sh -c "apk add --no-cache wireshark ttf-dejavu &"
```

Im Abschnitt `exec` werden Befehle definiert, die beim Start des Containers automatisch ausgeführt werden. Zunächst wird mit `udhcpc` ein DHCP-Client gestartet, um automatisch eine IPv4-Adresse vom Router zu beziehen. Anschließend wird eine statistische Route zum Netzwerk `10.0.0.0/8` eingerichtet, damit Pakete über den Router und nicht über das Containerlab-Managementnetz geleitet werden. Abschließend werden Wireshark sowie die benötigten Schriftarten automatisch installiert.

Tabelle 5.4: Wesentliche Konfigurationsparameter der Linux-Endgeräte

Parameter	Funktion
<code>kind: linux</code>	Definiert das Gerät als Linux-Container und dient als virtuelles Endgerät innerhalb der Netzwerktopologie.
<code>image</code>	Legt das Docker-Image <code>ghcr.io/srl-labs/network-multitool</code> fest. Dieses Image enthält bereits verschiedene Netzwerktools wie <code>ping</code> , <code>traceroute</code> , <code>iperf3</code> , <code>tcpdump</code> und <code>curl</code> .
<code>dns</code>	Konfiguriert den öffentlichen DNS-Server <code>8.8.8.8</code> zur Namensauflösung innerhalb des Containers.
<code>DISPLAY</code>	Übergibt die Umgebungsvariable des Hostsystems, damit grafische Anwendungen wie Wireshark auf dem Host angezeigt werden können.
<code>bind</code>	Bindet den X11-Socket des Hostsystems in den Container ein und ermöglicht dadurch die Ausführung grafischer Anwendungen.
<code>udhcpc -i eth1</code>	Startet den DHCP-Client auf der Netzwerkschnittstelle <code>eth1</code> , sodass der Container seine IPv4-Adresse automatisch von Router erhält.
<code>ip route add</code>	Richtet eine statische Route für das Netzwerk <code>10.0.0.0/8</code> über den jeweiligen Router ein, damit Pakete korrekt zwischen den beiden LANs weitergeleitet werden.
<code>apk add</code>	Installiert Wireshark im Container sowie die benötigten Schriftarten automatisch beim Start des Containers.

5.2.3 Durchführung und Testergebnisse

Nach Abschluss der Konfiguration wurde zunächst in das Verzeichnis von Szenario1 gewechselt. Dort befinden sich die Containerlab-Topologiedatei sowie die Startkonfigurationen der Router und Switches. Mit dem Befehl `containerlab deploy` wurde das Lab gestartet [27]. Containerlab liest dabei die Datei `topology.clab.yml`, erstellt die benötigten Container und verbindet deren Schnittstellen entsprechend der definierten Topologie. Anschließend wurde mit `docker ps` überprüft, ob alle acht Container ausgeführt wurden.

Codeauszug 5.12: Das Starten des Labs

```

1 cd ~/Bachelorarbeit/Containerlab/Szenario1
2 sudo containerlab deploy -t topology.clab.yml
3 docker ps

```

Nach einem erfolgreichen Start sollte folgende Tabelle angezeigt werden.

A newer containerlab version (0.77.0) is available!
 Release notes: <https://containerlab.dev/rn/0.77/>
 Run 'clab version upgrade' or see <https://containerlab.dev/install/> for other installation options.

Name	Kind/Image	State	IPv4/6 Address
clab-szenario1-pc1	linux ghcr.io/srl-labs/network-multitool	running	172.20.20.5 3fff:172:20:20::5
clab-szenario1-pc2	linux ghcr.io/srl-labs/network-multitool	running	172.20.20.18 3fff:172:20:20::12
clab-szenario1-pc3	linux ghcr.io/srl-labs/network-multitool	running	172.20.20.4 3fff:172:20:20::4
clab-szenario1-pc4	linux ghcr.io/srl-labs/network-multitool	running	172.20.20.20 3fff:172:20:20::14
clab-szenario1-router1	arista_ceos ceos:4.32.0F	running	172.20.20.19 3fff:172:20:20::13
clab-szenario1-router2	arista_ceos ceos:4.32.0F	running	172.20.20.17 3fff:172:20:20::11
clab-szenario1-switch1	arista_ceos ceos:4.32.0F	running	172.20.20.6 3fff:172:20:20::6
clab-szenario1-switch2	arista_ceos ceos:4.32.0F	running	172.20.20.7 3fff:172:20:20::7

Abbildung 5.2: Erfolgreicher Start der Containerlab-Topologie

Die Angezeigten IP-Adressen sind nicht die eigenen konfigurierten IP-Adressen, sondern die Management IP-Adressen.

DHCP-Adressvergabe

Nach dem Start des Labs wurde zunächst überprüft, ob die Endgeräte erfolgreich eine IP-Adresse über DHCP erhalten haben. Für PC1 und PC2 wurden Adressen aus dem DHCP-Bereich des ersten LANs erwartet. Für PC3 und PC4 wurden Adressen aus dem DHCP-Bereich des zweiten LANs erwartet. Die aktuelle Adressierung eines Clients wurde kontrolliert.

Codeauszug 5.13: DHCP-Adressvergabe auf PC1

```

1 $ docker exec -it clab-szenario1-pc1 ip addr show eth1
2   inet 10.1.0.50/24 scope global eth1
3
4 $ docker exec -it clab-szenario1-router1 Cli -c "show dhcp server leases"
5   10.1.0.50
6   End: 2027/03/03 11:59:22 UTC
7   Last transaction: 2026/08/05 11:59:22 UTC
8   MAC address: aac1.ab83.d987
9
10  10.1.0.51
11  End: 2027/03/03 11:59:22 UTC
12  Last transaction: 2026/08/05 11:59:22 UTC

```

13 | MAC address: aac1.abb9.6fe8

Ergebnis: Alle vier PCs erhielten automatisch eine IP-Adresse. Der DHCP-Server im Router funktioniert korrekt mit einer Lease-Zeit von 210 Tagen.

Ping-Test

Die Konnektivität innerhalb und außerhalb der lokalen Netzwerke wurde überprüft.

Konnektivität außerhalb des lokalen Netzwerkes

Codeauszug 5.14: Ping-Test von PC1 zu PC4

```
1 | $ docker exec -it clab-szenario1-pc1 ping -c 4 10.2.0.51
2 | PING 10.2.0.51 (10.2.0.51) 56(84) bytes of data.
3 | 64 bytes from 10.2.0.51: icmp_seq=1 ttl=62 time=26.2 ms
4 | 64 bytes from 10.2.0.51: icmp_seq=2 ttl=62 time=1.53 ms
5 | 64 bytes from 10.2.0.51: icmp_seq=3 ttl=62 time=1.55 ms
6 | 64 bytes from 10.2.0.51: icmp_seq=4 ttl=62 time=1.40 ms
7 |
8 | --- 10.2.0.51 ping statistics ---
9 | 4 packets transmitted, 4 received, 0% packet loss, time 3003ms
10 | rtt min/avg/max/mdev = 1.398/7.681/26.249/10.719 ms
```

Ergebnis: 0 % Paketverlust in allen Testfällen.

Traceroute

Nach der erfolgreichen Überprüfung der Konnektivität wurde der Übertragungsweg zwischen den beiden lokalen Netzwerken untersucht. Das Werkzeug zeigt die Zwischenstationen an, die ein Paket auf dem Weg zum Ziel durchläuft.

Codeauszug 5.15: Traceroute von PC1 nach PC4

```
1 | $ docker exec -it clab-szenario1-pc1 traceroute 10.2.0.51
2 | (10.2.0.51), 30 hops max, 46 byte packets
3 | 1 10.1.0.1 (10.1.0.1) 0.874 ms 0.576 ms 0.460 ms
4 | 2 192.168.100.2 (192.168.100.2) 0.822 ms 1.046 ms 0.876 ms
5 | 3 10.2.0.51 (10.2.0.51) 2.964 ms 2.568 ms 2.253 ms
```

Ergebnis: 3 Hops - der erwartete Netzwerkpfad wurde bestätigt.

Datendurchsatz-iperf3-Test

Auf PC3 wurde zunächst der iperf3-Server gestartet. Anschließend wurde auf PC1 die Messung zur IP-Adresse von PC4 gestartet:

```
1 docker exec -it clab-szenario1-pc3 iperf3 -s
2 docker exec -it clab-szenario1-pc1 iperf3 -c <IP-Adresse-von-PC3> -t 30
3 docker exec -it clab-szenario1-pc1 iperf3 -c <IP-Adresse-von-PC3> -t 30 -M ↵
  1400
```

Ergebnisse des TCP-Durchsatz-Tests mit unterschiedlichen MSS-Werten

MSS-Wert	Sender (Mbit/s)	Empfänger (Mbit/s)	Retr	Bewertung
Standard (ohne -M)	0,093	0,000	61	MTU-/MSS-Problemen
1400 Bytes	33,0	32,1	0	Optimal

Ergebnisse: Der höchste TCP-Durchsatz wurde bei einer MSS von 1400 Bytes erreicht. Ohne Begrenzung der MSS traten 61 Neuübertragungen auf, wodurch der Datendurchsatz erheblich reduziert wurde. Deshalb wurde überprüft, ob die MTU-Einstellungen der Geräte der Grund von diesen Neuübertragungen sind[28].

Dabei wurde festgestellt, dass die Linux-Container eine MTU von 9500 Byte verwenden. Auf den cEOS-Routern war dagegen standardmäßig eine MTU von 1500 Byte eingestellt. Mit `tcpdump` wurde außerdem festgestellt, dass beim Aufbau der TCP-Verbindung eine MSS von 9460 Byte angekündigt wurde. Dadurch waren die Einstellungen der Endgeräte und der Router nicht aufeinander abgestimmt.

Um dies genau zu überprüfen, wurden die MTU der Endgeräte zunächst auf 1500 Byte gesetzt. Bei einem erneuten iPerf3-Test wurde ein Datendurchsatz von etwa 22,1 Mbits/s erreicht und es traten keine Neuübertragungen mehr auf.

Danach wurde die MTU der Router und Endgeräte auf 9214 Byte angepasst. Mit einem Ping-Test wurde zuerst überprüft, ob Pakete diese Größe übertragen werden können. Der anschließende iPerf3-Test erreichte etwa 141 Mbit/s beim Sender und 139 Mbit/s beim Empfänger. Auch hier traten keine TCP-Neuübertragungen auf.

Die Tests zeigen, dass die MTU-Einstellungen einen deutlichen Einfluss auf TCP-Datendurchsatz haben. Wenn die MTU-Werte im Übertragungspfad aufeinander abgestimmt sind, können Neuübertragungen vermeiden und höhere Datenraten erreicht werden.

Analyse des Netzwerkverkehrs mit Wireshark

Zur Analyse der übertragenen Datenpakete wurde Wireshark eingesetzt. Eine Möglichkeit zur Paketaufzeichnung besteht darin, Wireshark direkt innerhalb eines Containers der Containerlab Umgebung zu starten. Der Befehl `xhost +local:` erlaubt lokalen Anwendungen den Zugriff auf den X11-Server des Hostsystems. Dadurch können grafische Programme, die innerhalb eines Docker-Containers ausgeführt werden, ihre Benutzeroberfläche auf dem Ubuntu-Host anzeigen [29].

Mit dem Befehl `docker exec` wird Wireshark innerhalb des Containers `<Containername>` gestartet [30].

Codeauszug 5.16: Aufzeichnung des Netzwerkverkehrs auf dem Container PC1

```
1 xhost +local:
2 docker exec -it <Containername> wireshark &
3 #Wireshark auf PC1 starten
4 docker exec -it clab-szenario1-pc1 wireshark &
```

Nach dem Start kann in Wireshark die gewünschte Netzwerkschnittstelle, beispielsweise `eth1`, ausgewählt und der darüber laufende Netzwerkverkehr direkt aufgezeichnet werden. Während Wireshark gestartet war, wurde ein iPerf3, Ping-Test und Traceroute von PC1 zu PC4 durchgeführt.

No.	Time	Source	Destination	Protocol	Length	Info
94710	147.753680651	10.2.0.51	10.1.0.50	TCP	66	5201 → 48478 [ACK] Seq=6 Ack=182 Win=57344 Len=0 TSval=3683712287 TSecr=2602605697
94711	147.754484534	10.2.0.51	10.1.0.50	TCP	66	5201 → 48478 [ACK] Seq=6 Ack=511 Win=57344 Len=0 TSval=3683712288 TSecr=2602605697
94712	147.755357916	10.2.0.51	10.1.0.50	TCP	70	5201 → 48478 [PSH, ACK] Seq=6 Ack=511 Win=57344 Len=4 TSval=3683712288 TSecr=2602605697
94713	147.755791773	10.2.0.51	10.1.0.50	TCP	394	5201 → 48478 [PSH, ACK] Seq=10 Ack=511 Win=57344 Len=328 TSval=3683712288 TSecr=2602605697
94714	147.755836458	10.1.0.50	10.2.0.51	TCP	66	48478 → 5201 [ACK] Seq=511 Ack=338 Win=57344 Len=0 TSval=2602605704 TSecr=3683712288
94715	147.756126112	10.2.0.51	10.1.0.50	TCP	67	5201 → 48478 [PSH, ACK] Seq=338 Ack=511 Win=57344 Len=1 TSval=3683712288 TSecr=2602605697
94716	147.756238466	10.1.0.50	10.2.0.51	TCP	67	48478 → 5201 [PSH, ACK] Seq=511 Ack=339 Win=57344 Len=1 TSval=2602605704 TSecr=3683712288
94717	147.756259559	10.1.0.50	10.2.0.51	TCP	66	48478 → 5201 [FIN, ACK] Seq=512 Ack=339 Win=57344 Len=0 TSval=2602605704 TSecr=3683712288
94718	147.759731952	10.2.0.51	10.1.0.50	TCP	66	5201 → 48478 [ACK] Seq=340 Ack=513 Win=57344 Len=0 TSval=3683712293 TSecr=2602605704
94719	147.759765512	10.1.0.50	10.2.0.51	TCP	66	48478 → 5201 [ACK] Seq=513 Ack=340 Win=57344 Len=0 TSval=2602605708 TSecr=3683712293
94720	147.760119998	10.2.0.51	10.1.0.50	TCP	66	5201 → 48478 [ACK] Seq=340 Ack=513 Win=57344 Len=0 TSval=3683712293 TSecr=2602605704
94721	148.899822407	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94722	150.949484422	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94723	152.950823027	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94724	155.008456132	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94725	157.049481771	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94726	159.058653406	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94727	161.009605332	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94728	161.843267526	10.1.0.50	10.2.0.51	ICMP	98	Echo (ping) request id=0x2554, seq=1/256, ttl=64 (reply in 94729)
94729	161.845694344	10.2.0.51	10.1.0.50	ICMP	98	Echo (ping) reply id=0x2554, seq=1/256, ttl=62 (request in 94728)
94730	162.716607067	aa:cl:ab:02:5b:0e	LLDP Multicast	LLDP	205	HA/00:1c:73:51:3d:9c IN/Ethernet1 120 Sys=switch1 SysB=Arista Networks EOS version 4.32.0F-3
94731	162.844452024	10.1.0.50	10.2.0.51	ICMP	98	Echo (ping) request id=0x2554, seq=2/512, ttl=64 (reply in 94732)
94732	162.846676578	10.2.0.51	10.1.0.50	ICMP	98	Echo (ping) reply id=0x2554, seq=2/512, ttl=62 (request in 94731)
94733	163.149843862	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94734	163.845411261	10.1.0.50	10.2.0.51	ICMP	98	Echo (ping) request id=0x2554, seq=3/768, ttl=64 (reply in 94735)
94735	163.847510210	10.2.0.51	10.1.0.50	ICMP	98	Echo (ping) reply id=0x2554, seq=3/768, ttl=62 (request in 94734)
94736	164.846410538	10.1.0.50	10.2.0.51	ICMP	98	Echo (ping) request id=0x2554, seq=4/1024, ttl=64 (reply in 94737)
94737	164.848797742	10.2.0.51	10.1.0.50	ICMP	98	Echo (ping) reply id=0x2554, seq=4/1024, ttl=62 (request in 94736)
94738	165.208277893	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94739	167.249420701	aa:cl:ab:02:5b:0e	Spanning-tree (for- STP		119	MST. Root = 32768/0/00:1c:73:51:3d:9c Cost = 0 Port = 0x8001
94740	167.684989473	10.1.0.50	10.2.0.51	UDP	60	59147 → 33435 Len=18
94741	167.685677447	10.1.0.1	10.1.0.50	ICMP	88	Time-to-live exceeded (Time to live exceeded in transit)
94742	167.713565178	10.1.0.50	10.2.0.51	UDP	60	59147 → 33436 Len=18
94743	167.714127973	10.1.0.1	10.1.0.50	ICMP	88	Time-to-live exceeded (Time to live exceeded in transit)
94744	167.714238164	10.1.0.50	10.2.0.51	UDP	60	59147 → 33437 Len=18
94745	167.714671254	10.1.0.1	10.1.0.50	ICMP	88	Time-to-live exceeded (Time to live exceeded in transit)
94746	167.714711370	10.1.0.50	10.2.0.51	UDP	60	59147 → 33438 Len=18
94747	167.714819183	192.168.100.2	10.1.0.50	ICMP	88	Time-to-live exceeded (Time to live exceeded in transit)
94748	167.740844594	10.1.0.50	10.2.0.51	UDP	60	59147 → 33439 Len=18
94749	167.741924646	192.168.100.2	10.1.0.50	ICMP	88	Time-to-live exceeded (Time to live exceeded in transit)
94750	167.742836077	10.1.0.50	10.2.0.51	UDP	60	59147 → 33440 Len=18
94751	167.743035458	192.168.100.2	10.1.0.50	ICMP	88	Time-to-live exceeded (Time to live exceeded in transit)
94752	167.743096884	10.1.0.50	10.2.0.51	UDP	60	59147 → 33441 Len=18
94753	167.745437776	10.2.0.51	10.1.0.50	ICMP	88	Destination unreachable (Port unreachable)
94754	167.770267630	10.1.0.50	10.2.0.51	UDP	60	59147 → 33442 Len=18
94755	167.772498069	10.2.0.51	10.1.0.50	ICMP	88	Destination unreachable (Port unreachable)
94756	167.772624041	10.1.0.50	10.2.0.51	UDP	60	59147 → 33443 Len=18
94757	167.774231106	10.2.0.51	10.1.0.50	ICMP	88	Destination unreachable (Port unreachable)

Abbildung 5.3: Aufzeichnung des Netzwerkverkehrs mit Wireshark

Ergebnis: Wireshark bestätigt korrektes TTL-Verhalten an allen Punkten. Identifizierte Protokolle: ICMP, UDP, TCP.

Eine weitere Möglichkeit Pakete aufzuzeichnen und die virtuellen Schnittstellen zwischen den Containern zu überwachen ist es, Wireshark direkt auf dem Host-System zu starten. Dabei sollte der Datenverkehr an ausgewählten virtuellen Schnittstellen der Netzwerkknoten aufgezeichnet und direkt in Wireshark dargestellt werden[31].

Bei Containerlab wurde hierfür zwei Möglichkeiten getestet. Die erste Möglichkeit besteht darin, über den von Containerlab bereitgestellten benannten Network Namespace auf die jeweiligen Knoten zuzugreifen. Die vorhandenen Network Namespaces können zunächst mit folgenden Befehl angezeigt werden.

```
1 | sudo ip netns list
```

Anschließend kann beispielsweise der Datenverkehr an der Schnittstelle eth1 des Knotens pc1 aufgezeichnet werden:

```
1 | sudo "ip netns exec clab-szenario2-pc1 tcpdump -U -nni eth1 -w -" | ↔  
    wireshark -k -i -
```

Dabei wird `tcpdump` im Network Namespace des ausgewählten Knotens ausgeführt. Die Option `eth1` legt die zu überwachende Schnittstelle fest. Durch `-w -` wird die Aufzeichnung über die Standardausgabe ausgegeben. Mit der Option `-i -` liest Wireshark die von `tcpdump` übertragenen Paketdaten über die Standardeingabe ein. `-k` sorgt dafür, dass Wireshark die Aufzeichnung bzw. Das Einlesen direkt startet, sobald wireshark geöffnet wird. Als zweite Möglichkeit wurde bei Containerlab der Zugriff über den Docker-Container getestet:

```
1 | docker exec clab-szenario2-pc1 tcpdump -U -nni eth1 -w - | wireshark -k -i -
```

Eine direkte Auswahl der eigenen konfigurierten Schnittstelle der Container über die grafische Oberfläche von Wireshark auf dem Host-System war dagegen nicht möglich. Dort waren lediglich die Management-Schnittstellen von den laufenden Containern in Containerlab sichtbar.

5.3 Umsetzung Szenario 2

5.3.1 Topologie

Das Szenario besteht aus einem virtuellen Arista-cEOS-Switch, zwei virtuellen Arista-cEOS-Router, vier Linux-Container als Endgeräte sowie einem WebDAV-Server.

Die Endgeräte sind in zwei VLANs unterteilt. Die Dozenten-PCs gehören zum VLAN 10 und die Studenten-PCs gehören zum VLAN 20. Beide VLANs werden über einen Tunk-Port

mit Router1 verbunden. Router1 übernimmt das Inter-VLAN-Routing sowie die DHCP-Adressvergabe für beide VLANs. Über Router2 besteht die Verbindung zum Webservernetz, in dem der WebDAV-Server betrieben wird. Abbildung 5.4 zeigt den vollständigen Aufbau des zweiten Netzwerkszenarios

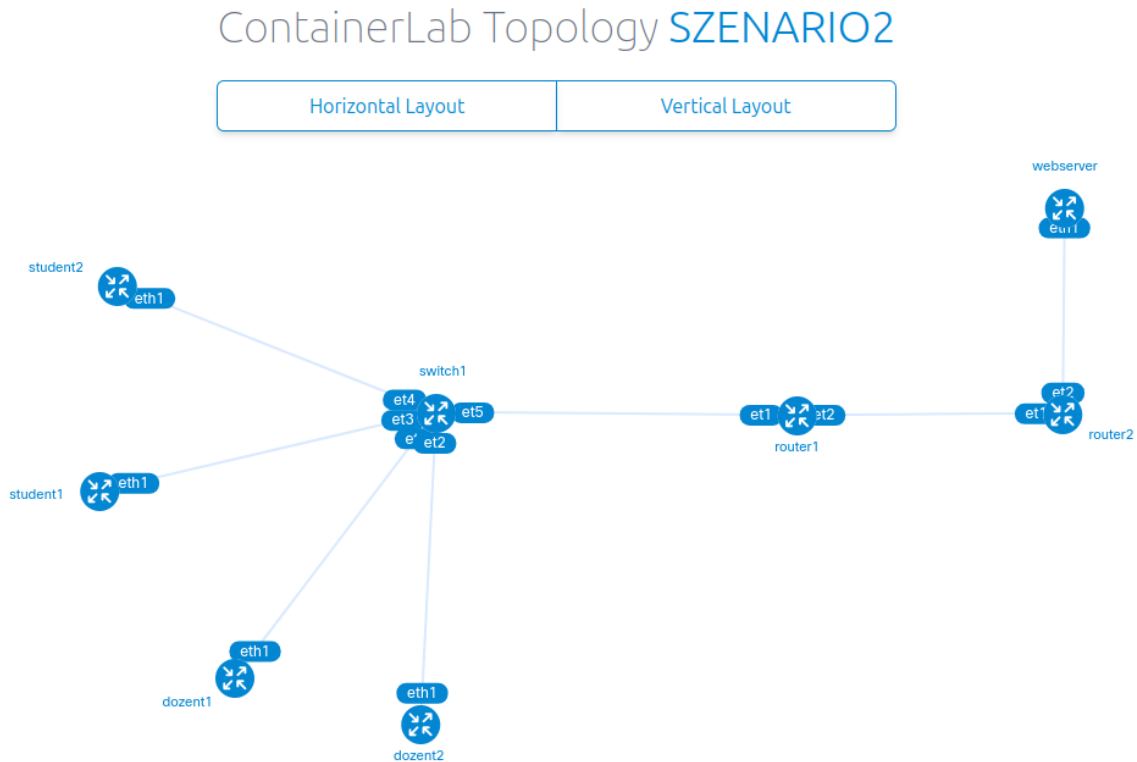


Abbildung 5.4: Netzwerktopologie des zweiten Containerlab-Szenarios

Tabelle 5.5: Adressierung des zweiten Containerlab-Szenarios

Geräte	IP-Adresse	Beschreibung
Router1 (VLAN 10)	10.0.10.1/24	Gateway für VLAN 10 (Dozenten)
Router1 (VLAN 20)	10.0.20.1/24	Gateway für VLAN 20 (Studenten)
Router1 (WAN)	10.0.1.1/30	Verbindung zu Router2
Router2 (WAN)	10.0.1.2/30	Verbindung zu Router1
Router2 (Servernetz)	10.0.30.1/24	Gateway des Webservernetzes
DHCP-Bereich VLAN 10	10.0.10.10 – 10.0.10.50	Automatische Adressvergabe für die Dozenten-PCs
DHCP-Bereich VLAN 20	10.0.20.10 – 10.0.20.50	Automatische Adressvergabe für die Studenten-PCs
WebDAV-Server	10.0.30.2/24	Zentraler Dateiserver im Servernetz

5.3.2 Konfiguration

Routerkonfiguration

Neben dem Routing zwischen den beiden VLANs stellt der Router1 den DHCP-Dienst für die automatische IPv4-Adressvergabe bereit. Router2 verbindet das interne Netzwerk mit dem Webservernetz.

Codeauszug 5.17: Auszug aus der Konfiguration von Router1

```
1
2 !
3 !VLAN Interfaces
4 interface Ethernet1.10
5     encapsulation dot1q vlan 10
6     ip address 10.0.10.1/24
7     dhcp server ipv4
8 !
9 interface Ethernet1.20
10     encapsulation dot1q vlan 20
11     ip address 10.0.20.1/24
12     dhcp server ipv4
13 !
14 interface Ethernet2
15     description "WAN-Link zu router2"
16     no switchport
17     ip address 10.0.1.1/30
18 !
19 ip routing
20 !
21 router ospf 1
22     network 10.0.10.0/24 area 0.0.0.0
23     network 10.0.20.0/24 area 0.0.0.0
24     network 10.0.1.0/30 area 0.0.0.0
25 !
```

Die Schnittstelle Ethernet1 wird als Trunk-Verbindung zum Switch verwendet. Für die beiden VLANs werden die Subinterfaces Ethernet1.10 und Ethernet1.20 eingerichtet. Über die Dot1Q-Kapselung werden die VLAN-Tags 10 bzw. 20 verarbeitet [32].

Für den dynamischen Austausch der Routineinformationen wird OSPF eingesetzt. Dadurch werden die erforderlichen Routen automatisch zwischen beiden Routern ausgetauscht. Router1 kündigt die beiden VLAN-Netze sowie das WAN-Netz zu Router2 an

Switchkonfiguration

Der virtuelle Arista-cEOS-Switch übernimmt die logische Trennung der beiden Benutzergruppen in VLAN 10 und VLAN 20.

Codeauszug 5.18: Auszug aus der VLAN-Konfiguration von Switch1

```

1 interface Ethernet2
2     switchport access vlan 10
3 !
4 interface Ethernet3
5     switchport access vlan 20
6 !
7 interface Ethernet5
8     switchport mode trunk
9     switchport trunk allowed vlan 10,20
10    no shutdown
11 !

```

Die Schnittstelle **Ethernet5** verbindet den Switch1 mit Router1 und wurde als Trunk-Port konfiguriert[33].

Konfiguration des Webserver-Containers

Für die Umsetzung des Webserver wurde ein Linux-Container mit dem offiziellen Apache-Image `httpd:2.4`. Die Konfiguration des Webserver-Containers erfolgt direkt in der Topologiedatei. siehe Code 5.19.

Codeauszug 5.19: Auszug aus der VLAN-Konfiguration des Webserver-Containers

```

1 webserver:
2     kind: linux
3     image: httpd:2.4
4     dns:
5     servers:
6         - 8.8.8.8
7     env:
8         DISPLAY: ${DISPLAY}
9     binds:
10        - webserver/webdav.conf:/usr/local/apache2/conf/extra/webdav.conf
11        - /tmp/.X11-unix:/tmp/.X11-unix
12     exec:
13        - sh -c "apt-get update && apt-get install -y iproute2"
14        - sh -c "echo 'Include conf/extra/webdav.conf' >> ↵
          /usr/local/apache2/conf/httpd.conf"
15        - sh -c "mkdir -p /usr/local/apache2/var && chown -R www-data:www-data ↵
          /usr/local/apache2/var"

```

```

16 - sh -c "mkdir -p /usr/local/apache2/htdocs/webdav && chown -R ↵
    www-data:www-data /usr/local/apache2/htdocs/webdav"
17 - sh -c "ip addr add 10.0.30.2/24 dev eth1 || true"
18 - sh -c "ip route add 10.0.0.0/8 via 10.0.30.1 dev eth1 || true"
19 - sh -c "apt-get update && apt-get install -y wireshark"
20 - sh -c "httpd -k graceful"

```

Da das verwendete Apache-Image die benötigten Netzwerkwerkzeuge nicht standardmäßig bereitstellte, wurde zunächst `iproute2` installiert. Damit konnte der Schnittstelle `eth1` die statische IP-Adresse `10.0.30.2/24` zugewiesen werden. Zusätzlich wurde eine Route für die internen Netze über Router2 mit der Adresse `10.0.30.1` eingerichtet.

Die Datei `webdav.conf` wurde über `binds` in das Apache-Konfigurationsverzeichnis eingebunden. Zusätzlich wird der X11-Socket eingebunden, um grafische Anwendungen nutzen zu können [34]. Anschließend wurde sie mit der `Include`-Direktive in die Apache-Hauptkonfiguration eingebunden, damit Apache die darin definierten WebDAV-Einstellungen berücksichtigt. Das verwendete Docker-Image speichert seine Apache-Konfiguration unter `/usr/local/apache2/conf/` und die bereitgestellten Webinhalten unter `/usr/local/apache2/htdocs/` [35].

Außerdem wurde mit `mkdir` das Verzeichnis für WebDAV-Lock-Datei erstellt und die benötigten Zugriffsrechte für Apache-Prozesse gesetzt. Durch `httpd -k graceful` wird die Apache-Konfiguration neu eingelesen.

Konfiguration des WebDAV-Dienstes

Die eigentliche Konfiguration des WebDAV-Dienstes erfolgt über das Apache-Modul `mod_dav` [36], das WebDAV-Funktionalitäten gemäß RFC 4918 [10] bereitstellt. Die Datei `webdav.conf` definiert die grundlegenden Einstellungen des Apache-Webservers sowie die WebDAV-spezifischen Zugriffsberechtigungen. Ein Auszug der Konfigurationsdatei ist in Codeauszug 5.20 dargestellt.

Codeauszug 5.20: Auszug aus der WebDAV-Konfigurationsdatei

```

1 LoadModule dav_module modules/mod_dav.so
2 LoadModule dav_fs_module modules/mod_dav_fs.so
3
4 DavLockDB /usr/local/apache2/var/DavLock
5
6 <VirtualHost *:80>
7     DocumentRoot /usr/local/apache2/htdocs/webdav
8     <Directory /usr/local/apache2/htdocs/webdav>
9         DAV On
10    <RequireAny>
11        Require method GET POST OPTIONS PROPFIND HEAD
12        Require ip 10.0.10.0/24

```

```
13 </RequireAny>
14 </Directory>
15 </VirtualHost>
```

Für die Bereitstellung der WebDAV-Funktionalität werden zunächst die Module `mod_dav` und `mod_dav_fs` geladen. `mod_dav` stellt die benötigten WebDAV-Funktionen zur Verfügung. Das zusätzliche Modul `mod_dav_fs` ermöglicht die Verwendung dieser Funktionen für Dateien und Verzeichnisse auf dem Webserver [36, 37].

Der WebDAV-Dienst wird anschließend innerhlab eines `VirtualHost` auf Port 80 konfiguriert. Die Angabe `*:80` bedeutet, dass der VirtualHost HTTP-Anfragen auf Port 80 für alle lokalen IP-Adressen des Webserver annimmt.

Die Direktive `DavLockDB /usr/local/apache2/var/DavLock` legt den Speicherort der WebDAV-Lock-Datenbank fest. Das Modul `mod_dav_fs` verwendet diese Datenbank zur Verwaltung von WebDAV-Sperren. Dieses Verzeichnis wurde beim Start des Containers erstellt und Apache-Prozess zugänglich gemacht. Der konkrete Speicherort wird von Apache nicht vorgeschrieben und kann entsprechend der jeweiligen Serverumgebung festgelegt werden [38, 39].

Mit `DocumentRoot /usr/local/apache2/htdocs/webdav` wird das Verzeichnis festgelegt, aus dem Apache die WebDAV-Ressourcen bereitstellt[40, 35].

Innerhalb des `<Directory>`-Blocks werden die Einstellungen für das WebDAV-Verzeichnis `/usr/local/apache2/htdocs/webdav` festgelegt. Mit `DAV On` werden die WebDAV-Methoden für dieses Verzeichnis aktiviert.

Die Zugriffssteuerung erfolgt über den Block `<RequireAny>`. Dabei wird der Zugriff erlaubt, wenn mindesten eine der definierten Bedingungen erfüllt ist. Die Methoden `GET`, `HEAD`, `OPTIONS` und `PROPFIND` sind für alle Clients zugelassen. Dadurch können die Studierenden auf die bereitgestellten Dateien zugreifen und Informationen über WebDAV-Ressourcen abrufen. Die zweite Bedingung `Require ip 10.0.10.0/24` erlaubt zusätzlich Clients aus den Dozenten-Netzwerk den Zugriff unabhängig von der verwendeten Methode. Dadurch können Dozenten auch schreibende WebDAV-Operationen wie `PUT` zum Hochladen und `DELETE` zum Löschen von Dateien ausführen, während diese Operationen für Studierende nicht freigegeben sind [41, 42, 43].

Grafische Anwendungen mittels X11-Forwarding

Da Docker-Container standardmäßig keine grafische Benutzeroberfläche besitzen, wurde zur Darstellung grafischer Anwendungen das X11-Forwarding des Host-Systems verwendet. Das X Window System (X11) ist das Grafiksystem von Linux, das die Darstellung grafischer Fenster auf dem Desktop ermöglicht [44]. Dadurch konnten Anwendungen wie Firefox und Wireshark direkt innerhalb der Linux-Container ausgeführt und ihre Benutzeroberfläche

auf dem Ubuntu-Desktop des Host-Rechners angezeigt werden. Die Konfiguration der X11-Forwarding erfordert drei Schritte: Erstens muss der X11-Zugriff für Docker-Container erlaubt werden. Zweitens müssen die Docker-Container entsprechend konfiguriert werden [30]. Drittens werden die grafischen Anwendungen gestartet [45].

Codeauszug 5.21: Konfiguration des X11-Forwardings auf dem Docker-Container

```
1 env:
2   DISPLAY: ${DISPLAY}
3 binds:
4   - /tmp/.X11-unix:/tmp/.X11-unix
5 exec:
6   - sh -c "apk add --no-cache firefox ttf-dejavu &"
```

`DISPLAY: ${DISPLAY}` Übernimmt die Display-Variable des Host-Systems, sodass grafische Anwendungen das Display des Hosts verwenden können. `/tmp/.X11-unix` Bindet den X11-Socket des Host-Systems in den Container ein und ermöglicht die Kommunikation mit dem X-Server. Dadurch können grafische Anwendungen ihre Fenster auf dem Desktop des Host-Rechners anzeigen. Firefox wird im Dozent-Container installiert und öffnet sich direkt auf dem Desktop des Host-Systems.

5.3.3 Durchführung

Zur Überprüfung der implementierten Netzwerktopologie wurden verschiedene Tests durchgeführt. Zunächst wurde die automatische IP-Adressvergabe über DHCP überprüft. Anschließend erfolgte die Validierung der VLANs-Trennung und des Routings. Abschließend wurden die Funktion von WebDAV auf Apache-Webserver sowie der Netzwerkverkehr mithilfe von Wireshark analysiert.

Überprüfung der VLAN-Trennung

Codeauszug 5.22: Überprüfung der VLAN-Zuordnung auf Switch1

```
1 switch1#show vlan
2 VLAN Name Status Ports
3 -----
4 1 default active
5 10 Dozenten active Et1, Et2, Et5, Et6
6 20 Studenten active Et3, Et4, Et5, Et7
```

Die Ausgabe bestätigt, dass die Endgeräte den vorgesehenen VLANs zugeordnet wurden.

OSPF-Test

Codeauszug 5.23: Überprüfung der OSPF-Nachbarschaft auf Router1

```
1 router1#show ip ospf neighbor
2 Neighbor ID Instance VRF Pri State Dead Time Address Interface
3 10.0.30.1 1 default 1 FULL/DR 00:00:33 10.0.1.2 Ethernet2
4
5 router2#show ip ospf neighbor
6 Neighbor ID Instance VRF Pri State Dead Time Address Interface
7 10.0.20.1 1 default 1 FULL/BDR 00:00:29 10.0.1.1 Ethernet1
```

Der Status bestätigt, dass die OSPF-Nachbarschaft erfolgreich aufgebaut wurde und beide Router ihre Routinginformationen vollständig ausgetauscht haben.

WebDAV-Dienst

Zur Überprüfung der Zugriffsrechte wurde Firefox innerhalb des Linux-Container gestartet und die IP-Adresse des Webservers aufgerufen.

Codeauszug 5.24: Starten von Firefox auf dem Client

```
1 xhost +local:
2 docker exec -it clab-szenario2-dozent1 firefox http://10.0.30.2 &
```

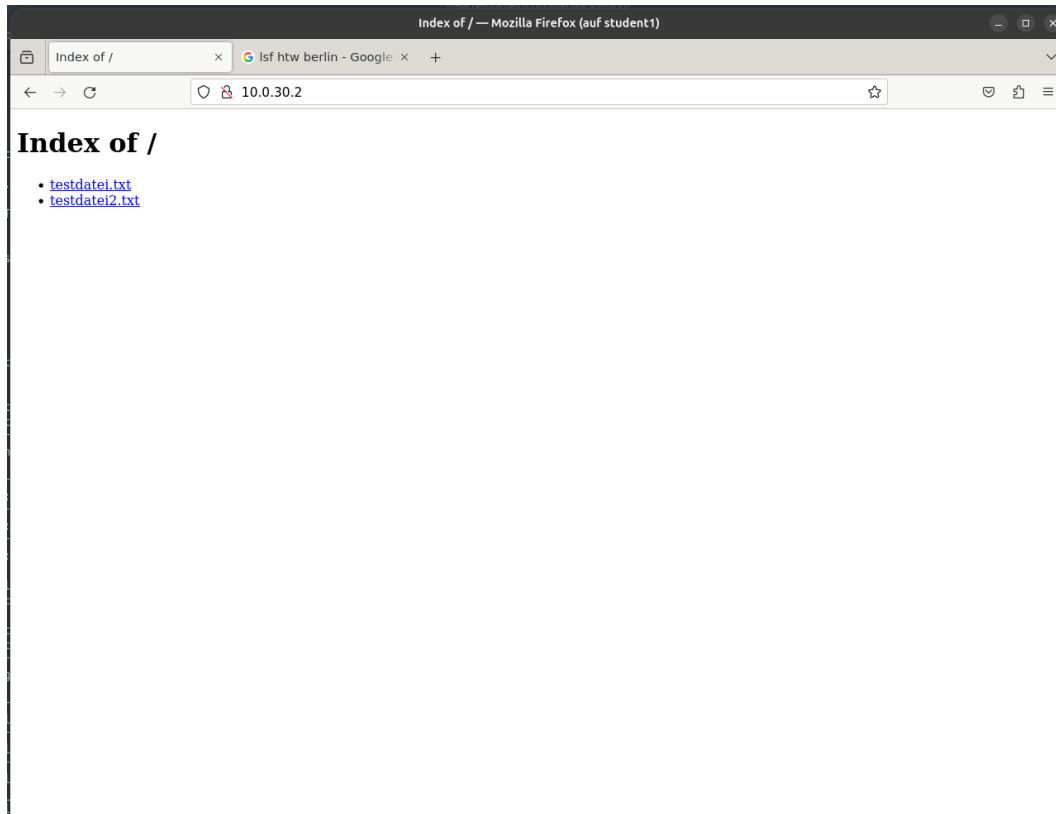
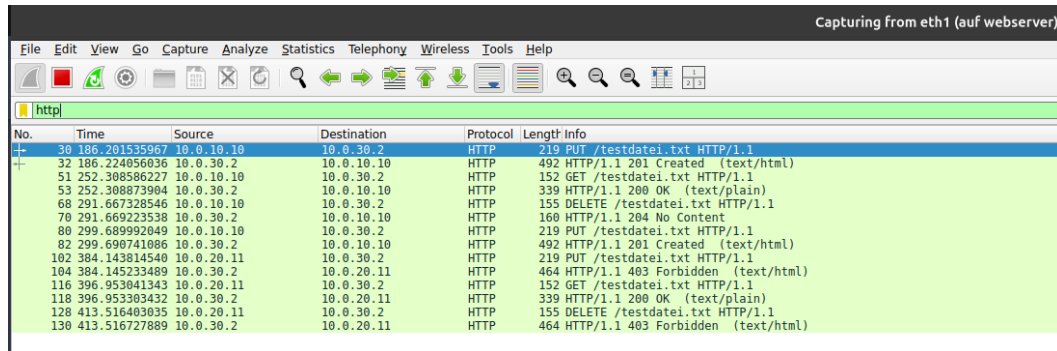


Abbildung 5.5: Grafische Darstellung von Firefox

Die Startseite des WebDAV-Servers wird im Browser angezeigt. Die angezeigte Verzeichnisübersicht bestätigt, dass der Apache-Webserver erreichbar ist und die Grafische Anwendung von Browser ordnungsgemäß funktioniert. Zur Überprüfung der Schreib- und Leseberechtigungen wurde zunächst eine Testdatei vom Host-System in den Client-Container kopiert. Anschließend wurde die Datei auf den Apache-Webserver hochgeladen, heruntergeladen und wieder gelöscht.

Codeauszug 5.25: Befehle von HTTP-Anfragen

```
1 $docker cp <Pfad zur Datei auf dem Hostsystem>/testdatei.txt ↵  
   clab-szenario2-dozent1:testdatei.txt  
2 $docker exec -it clab-szenario2-pc1 sh  
3  /#curl -v -T <Die auf dem Client-Container gespeicherte Datei ↵  
   (testdati.txt)> http://10.0.30.2/testdatei.txt  
4  /#curl http://10.0.30.2/testdatei.txt  
5  /#curl -X DELETE http://10.0.30.2/test.txt
```



No.	Time	Source	Destination	Protocol	Length	Info
30	186.201535967	10.0.10.10	10.0.30.2	HTTP	219	PUT /testdatei.txt HTTP/1.1
32	186.224056036	10.0.30.2	10.0.10.10	HTTP	492	HTTP/1.1 201 Created (text/html)
51	252.308586227	10.0.10.10	10.0.30.2	HTTP	152	GET /testdatei.txt HTTP/1.1
53	252.308873904	10.0.30.2	10.0.10.10	HTTP	339	HTTP/1.1 200 OK (text/plain)
68	291.667328546	10.0.10.10	10.0.30.2	HTTP	155	DELETE /testdatei.txt HTTP/1.1
70	291.669223538	10.0.30.2	10.0.10.10	HTTP	160	HTTP/1.1 204 No Content
80	299.689992049	10.0.10.10	10.0.30.2	HTTP	219	PUT /testdatei.txt HTTP/1.1
82	299.690741086	10.0.30.2	10.0.10.10	HTTP	492	HTTP/1.1 201 Created (text/html)
102	384.143814540	10.0.20.11	10.0.30.2	HTTP	219	PUT /testdatei.txt HTTP/1.1
104	384.145233489	10.0.30.2	10.0.20.11	HTTP	464	HTTP/1.1 403 Forbidden (text/html)
116	396.953041343	10.0.20.11	10.0.30.2	HTTP	152	GET /testdatei.txt HTTP/1.1
118	396.953303432	10.0.30.2	10.0.20.11	HTTP	339	HTTP/1.1 200 OK (text/plain)
128	413.516403035	10.0.20.11	10.0.30.2	HTTP	155	DELETE /testdatei.txt HTTP/1.1
130	413.516727889	10.0.30.2	10.0.20.11	HTTP	464	HTTP/1.1 403 Forbidden (text/html)

Abbildung 5.6: Wireshark-Mitschnitt von HTTP-Methoden

Die Abbildung 5.6 zeigt den mit Wireshark aufgezeichneten HTTP-Verkehr zwischen den Clients und dem Webserver. Für den Dozent-Client mit der IP-Adresse 10.0.10.10 werden die Methoden PUT und DELETE erfolgreich ausgeführt. Der Server antwortet dabei mit 201 Created bzw. 204 No Content. Der Studenten-Client mit der IP-Adresse 10.0.20.11 kann mit GET auf eine Datei zugreifen und erhält die Antwort 200 OK. Die Zugriffe mit den Methoden PUT und DELETE werden dagegen mit 403 Forbidden abgelehnt. Damit zeigt der Mitschnitt die unterschiedlichen Zugriffsrechte für das Dozenten- und Studentennetz.

5.3.4 Testergebnisse

Tabelle 5.6: Zusammenfassung der Testergebnisse des Szenario2

Test	Status	Ergebnis
VLAN-Zuordnung (show vlan)	✓	Alle Endgeräte wurden den vorgesehenen VLANs zugeordnet.
Kommunikation innerhalb VLAN 10	✓	Ping zwischen den Clients erfolgreich.
Kommunikation innerhalb VLAN 20	✓	Ping zwischen den Clients erfolgreich.
Inter-VLAN-Routing	✓	Kommunikation zwischen VLAN 10 und VLAN 20 über Router1 erfolgreich.
OSPF-Nachbarschaft	✓	Nachbarschaft im Zustand FULL; Routinginformationen wurden erfolgreich ausgetauscht.
Erreichbarkeit des Webservers	✓	Webserver über Router2 erfolgreich erreichbar.
Grafische Anwendungen über X11	✓	Firefox und Wireshark konnten innerhalb der Container gestartet und auf dem Desktop des Host-Systems grafisch dargestellt werden.
WebDAV-Zugriff (Dozent)	✓	Datei konnte erfolgreich hochgeladen, heruntergeladen und gelöscht werden 5.20.
WebDAV-Zugriff (Student)	✓	Herunterladen möglich; Schreib- und Löschzugriffe wurden mit 403 Forbidden verweigert.

5.4 Umsetzung Szenario 3

5.4.1 Topologie

Die Topologie des dritten Szenarios besteht aus drei Routern, zwei Layer-2-Switches, zwei transparenten Linux-Firewalls und vier Linux-Containern als Endgeräte. PC1 und PC2 befinden sich im Netzwerk 10.0.1.0/24, während PC3 und PC4 dem Netzwerk 10.0.2.0/24 zugeordnet sind. Zwischen Router1 und Router2 existieren zwei mögliche Übertragungswege. Der Primärpfad verläuft über Firewall1 direkt zwischen Router1 und Router2. Als

alternativer Pfad dient die Verbindung über Firewall2, Router3 und anschließend Router2. Dadurch können neben der Zugriffskontrolle auch das Failover und die Redundanz des Netzwerks untersucht werden.

ContainerLab Topology **SZENARIO3**

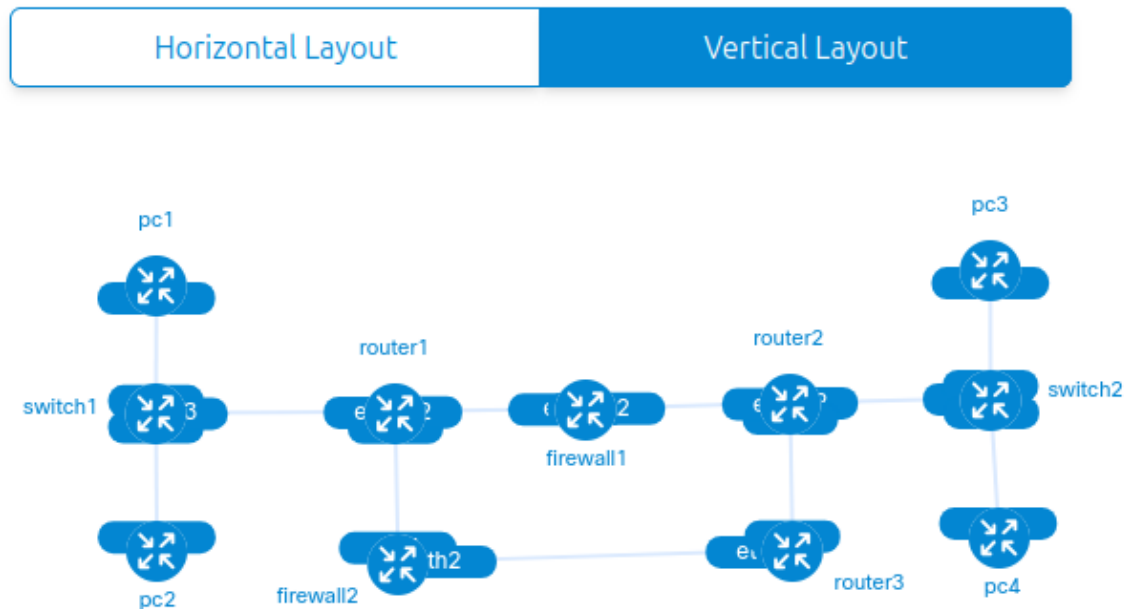


Abbildung 5.7: Containerlab-Topologie des dritten Szenarios

Tabelle 5.7: Adressierung des dritten Containerlab-Szenarios

Komponente	IP-Adresse/Netzwerk	Beschreibung
Router1 (LAN)	10.0.1.1/24	Gateway für Netzwerk 1 mit PC1 und PC2
Router1 (Primärpfad)	10.0.12.1/30	Verbindung über Firewall1 zu Router2
Router2 (Primärpfad)	10.0.12.2/30	Verbindung über Firewall1 zu Router1
Router1(Backup-Pfad)	10.0.13.1/30	Verbindung über Firewall2 zu Router3
Router3 (zu Router1)	10.0.13.2/30	Verbindung des Backup-Pfades zu Router1
Router3 (zu Router2)	10.0.23.1/30	Verbindung des Backup-Pfades zu Router2
Router2 (zu Router3)	10.0.23.2/30	Verbindung des Backup-Pfades zu Router3
Router2 (LAN)	10.0.2.1/24	Gateway für Netzwerk 2 mit PC3 und PC4
DHCP-Bereich Netzwerk 1	10.0.1.20 – 10.0.1.100	Automatische IPv4-Adressvergabe für die Endgeräte im Netzwerk 1
PC2	10.0.1.11	Per DHCP-Reservierung fest zugewiesene IPv4-Adresse
DHCP-Bereich Netzwerk 2	10.0.2.20 – 10.0.2.100	Automatische IPv4-Adressvergabe für die Endgeräte im Netzwerk 2
PC4	10.0.2.11	Per DHCP-Reservierung fest zugewiesene IPv4-Adresse
Multicast-Bereich	224.0.0.0/4	Adressbereich für den Multicast-Verkehr

5.4.2 Konfiguration

Routerkonfiguration

Die Grundkonfiguration der Router entspricht weitgehend den vorherigen Szenarien. Dazu gehören die Konfiguration der Layer-3-Schnittstellen, die Aktivierung des IP-Routings sowie die Verwendung von OSPF für den dynamischen Austausch der Routinginformationen. Im Szenario 3 wurde die Router um Funktionen für Multicast-Übertragung und Zugriffskontrolle erweitert.

Konfiguration der ACL-Regeln

Die in Arista cEOS konfigurierte Access Control (ACL) innerhalb des Router-Containers ermöglichte keine wirksame Filterung des Datenverkehrs. Beim Laden der Konfiguration erschien folgende Warnung:

Codeauszug 5.26: ACL Warnung

```
1 router1#show ip access-lists
2 ! Warning: ACL FW_INBOUND_PC2_PC4 contains rules not supported in data-plane.
```

```

3  Unsupported options will be ignored by hardware.
4  Acl not supported by hardware
5  IP Access List FW_INBOUND_PC2_PC4
6  10 deny ip host 10.0.1.11 host 10.0.2.11
7  20 permit ip any any

```

Zur endgültigen Absicherung dieses Befund wurde die Fragestellung direkt in der offiziellen Arista Community (Diskussionsforum, Kategorie cEOS) gestellt. Ein als Mitarbeiter gekennzeichnete Nutzer (Tag „Arista-Netzwerke“) bestätigte den Befund [46].

Diskussionen Artikel

Sortieren nach:
Top Posts ▼

Benutzer17858830810109377287 Haben Sie eine Frage gestellt.
5. August 2026 um 1:30 Uhr

1. Ist dieses Verhalten eine bekannte Designbeschränkung der Software Forwarding Engine (SFE) innerhalb des cEOS Docker-Containers in Bezug auf L3-geroutete Ports?
2. Was ist die empfohlene Problemumgehung, um die zustandslose L3-Paketfilterung im cEOS-Lab zu erzwingen?

Hallo alle,

Ich simuliere eine Topologie innerhalb von Containerlab mit Arista cEOS (Version:ceos:4.32.0F) für meine Bachelorarbeit.

Ich habe eine Standard-IP-Zugriffsliste auf einer gerouteten Layer-3-Schnittstelle (kein Switchport) konfiguriert, um den Datenverkehr zwischen bestimmten Hosts zu filtern:

Schützels Etikett
Post erweitern

cEOS

Wie Antwort Teilen 2 Antworten · 31 Ansichten

Kshitij Kumar (Arista-Netzwerke)
Vor 3 Tagen

Hallo

cEOS-Instanzen sind CPU-only und haben keine Hardware-ASIC. Bei physischen EOS-Switches werden ACLs direkt in die Hardware ASIC (TCAM) programmiert, um die Daten-Ebenen-Filterung zu erzwingen. Da cEOS diese Hardware vollständig fehlt, können ACL-Regeln in der Datenebene nicht durchgesetzt werden; sie werden in die Konfiguration aufgenommen, aber bei der tatsächlichen Paketweiterleitung stillschweigend ignoriert. Dies ist kein BUG, sondern eine Hardware-Zwänge.

Wie · Antworten · Wählen Sie als Best

Abbildung 5.8: Offizielle Bestätigung - Arista Community

Obwohl die ACL in der Konfiguration vorhanden war, wurde die entsprechende Filterregel im Datenpfad nicht durchgesetzt. Als Alternative zur nicht wirksamen ACL-Filterung in cEOS wurden transparente Linux-Bridge-Firewalls eingesetzt. Eine solche Firewall arbeitet als transparente Zwischenstation zwischen zwei Netzwerkkomponenten. Die beiden Netzwerkschnittstellen des Firewall-Containers werden hierfür zu einer Linux-Bridge zusammengefasst. Dadurch kann der Datenverkehr auf Layer-2 zwischen den beiden Schnittstellen weitergeleitet werden, ohne das bestehende Adressschema der Router zu ändern [47].

In Szenario 3 wurde `firewall1` zwischen Router1 und Router2 auf dem Primärpfad eingesetzt. Eine zweite, identisch konfigurierte Firewall `firewall2` befindet sich auf dem

alternativen Pfad zwischen Router1 und Router3. Dadurch bleibt die Filterung auch bei einem Wechsel auf den Backup erhalten.

Codeauszug 5.27: Konfiguration der transparenten Firewall in Containerlab

```
1 firewall1:
2   kind: linux
3   image: ghcr.io/srl-labs/network-multitool
4
5   exec:
6     - sh -c "ip link add name br0 type bridge"
7     - sh -c "ip link set eth1 master br0"
8     - sh -c "ip link set eth2 master br0"
9     - sh -c "ip link set eth1 up"
10    - sh -c "ip link set eth2 up"
11    - sh -c "ip link set br0 up"
12    - sh -c "iptables -A FORWARD -s 10.0.1.11 -d 10.0.2.11 -j DROP"
13    - sh -c "iptables -A FORWARD -s 10.0.2.11 -d 10.0.1.11 -j DROP"
```

Für die Umsetzung wurde innerhalb jedes Firewall-Containers eine Linux-Bridge `br0` mit `ip link add name br0 type bridge` erstellt und die Schnittstellen `eth1` und `eth2` dieser Bridge mit `ip link set eth1 master br0` zugeordnet und dann aktiviert [48]. Die Filterung des Datenverkehrs erfolgt mit `iptables`. Die beiden `DROP`-Regeln sperren nur die Kommunikation zwischen `10.0.1.11` und `10.0.2.11` in beide Richtungen. Dabei geben `-s` und `-d` die Quell- und Zieladresse an und `-j DROP` legt fest, dass passende Pakete verworfen werden [49].

Damit der über die Linux-Bridge weitergeleitete IPv4-Verkehr durch `iptables` verarbeitet werden kann, wurde der Parameter `net.bridge.bridge-nf-call-iptables` aktiviert. Dafür muss Kernel-Modul `br_netfilter` durch `sudo modprobe br_netfilter` auf dem Host geladen sein [50, 51].

Multicast-Konfiguration

Im Gegensatz zur physischen Cisco-Umgebung, in der PIM Dense Mode eingesetzt wurde, wurde in der verwendeten Arista-cEOS-Umgebung PIM Sparse Mode verwendet. Die aktuelle EOS-Dokumentation beschreibt PIM Sparse Mode und Bidirectional PIM als unterstützte PIM-Mode. Daher wurde für die Containerlab-Implementierung PIM Sparse Mode mit einem statisch konfigurierten Rendezvous Punkt eingesetzt [52].

Codeauszug 5.28: Multicast-Konfiguration am Beispiel von Router1

```
1 interface Ethernet1
2   pim ipv4 sparse-mode
3 interface Ethernet2
4   pim ipv4 sparse-mode
5 interface Ethernet3
```

```

6   pim ipv4 sparse-mode
7 router multicast
8   ipv4
9     routing
10    software-forwarding sfe
11
12 router pim sparse-mode
13   ipv4
14     rp address 10.0.2.1 224.0.0.0/4

```

In der Standard-Konfiguration von cEOS-lab wird Multicast-Routing zwar über den Befehl `Router multicast/ipv4/routing` grundsätzlich aktiviert, dies allein reicht jedoch nicht aus, damit ein Router tatsächlich empfangene Multicast-Pakete verarbeitet wird. Ohne den zusätzlichen Befehl `software-forwarding sfe` blieb die PIM-Routing-Tabelle auch bei nachweislich aktivem, kontinuierlichem Multicast-Sendeverkehr leer. Der Grund liegt darin, dass Multicast-Routing in Arista Netzwerkgeräte standardmäßig deaktiviert ist. Zur Aktivierung wird ein expliziter Software-Forwarding-Agent (Sfe) benötigt, der die eigentliche Paketverarbeitung übernimmt. Dieser Agent ist nicht standardmäßig aktiv. Nach Aktivierung von `software-forwarding sfe` und Neustart konnte eine aktive (S,G)-Weiterleitung mit steigenden Paket- und Byte-Zählern festgestellt werden [53].

Konfiguration der Switches für Multicast

Auf beiden Switches wurde IGMP Snooping aktiviert. Ohne IGMP-Snooping würde der Layer-2-Switch Multicast-Verkehr wie Broadcast behandeln und an alle angeschlossenen Ports senden. IGMP-Snooping verhindert das, indem der Switch die IGMP-Beitrittsnachrichten der Hosts passiv mitliest und daraus eine portbezogene Weiterleitungstabelle aufbaut.

Codeauszug 5.29: Konfiguration der globalen Aktivierung von IGMP Snooping auf den Switches

```

1 switch2# configure terminal
2 switch2(config)# ip igmp snooping

```

Codeauszug 5.30: Aktivierung von IGMP Snooping auf den Switches

```

1 switch2#show ip igmp snooping groups
2 IGMP Snooping Group Membership
3 EX : Filter mode Exclude
4 IN : Filter mode Include
5 IR : Ingress Replication
6 VLAN Group Members
7 -----
8 1 239.1.2.3 Et2
9 1 * Et1

```

Die Ausgabe bestätigt, dass die Gruppe 239.1.2.3 gezielt nur an Port Et2 (PC3 und PC4) weitergeleitet wird.

5.4.3 Durchführung

Nach Abschluss der Konfiguration wurde die Funktionsfähigkeit des Netzwerks schrittweise überprüft. Neben der grundlegenden Netzwerkkonnektivität wurden insbesondere die Filterfunktion der transparenten Firewalls, das Verhalten bei einem Ausfall des Primärpfades sowie die Multicast-Kommunikation untersucht.

Überprüfung der Firewall-Regeln

Codeauszug 5.31: Überprüfung der Kommunikation zwischen Endgeräten

```

1 $docker exec -it clab-szenario3-pc2 ping -c 4 10.0.2.11
2   PING 10.0.2.11 (10.0.2.11) 56(84) bytes of data.
3
4   --- 10.0.2.11 ping statistics ---
5   4 packets transmitted, 0 received, 100% packet loss, time 3076ms
6
7 $docker exec -it clab-szenario3-pc2 ping -c 4
8   10.0.2.20 %(IP-Adresse von PC3)
9   PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
10  64 bytes from 10.0.2.20: icmp_seq=1 ttl=62 time=3.22 ms
11  64 bytes from 10.0.2.20: icmp_seq=2 ttl=62 time=1.68 ms
12  64 bytes from 10.0.2.20: icmp_seq=3 ttl=62 time=1.93 ms
13  64 bytes from 10.0.2.20: icmp_seq=4 ttl=62 time=1.72 ms
14
15  --- 10.0.2.20 ping statistics ---
16  4 packets transmitted, 4 received, 0% packet loss, time 3005ms
17  rtt min/avg/max/mdev = 1.680/2.137/3.220/0.632 ms

```

Überprüfung des Failovers

Zur Überprüfung der Redundanz wurde der Primärpfad zwischen Router1 und Router2 gezielt unterbrochen.

Vor dem Ausfall

Codeauszug 5.32: Vor dem Ausfall

```

1 router1#show ip route ospf
2   0 10.0.2.0/24 [110/20]
3     via 10.0.12.2, Ethernet2
4 router1#show ip ospf neighbor

```

```

5 Neighbor ID Instance VRF Pri State Dead Time Address Interface
6 2.2.2.2 1 default 1 FULL/DR 00:00:30 10.0.12.2 Ethernet2
7 3.3.3.3 1 default 1 FULL/DR 00:00:34 10.0.13.2 Ethernet3

```

Nach dem Ausfall

Codeauszug 5.33: Nach dem Ausfall

```

1 router1(config)#interface Ethernet2
2 router1(config-if-Et2)#shutdown
3 router1(config-if-Et2)#end
4 router1#show ip route ospf
5   0 10.0.2.0/24 [110/30]
6   via 10.0.13.2, Ethernet3
7 router1#show ip ospf neighbor
8 Neighbor ID Instance VRF Pri State Dead Time Address Interface
9 3.3.3.3 1 default 1 FULL/DR 00:00:31 10.0.13.2 Ethernet3

```

Nach der Deaktivierung des Primärpfades wurde das Netzwerk 10.0.2.0/24 nicht mehr über Router2 (10.0.12.2), sondern über Router3 (10.0.13.2) erreicht. Gleichzeitig erhöhte sich die OSPF-Metrik von 20 auf 30. Die OSPF-Nachbarschaft zu Router2 verschwand, während die Nachbarschaft zu Router3 weiterhin den Zustand FULL aufwies. Damit konnte die automatische Umschaltung auf den Backup-Pfad nachgewiesen werden.

Überprüfung der Multicast-Kommunikation

Zur Überprüfung der Multicast-Weiterleitung wurde zunächst `iperf` verwendet. PC1 diene dabei als Sender und PC3 als Empfänger. Als Multicast-Gruppe wurde die Adresse 239.1.2.3 mit dem UDP-Port 1234 verwendet.

Codeauszug 5.34: Start des Multicast-Empfängers auf PC3

```

1 iperf -s -B 239.1.2.3 -u -p 1234 -i 1

```

Codeauszug 5.35: Erzeugung des Multicast-Datenverkehrs auf PC1

```

1 iperf -c 239.1.2.3 -u -b 1M -t 30 -T 11 -p 1234

```

Der Parameter `-T 11` setzt den TTL-Wert der Multicast-Pakete auf 11. Dies ist erforderlich, da der Datenverkehr mehrere Router durchläuft. Bei einem TTL-Wert von 1 würde das Multicast-Paket bereits nach dem ersten Router verworfen werden. Mit `-B 239.1.2.3` wird festgelegt, von welcher Multicast-Gruppe der Empfänger die Daten empfangen soll.

Codeauszug 5.36: Kontrolle der Multicast-Weiterleitung

```

1 router1#show ip mroute
2 239.1.2.3
3 10.0.1.20, 0:00:11, flags: SLP

```

```
4 Incoming interface: Ethernet1
5 RPF route: [U] 10.0.1.0/24 [0/0]
6 Outgoing interface list:
7 Ethernet2
8
9 router1#show multicast fib ipv4 software
10 239.1.2.3 10.0.1.20
11 Packets Received: 5361
12 Bytes Received : 8069544
13 RPF Failures : 3
14
15 router2#show ip mroute
16 239.1.2.3
17 0.0.0.0, 0:00:51, RP 10.0.2.1, flags: W
18 Incoming interface: Register
19 Outgoing interface list:
20 Ethernet2
21 10.0.1.20, 0:00:32, flags: SNCP
22 Incoming interface: Ethernet1
23 RPF route: [U] 10.0.1.0/24 [110/20] via 10.0.12.1
24 Outgoing interface list:
25 Ethernet2
26
27 router2#show ip igmp groups
28 IGMP Connected Group Membership
29 Group Address Interface Uptime Expires Last Reporter
30 239.1.2.3 Ethernet2 0:00:39 0:03:40 10.0.2.20
31
32 router2#show multicast fib ipv4 software
33 239.1.2.3 10.0.1.20
34 Packets Received: 5358
35 Bytes Received : 8101884
36 RPF Failures : 1066
37
38 switch2#show ip igmp snooping groups
39 IGMP Snooping Group Membership
40 EX : Filter mode Exclude
41 IN : Filter mode Include
42 IR : Ingress Replication
43 VLAN Group Members
44 ----
45 1 239.1.2.3 Et2
```

Die Multicast-Routingtabelle von Router1 zeigt einen (S,G)-Eintrag für die Quelle 10.0.1.20 und die Multicast-Gruppe 239.1.2.3. Der Datenverkehr wird über **Ethernet1** empfangen und über **Ethernet2** weitergeleitet. Die steigenden Paket- und Byte-Zähler im

Software-Forwarding bestätigen zusätzlich, dass Multicast-Daten tatsächlich verarbeitet wurden. Auf Router2 ist ebenfalls ein (S,G)-Eintrag für 10.0.1.20 und 239.1.2.3 vorhanden. Zusätzlich zeigt die IGMP-Tabelle, dass sich der Empfänger 10.0.2.20 (PC3) erfolgreich für die Multicast-Gruppe 239.1.2.3 registriert hat.

Während der Multicast-Tests wurden auf Router2 außerdem erhöhte RPF-Failure-Zähler beobachtet. Die Überprüfung der Unicast- und Multicast-Routingtabellen zeigte jedoch einen korrekten RPF-Pfad zur Quelle. Paket-mitschnitte bestätigten zusätzlich, dass der Multicast-Verkehr über das erwartete Interface eintraf und nicht über den Backup-Pfad. Zur besseren Verständnis des beobachteten Verhaltens wurde die Problematik in der Arista Community beschrieben. In der Arista Community wurde als mögliche Erklärung für die erhöhten RPF-Failure-Zähler zunächst der Wechsel vom Rendezvous Point Tree (RPT) auf den Shortest Path Tree (SPT) genannt. Es wurde empfohlen zu prüfen, ob der Zähler nach dem SPT-Wechsel weiter ansteigt und den Wechsel auf den SPT testweise zu unterbinden[54].

Die weiteren Versuchen zeigten, dass die RPF-Failures auch nach dem Aufbau des Shortest Path Tree (SPT) weiter auftraten. Der (S,G)-Eintrag war bereits vorhanden und sowohl der eingehende Pfad als auch die RPF-Route zur Quelle blieben unverändert. Dennoch stieg der RPF-Failure-Zähler auf Router kontinuierlich weiter an. Dabei wurden ungefähr 20% der empfangenen Multicast-Pakete als RPF-Failures gezählt.

Als zusätzlicher Test wurde auf Router2 `spt threshold infinity` konfiguriert. Mit `spt threshold infinity` wird verhindert, dass Router2 dem Shortest Path Tree (SPT) beitrifft. Die Einstellung wurde verwendet, um den Einfluss des SPT-Wechsels auf RPF-Failures zu untersuchen. War Router1 oder Router3 als Rendezvous Point (RP) eingesetzt, sank der Anteil der RPF-Failures auf etwa 0,062% der empfangenen Multicast-Pakete. War Router2 dagegen selbst als RP konfiguriert, wurde trotz dieser Einstellung ein gesetztes SPT-Bit(SNCP) als Flag angezeigt und der Anteil lag weiterhin bei ungefähr 20%.

5.4.4 Testergebnisse

Tabelle 5.8: Testergebnisse von Szenario 3

Test	Status	Beobachtung
Netzwerkcommunication	✓	Kommunikation zwischen den beiden Netzen Über OSPF möglich.
Firewall-Filterung	✓	Kommunikation zwischen PC2 und PC4 erfolgreich blockiert.
OSPF-Failover	✓	Nach Ausfall des Primärpfades erfolgte die Umschaltung über Router3.
Firewall nach Failover	✓	Die Filterung zwischen PC2 und PC4 blieb auch auf dem Backup-Pfad wirksam.
Multicast-Routing	✓	Der (S,G)-Eintrag wurde aufgebaut und der Multicast-Verkehr weitergeleitet.
IGMP Snooping	✓	Switch2 erkannte PC3 als Empfänger der Multicast-Gruppe 239.1.2.3 und leite den Multicast-Verkehr über den entsprechenden Port Ethernet2 weiter.
Multicast-Datenübertragung	✓	Die UDP-Multicast-Übertragung mit <code>iperf</code> wurde erfolgreich nachgewiesen.

5.4.5 Grafische Benutzeroberfläche von Containerlab

Neben der Bedienung über die Kommandozeile stellt Containerlab auch eine grafische Benutzeroberfläche (GUI) zur Verfügung. Diese ermöglicht es, Netzwerktopologie grafisch zu erstellen, zu bearbeiten und zu verwalten. Die GUI basiert auf dem gemeinsamen Topologie-Editor *Topo Viewer* und steht unter anderem als VS-Code-Erweiterung sowie als Desktop- und Web-Anwendung zur Verfügung [55].

Über den Topologie-Editor können Netzwerkkomponenten hinzugefügt, miteinander verbunden und ihre Eigenschaften bearbeitet werden. Änderungen an der Topologie können weiterhin in der zugehörigen `*.clab.yml`-Datei gespeichert werden. Darüber hinaus können vorhandene Labs über die grafische Oberfläche bereitgestellt, neu bereitgestellt, untersucht und beendet werden. Damit stellt die GUI insbesondere für visuelle Erstellung und Verwaltung von Containerlab-Topologie eine Ergänzung zur klassischen Bedienung über Kommandozeile dar [55].

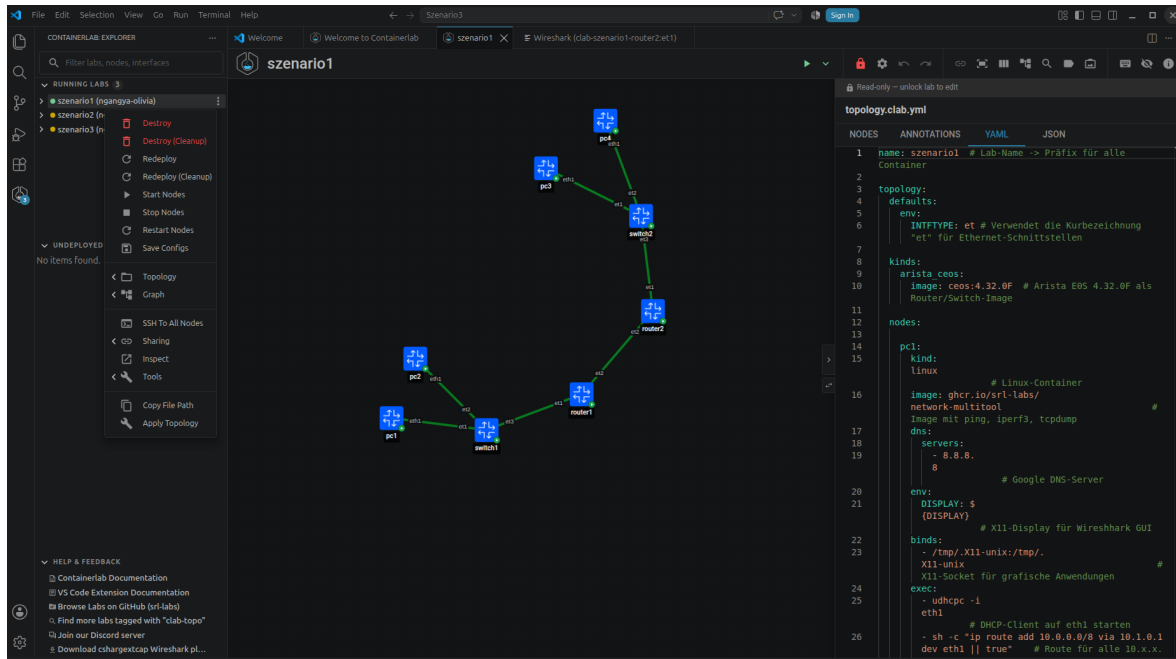


Abbildung 5.9: Graphische Benutzeroberfläche von Containerlab

Paket Aufzeichnung über VS-Code-Erweiterung

Neben der grafischen Darstellung und Verwaltung der Topologie ermöglicht die Containerlab-Erweiterung für Visual Studio Code auch die direkte Aufzeichnung des Netzwerkverkehrs. Hierfür kann an einem laufenden Netzwerknoten eine bestimmte Schnittstelle ausgewählt und eine Paketaufzeichnung mit Wireshark gestartet werden. Für die Paketaufzeichnung wird zunächst Edgeshark gestartet. Anschließend wird in der Containerlab-Erweiterung der gewünschte Netzwerknoten ausgewählt. Edgeshark ist ein Tool, der von Siemens entwickelt wurde. Es virtualisiert die Kommunikation von Containern und hilft so bei Diagnose, sowohl zwischen Containern als auch mit der Außenwelt. Er unterstützt auch das Erfassen Containerverkehr mit Wireshark.

Im Versuch wurde Router2 aus Szenario 1 verwendet. Durch einen Rechtsklick auf Router2 kann die Funktion zur Paketaufzeichnung ausgewählt werden. Danach wird die gewünschte Netzwerkschnittstelle des Routers ausgewählt. Wireshark öffnet sich anschließend direkt in Visual Studio Code und zeigt den laufenden Netzwerkverkehr dieser Schnittstelle an.

Damit kann der Datenverkehr an unterschiedlichen Schnittstellen der simulierten Netzwerknoten mit wenigen Schritten aufgezeichnet und analysiert werden.

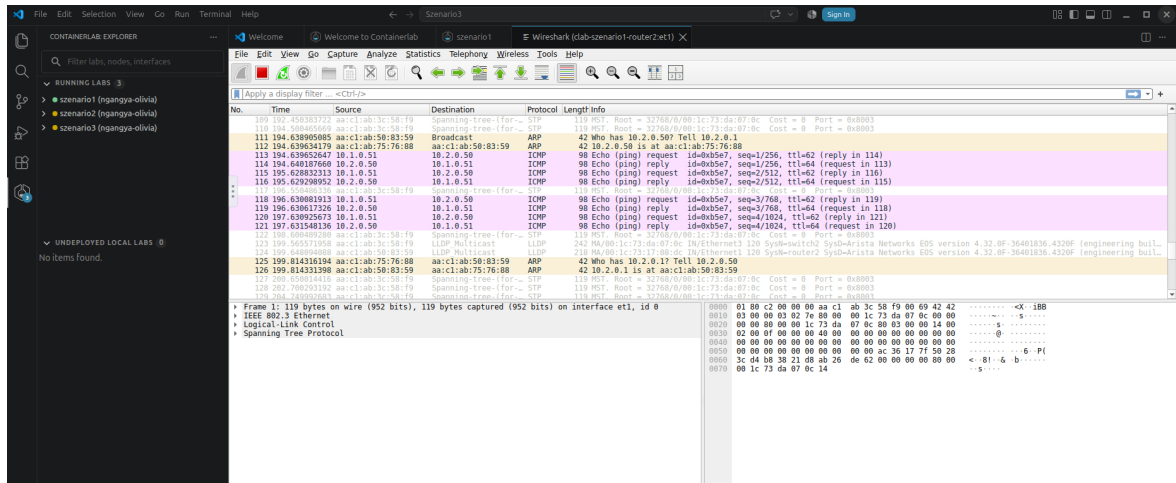


Abbildung 5.10: Live-Paketaufnahme an Router2 in Szenario1 über die VS Code-Erweiterung

5.4.6 Bewertung der praktischen Umsetzung mit Containerlab

Die praktische Umsetzung der drei Szenarien zeigte, dass sich mit Containerlab unterschiedliche virtuelle Netzwerktopologien aufbauen und testen lassen. Die benötigten Netzwerkkomponenten konnten in einer gemeinsamen Topologiedatei definiert, miteinander verbunden und wiederholt bereitgestellt werden.

In den Umgesetzten Szenarien wurden sowohl grundlegende als auch erweiterte Netzwerkfunktionen untersucht. Dazu gehörten unter anderem VLANs, OSPF-Routing, Zugriffskontrollen, redundante Netzwerkverbindungen und Multicast-Kommunikation. Durch die Verwendung unterschiedlicher Container-Images konnten zudem virtuelle Netzwerkgeräte und Linux-basierte Endgeräte innerhalb derselben Topologie kombiniert werden.

Änderungen an der Topologie oder an einzelnen Gerätekonfigurationen konnten vorgenommen und das gesamte Lab anschließend neu gestartet werden. Die gespeicherte Konfiguration ermöglichte es außerdem, dieselbe Netzwerktopologie für mehrere Testdurchläufe erneut bereitzustellen. Auch die grafische Verwaltung der Topologie und die direkte Paketaufnahme mit Wireshark über VS Code-Erweiterung erleichterten die Durchführung und Analyse der Versuche.

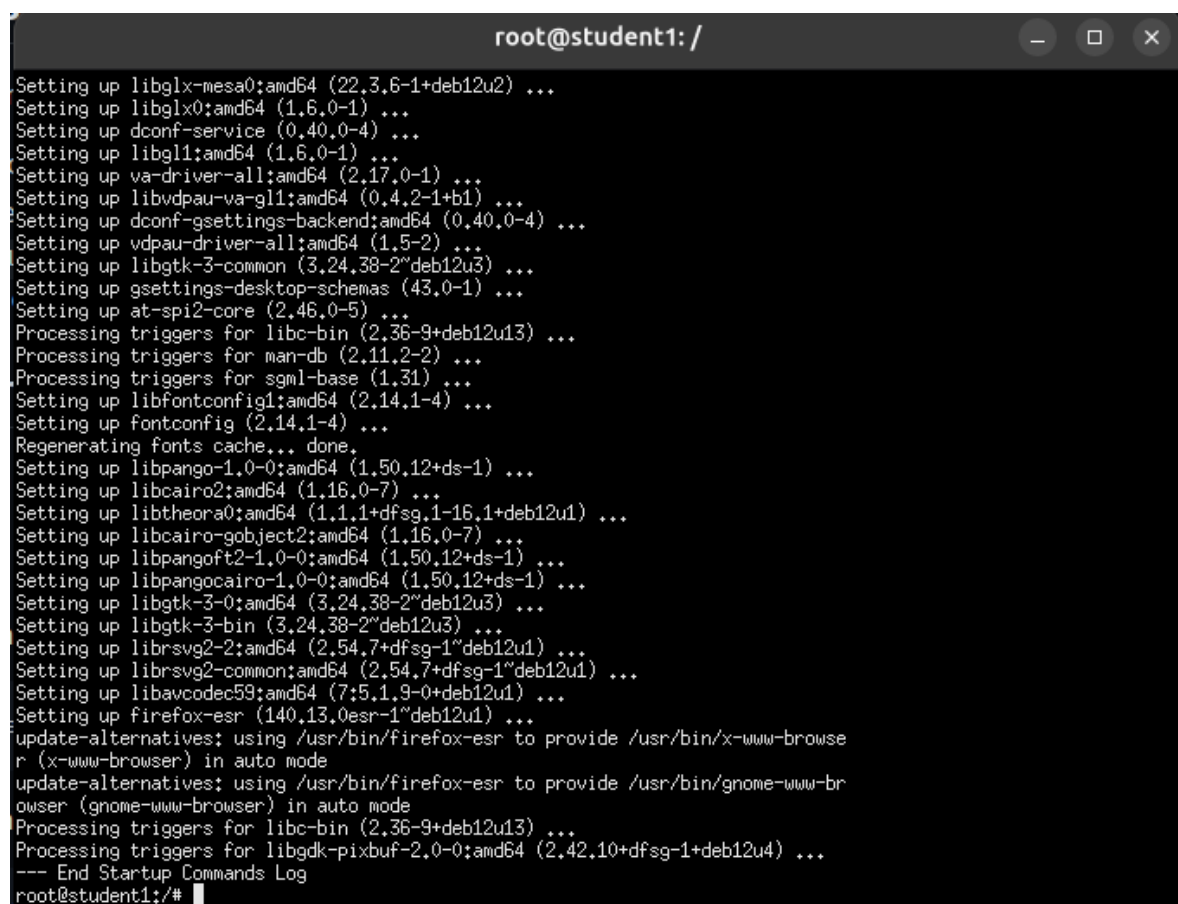
Bei der Umsetzung zeigte sich jedoch auch, dass die verfügbaren Netzwerkfunktionen nicht allein von Containerlab abhängen. Containerlab übernimmt hauptsächlich die Definition, Bereitstellung und Vernetzung der Container. Welche Netzwerkfunktionen innerhalb eines Knotens zur Verfügung stehen, wird dagegen wesentlich durch das verwendete Container-Image bestimmt.

Kapitel 6: Umsetzung in Kathara

6.1 Installation und Vorbereitung

6.1.1 Installation von Kathara

Für die Durchführung der Netzwerkszenarien wurde Kathara unter Ubuntu 26.04 LTS eingesetzt. Kathara verwendet Container zur Abbildung der virtuellen Netzwerkgeräte und benötigt daher eine Container-Laufzeitumgebung. Als Container-Laufzeitumgebung wurde Docker verwendet. Die Dokumentation empfiehlt außerdem die Installation von `xterm` Terminal-Emulator (`sudo apt install xterm`), die standardmäßig verwendet wird.



```
root@student1: /
Setting up libglx-mesa0:amd64 (22.3.6-1+deb12u2) ...
Setting up libglx0:amd64 (1.6.0-1) ...
Setting up dconf-service (0.40.0-4) ...
Setting up libgl1:amd64 (1.6.0-1) ...
Setting up va-driver-all:amd64 (2.17.0-1) ...
Setting up libvdpau-vanilla:amd64 (0.4.2-1+b1) ...
Setting up dconf-gsettings-backend:amd64 (0.40.0-4) ...
Setting up vdpau-driver-all:amd64 (1.5-2) ...
Setting up libgtk-3-common (3.24.38-2~deb12u3) ...
Setting up gsettings-desktop-schemas (43.0-1) ...
Setting up at-spi2-core (2.46.0-5) ...
Processing triggers for libc-bin (2.36-9+deb12u13) ...
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for sgml-base (1.31) ...
Setting up libfontconfig1:amd64 (2.14.1-4) ...
Setting up fontconfig (2.14.1-4) ...
Regenerating fonts cache... done.
Setting up libpango-1.0-0:amd64 (1.50.12+ds-1) ...
Setting up libcairo2:amd64 (1.16.0-7) ...
Setting up libtheora0:amd64 (1.1.1+dfsg.1-16.1+deb12u1) ...
Setting up libcairo-gobject2:amd64 (1.16.0-7) ...
Setting up libpangoft2-1.0-0:amd64 (1.50.12+ds-1) ...
Setting up libpangocairo-1.0-0:amd64 (1.50.12+ds-1) ...
Setting up libgtk-3-0:amd64 (3.24.38-2~deb12u3) ...
Setting up libgtk-3-bin (3.24.38-2~deb12u3) ...
Setting up librsvg2-2:amd64 (2.54.7+dfsg-1~deb12u1) ...
Setting up librsvg2-common:amd64 (2.54.7+dfsg-1~deb12u1) ...
Setting up libavcodec59:amd64 (7:5.1.9-0+deb12u1) ...
Setting up firefox-esr (140.13.0esr-1~deb12u1) ...
update-alternatives: using /usr/bin/firefox-esr to provide /usr/bin/x-www-browser
r (x-www-browser) in auto mode
update-alternatives: using /usr/bin/firefox-esr to provide /usr/bin/gnome-www-br
owser (gnome-www-browser) in auto mode
Processing triggers for libc-bin (2.36-9+deb12u13) ...
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.10+dfsg-1+deb12u4) ...
--- End Startup Commands Log
root@student1:/#
```

Abbildung 6.1: Terminal-Emulator xterm

Die Installation von Kathara erfolgte über für Ubuntu bereitgestellte PPA-Repository. Dazu wurde zunächst der zugehörige Schlüssel importiert und anschließend das Kathara-Repository zum System hinzugefügt. Nach der Aktualisierung der Paketquellen wurde Kathara über den Paketmanager installiert. Abschließend wurde die erfolgreiche Installation durch die Abfrage der Installierten Version überprüft [56].

Codeauszug 6.1: Installation von Kathara unter Ubuntu

```
1 gpg --keyserver keyserver.ubuntu.com --recv-keys ↵
   21805A48E6CBBA6B991ABE76646193862B759810
2 sudo add-apt-repository ppa:katharaframework/kathara
3 sudo apt update
4 sudo apt install kathara
5
6 kathara --version
```

6.1.2 Verwendete Docker-Images

Für die Umsetzung der Netzwerkszenarien wurden hauptsächlich die Docker-Images `kathara/frr` und `kathara/base` verwendet.

Für die Router wurde das Image `kathara/frr` verwendet. Dieses Image stellt FRRouting (FRR) innerhalb eines Containers vom Kathara-Labor zur Verfügung und ermöglicht dadurch die Umsetzung von Routing-Funktionen und Routingprotokollen [17].

Für die PC-Clients wurde das Image `kathara/base` verwendet. Dieses Image stellt eine Linux-basierte Umgebung bereit [17]. Tabelle 6.1 zeigt die wichtigsten Komponenten der verwendeten Testumgebung.

Tabelle 6.1: Systemumgebung für Kathara

Komponente	Version	Zweck
Betriebssystem	Ubuntu 26.04 LTS	Host-System
Docker Engine	29.5.3	Container-Laufzeitumgebung
Kathara	3.8.3	Netzwerk-Emulationsframework
Kathara/frr	latest	Routerbetriebsystem
Kathara/base Image	latest	PC-Clients

Die Zuordnung der Images erfolgt in der datei `lab.conf`.

6.2 Umsetzung Szenario 1

6.2.1 Topologie

Die Topologie von Szenario1 besteht aus vier Client-PCs und zwei Routern. PC1 und PC2 sind der Kollisionsdomäne **CD1** zugeordnet, während PC3 und PC4 der Kollisionsdomäne **CD2** zugeordnet sind. Router1 ist ebenfalls mit der Kollisionsdomäne **CD1** verbunden und Router2 mit Kollisionsdomäne **CD2**. Die Verbindung zwischen den beiden Routern wird über die zusätzliche Kollisionsdomäne **wan** hergestellt. Die verwendete Adressierung ist in der Tabelle 6.2 zusammengefasst. Die Router verwenden statische konfigurierte IP-Adressen.

Tabelle 6.2: Adressierung der Geräte in Kathara – Szenario 1

Gerät	Interface	IP-Adresse	Beschreibung
Router1 (CD1)	eth0	10.1.0.1/24	Gateway und DHCP-Server für Clients aus der Kollisionsdomäne 1
Router1 (WAN)	eth1	192.168.100.1/30	Verbindung zu Router2
Router2 (CD2)	eth0	10.2.0.1/24	Gateway und DHCP-Server für Clients aus der Kollisionsdomäne 2
Router2 (WAN)	eth1	192.168.100.2/30	Verbindung zu Router1
DHCP-Bereich	eth0	10.1.0.50 – 10.1.0.100	Automatische Adressvergabe für PC1 und PC2
DHCP-Bereich	eth0	10.2.0.50 – 10.2.0.100	Automatische Adressvergabe für PC3 und PC4

6.2.2 Konfiguration

Kathara-Topologiedatei

Die Netzwerktopologie wurde über die Datei `lab.conf` definiert. In dieser Datei werden den verwendeten Geräten, ihre Netzwerkverbindung sowie die zugehörigen Docker-Images festgelegt.

Die Netzwerkverbindungen zwischen den Geräten werden in Kathara über sogenannte Kollisionsdomäne definiert. Dabei werden die Netzwerkschnittstellen der Geräte einer Kollisionsdomäne zugeordnet. Geräte, deren Schnittstellen derselben Kollisionsdomäne zugeordnet sind, können direkt miteinander kommunizieren [57].

Codeauszug 6.2: Definition der Netzwerkverbindungen in der Datei `lab.conf`

```

1 pc1[0]="CD1"
2 pc2[0]="CD1"
3 pc3[0]="CD2"
4 pc4[0]="CD2"
```

```
5 router1[0]="CD1"  
6 router1[1]="wan"  
7 router2[0]="CD2"  
8 router2[1]="wan"
```

Die Nummer in den eckigen Klammern legt dabei die jeweilige Netzwerkschnittstelle des Containers fest. Damit sind PC1 und PC2 gemeinsam mit `eth0` von Router1 an Kollisionsdomäne CD1 angeschlossen. PC3 und PC4 befinden sich zusammen mit `eth0` von Router2 in der Kollisionsdomäne CD2. Die Schnittstellen `eth1` der beiden Router werden über die Kollisionsdomäne `wan` miteinander verbunden.

Außerdem werden in der Datei `lab.conf` die verwendeten Docker-Images festgelegt:

Codeauszug 6.3: Zuordnung des Docker-Images

```
1 pc1[image]="kathara/base"  
2 pc2[image]="kathara/base"  
3 pc3[image]="kathara/base"  
4 pc4[image]="kathara/base"  
5 router1[image]="kathara/frr"  
6 router2[image]="kathara/frr"
```

Für die Endgeräte wird das Image `kathara/base` verwendet, während bei beiden Routern das Image `kathara/frr` eingesetzt wird.

Anschließend wurde für alle Geräte der Parameter `bridged="true"` gesetzt. Um fehlende Softwarepakete in den Containern der Kathara Umgebung nachinstallieren zu können, ist ein Internetzugang erforderlich. Dieser wird durch den Parameter `bridged=true` in der `lab.conf` bereitgestellt [57].

Für die Endgeräte wurden außerdem die X11-Umgebungsvariable `DISPLAY=:0` sowie der X11-Socket `/tmp/.x11-unix` eingebunden, um grafische Anwendungen aus den Container auf dem Host-System darstellen zu können. Diese Konfiguration sieht so aus:

```
1 pc1[bridged]="true"  
2  
3 router1[bridged]="true"  
4  
5 pc1[env]="DISPLAY=:0"  
6  
7 pc1[volume]="/tmp/.X11-unix|/tmp/.X11-unix|rw"
```

Tabelle 6.3: Verwendete Geräteoptionen in der `lab.conf`

Geraeteoption	Beschreibung
<code>image</code>	Docker-Image, das für dieses Gerät verwendet wird.
<code>bridged</code>	Verbindet das Gerät mit dem Host-Netzwerk, indem eine zusätzliche Netzwerkschnittstelle hinzugefügt wird. Diese Schnittstelle wird über eine NAT-Verbindung mit dem Host-Netzwerk verbunden.
<code>env</code>	Legt eine Umgebungsvariable für das Gerät fest. Das Format ist: <code>ENV_NAME=ENV_VALUE</code> .
<code>volume</code>	Gibt ein zusätzliches Volume an, das auf dem Gerät montiert werden soll. Die Syntax ist <code>HOST GÄSTE [MODE]</code> . Modus kann sein: <code>ro</code> (schreibgeschützt), <code>rw</code> (schreibgeschützt) oder <code>rx</code> .
<code>[0]</code> , <code>[1]</code>	Ordnet die jeweilige Netzwerkschnittstelle eines Geräts einer Kollisionsdomäne zu. Dabei entspricht beispielsweise <code>[0]</code> der Schnittstelle <code>eth0</code> .

Routerkonfiguration

Die beiden Router verbinden die lokalen Netzwerke miteinander. Jeder Router verfügt über eine Schnittstelle `eth0` für die jeweiligen lokalen Netzwerk und eine zweite Schnittstelle `eth1` für die WAN-Verbindung zwischen den Routern. Die Konfigurationen der Router erfolgt über die Startup-Datei, die beim Start geladen wird.

Damit beide lokalen Netzwerke miteinander kommunizieren können, wurde auf jedem Router zusätzlich eine statische Route zum jeweils entfernten Netzwerk eingerichtet. Router2 wird analog konfiguriert. Die verwendete Adressierung ist in Tabelle 6.2 dargestellt.

```

1 #Schnittstelle fuer Kollisionsdomaene 1
2 ip addr add 10.1.0.1/24 dev eth0
3
4 #Schnittstelle fuer Kollisionsdomaene wan
5 ip addr add 192.168.100.1/30 dev eth1
6 ip route add 10.2.0.0/24 via 192.168.100.2

```

Für die automatische Vergabe der IP-Adressen an die Client-PCs wird auf beiden Routern `dnsmasq` als DHCP-Server eingesetzt [58].

Router1 stellt für Netzwerk 1 durch DHCP-Server den Adressbereich `10.1.0.50–10.1.0.100` bereit und verteilt `10.1.0.1` als Gateway. Router2 übernimmt dieselbe Aufgabe für Netzwerk 2 mit dem Adressbereich `10.2.0.50–10.2.0.100` und mit dem Gateway `10.2.0.1`.

Die DHCP-Server werden jeweils an der Schnittstelle `eth0` gebunden. Dadurch erhalten die Client-PCs nach dem Start des Labs automatisch eine IP-Konfiguration aus dem jeweiligen lokalen Netzwerk.

```
1 #dnsmasq fuer DHCP-Server im Router1
2 cat > /etc/dnsmasq.conf << 'EOF'
3 port=0
4 interface=eth0
5 dhcp-range=10.1.0.50,10.1.0.100,255.255.255.0,210d
6 EOF
7
8 dnsmasq
9
10 #dnsmasq fuer DHCP-Server im Router2
11 cat > /etc/dnsmasq.conf << 'EOF'
12 port=0
13 interface=eth0
14 dhcp-range=10.2.0.50,10.2.0.100,255.255.255.0,210d
15 EOF
16
17 dnsmasq
```

Die Konfiguration des DHCP-Servers erfolgt über die Standard-Konfigurationsdatei von `dnsmasq` unter `/etc/dnsmasq.conf`. Der Parameter `port=0` deaktiviert die DNS-Funktion von `dnsmasq`, da `dnsmasq` in diesem Szenario ausschließlich als DHCP-Server eingesetzt wird. Mit `interface=eth0` wird festgelegt, dass der DHCP-Dienst über die LAN-Schnittstelle bereitgestellt wird. `dhcp-range` definiert den Verfügbaren IP-Adressbereich sowie die Subnetzmaske und die Lease-Dauer.

Konfiguration der Endgeräte

Die IP-Konfiguration der Client-PCs erfolgt automatisch über DHCP. Beim Start eines Endgerätes fordert dieser mit `dhclient eth0` eine IP-Adresse für die Schnittstelle `eth0` an. Die DHCP-Versorgung ist nach den beiden lokalen Netzwerken getrennt: DHCP im Router1 versorgt PC1 und PC2, während DHCP im Router2 PC3 und PC4 versorgt.

Zusätzlich wurde auf den Clients eine Route über das jeweilige lokale Gateway eingerichtet. In der verwendeten Containerumgebung war zusätzlich eine Docker-Netzwerkschnittstelle vorhanden, über die eine Default-Route eingerichtet war. Dadurch konnten Pakete zu den für die Szenarien verwendeten privaten Netzwerken (10.x.x.x) über dieses Default-Route geleitet werden, anstatt den vorgesehenen Weg über die konfigurierten Router zu nehmen[59]. Dies Verhalten konnte während der Tests beobachtet werden, als Ping-Pakete an 10.2.0.x mit der Antwort *Packet filtered* von der externen IP-Adresse 85.232.0.55 zurückkamen, anstatt vom Ziel-PC. Das hinzufügen einer spezifischen Route für 10.0.0.0/8 stellte eine gute Lösung dar.

```
1 #Netzwerk 1 (PC1, PC2)
2 ip route add 10.0.0.0/8 via 10.1.0.1 dev eth0 || true
3
4 #Netzwerk 2 (PC3, PC4)
5 ip route add 10.0.0.0/8 via 10.2.0.1 dev eth0 || true
```

Der DNS-Server 8.8.8.8. wird zusätzlich in der Datei `/etc/resolv.conf` eingetragen. Diese Datei legt fest, welche DNS-Server der Container für die Namensauflösung verwendet [60].

```
1 echo "nameserver 8.8.8.8" > /etc/resolv.conf
```

Anschließend werden die für die Aufzeichnung und Analyse des Netzwerkverkehrs benötigten Werkzeuge wie Wireshark installiert:

```
1 apt-get update -q
2 apt-get install -y -q wireshark
```

6.2.3 Durchführung und Testergebnisse

Starten und Überprüfung des Labs

Nach Abschluss der Konfiguration wurde zunächst in das Verzeichnis von Szenario1 gewechselt. Dort befinden sich die Containerlab-Topologiedatei sowie die Startup-Dateien der Client-PCs und Router. Mit dem Befehl `kathara lstart` werden die in der Datei `lab.conf` definierten Geräte und Kollisionsdomäne erstellt und die zugehörigen Startup-Dateien ausgeführt [61].

Codeauszug 6.4: Starten des Kathara-Labs

```
1 cd ~/Bachelorarbeit/Kathara/Szenario1
2 kathara lstart
```

Nach einem erfolgreichen Start sollte folgende Tabelle angezeigt werden:

```

Starting Network Scenario

Name: Szenario1
Description: Einfaches Netzwerk mit zwei Routern, vier PCs
Version: 2.0
Author(s): ngangya-olivia

[Deploying collision domains] 3/3
The following devices have volumes configured:
* Device 'pc1'
  └─ Host Path: /tmp/.X11-unix -> Device Path: /tmp/.X11-unix
* Device 'pc2'
  └─ Host Path: /tmp/.X11-unix -> Device Path: /tmp/.X11-unix
* Device 'pc3'
  └─ Host Path: /tmp/.X11-unix -> Device Path: /tmp/.X11-unix
* Device 'pc4'
  └─ Host Path: /tmp/.X11-unix -> Device Path: /tmp/.X11-unix
[Deploying devices] 6/6

```

Abbildung 6.2: Erfolgreicher Start des Kathara-Netzwerkszenarios

Mit Kathara connect kann auf einzelne Geräte zugegriffen werden:

```

1 #Beispiel von Zugriff auf PC1
2 kathara connect pc1

```

Überprüfung der DHCP-Adressvergabe

Nach dem Start des Labs wurde zunächst überprüft, ob die Client-PCs erfolgreich eine IP-Adresse über DHCP erhalten haben. Die aktuelle Adressierung eines Clients wurde kontrolliert.

Codeauszug 6.5: Überprüfung der IP-Adresse auf PC1

```

1 root@pc1:/# ip addr show eth0
2 24: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state ↔
   UP group default qlen 1000
3 link/ether e2:ee:f0:b6:53:a9 brd ff:ff:ff:ff:ff:ff
4 inet 10.1.0.87/24 brd 10.1.0.255 scope global dynamic eth0
5 valid_lft 18143490sec preferred_lft 18143490sec
6
7
8 root@router1:/# cat /var/lib/misc/dnsmasq.leases
9 1804670899 aa:ef:f1:7f:4f:8d 10.1.0.54 pc2 *
10 1804670899 e2:ee:f0:b6:53:a9 10.1.0.87 pc1 *
11
12 root@router2:/# cat /var/lib/misc/dnsmasq.leases
13 1804670899 12:62:7f:1d:ac:62 10.2.0.81 pc3 *
14 1804670899 92:20:7f:68:4e:6f 10.2.0.98 pc4 *

```

Ergebnisse: Alle vier PCs erhielten automatisch eine IP-Adresse. Der DHCP-Server im Router funktioniert korrekt mit einer Lease-Zeit von 210 Tagen.

Ping-Test

Die Konnektivität innerhalb und außerhalb der lokalen Netzwerke wurde überprüft.

Konnektivität innerhalb des lokalen Netzwerkes

Codeauszug 6.6: Ping-Test zwischen PC1 und PC2

```
1 root@pc1:/# ping -c 3 10.1.0.54
2 PING 10.1.0.54 (10.1.0.54) 56(84) bytes of data.
3 64 bytes from 10.1.0.54: icmp_seq=1 ttl=64 time=1.02 ms
4 64 bytes from 10.1.0.54: icmp_seq=2 ttl=64 time=0.382 ms
5 64 bytes from 10.1.0.54: icmp_seq=3 ttl=64 time=0.305 ms
6
7 --- 10.1.0.54 ping statistics ---
8 3 packets transmitted, 3 received, 0% packet loss, time 2055ms
9 rtt min/avg/max/mdev = 0.305/0.569/1.021/0.320 ms
```

Konnektivität außerhalb des lokalen Netzwerkes

Codeauszug 6.7: Ping-Test zwischen PC1 und PC3

```
1 root@pc1:/# ping -c 3 10.2.0.81
2 PING 10.2.0.81 (10.2.0.81) 56(84) bytes of data.
3 64 bytes from 10.2.0.81: icmp_seq=1 ttl=62 time=1.23 ms
4 64 bytes from 10.2.0.81: icmp_seq=2 ttl=62 time=0.813 ms
5 64 bytes from 10.2.0.81: icmp_seq=3 ttl=62 time=0.830 ms
6
7 --- 10.2.0.81 ping statistics ---
8 3 packets transmitted, 3 received, 0% packet loss, time 2033ms
9 rtt min/avg/max/mdev = 0.813/0.956/1.226/0.190 ms
```

Ergebnis: 0 % Paketverlust in allen Testfällen.

Traceroute

Nach der erfolgreichen Überprüfung der Konnektivität wurde der Übertragungsweg zwischen den beiden lokalen Netzwerken untersucht. Das Werkzeug zeigt die Zwischenstationen an, die ein Paket auf dem Weg zum Ziel durchläuft.

Codeauszug 6.8: Überprüfung des Routing-Pfades zwischen PC1 und PC3

```
1 root@pc1:/# traceroute 10.2.0.81
2 traceroute to 10.2.0.81 (10.2.0.81), 30 hops max, 60 byte packets
3 1 10.1.0.1 (10.1.0.1) 0.458 ms 0.765 ms 1.070 ms
4 2 192.168.100.2 (192.168.100.2) 1.374 ms 1.824 ms 2.111 ms
5 3 10.2.0.81 (10.2.0.81) 2.697 ms 2.998 ms 3.295 ms
```

Ergebnis: 3 Hops - der erwartete Netzwerkpfad wurde bestätigt.

Datendurchsatz-iPerf3-Test

Auf PC3 wurde zunächst der iperf3-Server gestartet:

Codeauszug 6.9: Starten des iperf3-Servers auf PC3

```
1 root@pc3:/#iperf3 -s
2 -----
3 Server listening on 5201 (test #1)
4 -----
5 Accepted connection from 10.1.0.87, port 50226
6 [ 5] local 10.2.0.81 port 5201 connected to 10.1.0.87 port 50240
```

Anschließend wurde auf PC1 die Messung zur IP-Adresse von PC3 gestartet:

Codeauszug 6.10: Starten des iperf3-Client auf PC1

```
1 root@pc1:/# iperf3 -c 10.2.0.81 -t 30
2 Connecting to host 10.2.0.81, port 5201
3 [ 5] local 10.1.0.87 port 50240 connected to 10.2.0.81 port 5201
```

Tabelle 6.4: Ergebnisse der iPerf3-Durchsatzmessung mit unterschiedlichen MSS-Werten

MSS-Wert	Sender (Mbit/s)	Empfänger (Mbit/s)	Retr	Bewertung
Standard (ohne -M)	18,5	17,7	502	Hohe Neuübertragungen
1400 Bytes	18,3	17,4	92	Weniger Neuübertragungen

Ergebnis: Bei der Messung trat trotz eines MTU von 1500 Byte auf den beteiligten Interfaces Neuübertragungen auf. Ein MTU-Mismatch, wie er zuvor bei Containerlab festgestellt wurde, konnte daher nicht nachgewiesen werden. Die Ursache liegt daher wahrscheinlich nicht in einer fehlerhaften MTU-Konfiguration.

Zur weiteren Untersuchungen der TCP-Neuübertragungen wurde zunächst geprüft, ob FRRouting die Ursache sein könnte. Dazu wurden Client und Server in derselben Kollisionsdomäne betrieben. Trotzdem traten bei allen Messungen weiterhin TCP-Neuübertragungen auf.

Zusätzlich wurde die Empfehlung des Kathara-Teams getestet. Nach Aussage des Kathara-Teams verwendet Kathara standardmäßig das VDE Network Plugin. Da dieses benutzerraumbasiert ist, kann es die Netzwerkleistung zusätzlich beeinflussen. Das Kathara-Team empfahl deshalb, in den Kathara-Einstellungen auf das Linux-Bridge-basierte Network Plugin `kathara/katharap` zu wechseln und den Test erneut durchzuführen[62].

Nach dem Wechsel des Network Plugins wurde der TCP-Test wiederholt. Auch hier traten bei einigen Messungen Neuübertragungen auf, während andere Messungen ohne Neuübertragungen verliefen. Die Anzahl der Neuübertragung wurde aber deutlich reduziert. Somit konnte weder FRRouting noch das verwendete Network Plugin als eindeutige Ursache für die beobachteten Neuübertragungen festgestellt werden.

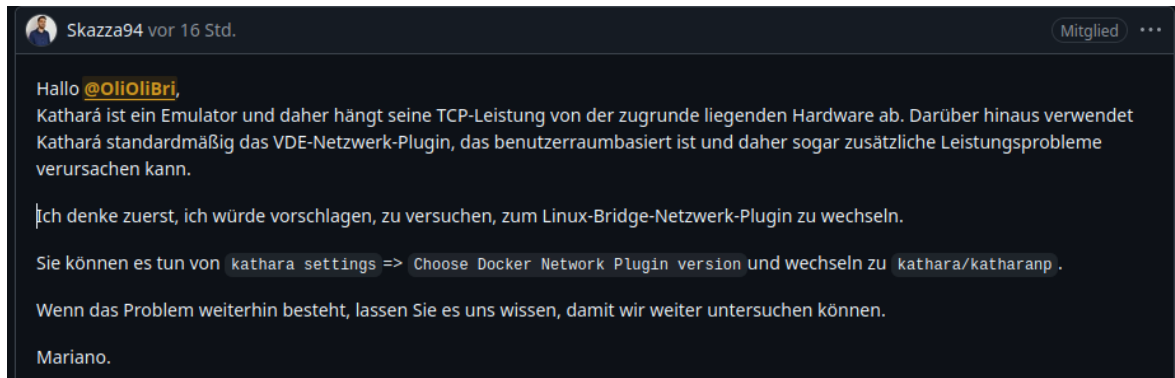


Abbildung 6.3: Empfehlung vom Kathara-Team

Paketaufzeichnung mit Wireshark

Eine Möglichkeit zur Paketaufzeichnung besteht darin, Wireshark direkt innerhalb eines Containers von der Kathara Umgebung zu starten. Die grafische Oberfläche wird dabei über X11 auf dem Ubuntu-Host dargestellt. Als Beispiel wurde Wireshark auf dem PC1 mit folgendem Befehl gestartet:

```
1 xhost +local:
2 docker exec -it <Container-ID> wireshark
3
4 #Wireshark auf PC1 starten
5 docker exec -it 9387b55c2dd8 wireshark
```

Nach dem Start kann in Wireshark die gewünschte Netzwerkschnittstelle, beispielsweise `eth0`, ausgewählt und der darüber laufende Netzwerkverkehr direkt aufgezeichnet werden.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.1.0.87	10.2.0.81	ICMP	98	Echo (ping) request id=0xbcf0, seq=1/256, ttl=64 (reply in 2)
2	0.002002452	10.2.0.81	10.1.0.87	ICMP	98	Echo (ping) reply id=0xbcf0, seq=1/256, ttl=62 (request in 1)
3	1.001289432	10.1.0.87	10.2.0.81	ICMP	98	Echo (ping) request id=0xbcf0, seq=2/512, ttl=64 (reply in 4)
4	1.003722405	10.2.0.81	10.1.0.87	ICMP	98	Echo (ping) reply id=0xbcf0, seq=2/512, ttl=62 (request in 3)
5	2.002147658	10.1.0.87	10.2.0.81	ICMP	98	Echo (ping) request id=0xbcf0, seq=3/768, ttl=64 (reply in 6)
6	2.002904516	10.2.0.81	10.1.0.87	ICMP	98	Echo (ping) reply id=0xbcf0, seq=3/768, ttl=62 (request in 5)
7	5.053149096	e2:ee:f0:b6:53:a9	72:12:5d:76:3c:b4	ARP	42	Who has 10.1.0.1? Tell 10.1.0.87
8	5.054038437	72:12:5d:76:3c:b4	e2:ee:f0:b6:53:a9	ARP	60	Who has 10.1.0.87? Tell 10.1.0.1
9	5.054055481	e2:ee:f0:b6:53:a9	72:12:5d:76:3c:b4	ARP	42	10.1.0.87 is at e2:ee:f0:b6:53:a9
10	5.055396505	72:12:5d:76:3c:b4	e2:ee:f0:b6:53:a9	ARP	60	10.1.0.1 is at 72:12:5d:76:3c:b4
11	68.271688574	10.1.0.87	10.1.0.54	ICMP	98	Echo (ping) request id=0xbcf1, seq=1/256, ttl=64 (reply in 12)
12	68.272571222	10.1.0.54	10.1.0.87	ICMP	98	Echo (ping) reply id=0xbcf1, seq=1/256, ttl=64 (request in 11)
13	69.272863115	10.1.0.87	10.1.0.54	ICMP	98	Echo (ping) request id=0xbcf1, seq=2/512, ttl=64 (reply in 14)
14	69.273853617	10.1.0.54	10.1.0.87	ICMP	98	Echo (ping) reply id=0xbcf1, seq=2/512, ttl=64 (request in 13)
15	70.273255444	10.1.0.87	10.1.0.54	ICMP	98	Echo (ping) request id=0xbcf1, seq=3/768, ttl=64 (reply in 16)
16	70.274286993	10.1.0.54	10.1.0.87	ICMP	98	Echo (ping) reply id=0xbcf1, seq=3/768, ttl=64 (request in 15)
17	73.661135482	e2:ee:f0:b6:53:a9	aa:ef:f1:7f:4f:8d	ARP	42	Who has 10.1.0.54? Tell 10.1.0.87
18	73.661651203	aa:ef:f1:7f:4f:8d	e2:ee:f0:b6:53:a9	ARP	60	10.1.0.54 is at aa:ef:f1:7f:4f:8d
19	73.662191499	aa:ef:f1:7f:4f:8d	e2:ee:f0:b6:53:a9	ARP	60	Who has 10.1.0.87? Tell 10.1.0.54
20	73.662202464	e2:ee:f0:b6:53:a9	aa:ef:f1:7f:4f:8d	ARP	42	10.1.0.87 is at e2:ee:f0:b6:53:a9

Abbildung 6.4: Aufzeichnung des Netzwerkverkehrs mit Wireshark auf dem Container PC1

Eine weitere Möglichkeit Paket aufzuzeichnen und die virtuellen Schnittstellen zwischen den Containern zu überwachen ist es, Wireshark direkt auf dem Host-System zu starten. Dabei sollte der Datenverkehr an ausgewählten virtuellen Schnittstellen der Netzwerkknoten aufgezeichnet und direkt in Wireshark dargestellt werden.

Bei Kathara wurde die Variante über `docker exec` verwendet.

```
1 | docker exec <Container-ID oder Containername> tcpdump -U -nni eth1 -w - <->
   | wireshark -k -i -
```

Kathara verwendet ebenfalls Linux Network Namespace für die Isolation der Container. Diese waren im verwendeten Versuchsaufbau jedoch nicht als benannte Network Namespaces über `ip netns list` verfügbar. Ein Zugriff über die Prozess-ID des Containers war zwar möglich, erfordert jedoch zusätzliche Schritte. Daher wurde für die praktische Paketaufzeichnung die einfachere Variante über `docker exec` verwendet.

Während Wireshark gestartet war, wurde ein Ping-Test von PC1 zu PC3 durchgeführt
Ergebnis: Wireshark bestätigt korrektes TTL-Verhalten an allen Punkten. Identifizierte Protokolle: ICMP und ARP.

6.3 Umsetzung Szenario 2

6.3.1 Topologie

In Szenario 2 werden die Dozenten und Studenten durch VLANs in zwei Netze getrennt. Diese VLAN-Trennung erfolgte durch den Einsatz von einem virtuellen Switch. Die Clients sind über Access-Ports mit dem Switch verbunden. Dieser Switch ist dann über den Trunk-Port mit dem Router1 verbunden. Router1 verbindet dann die beiden VLAN-Netze und ist

über das Netz 10.0.1.0/30 mit Router2 verbunden. Router2 ist auch mit Webserver-Netz verbunden. Für ein dynamisches Routing wurde OSPF eingesetzt.

Tabelle 6.5: Adressierung des zweiten Kathara-Szenarios

Geräte	Schnittstelle	IP-Adresse	Beschreibung
Router1	eth0.10	10.0.10.1/24	Gateway für VLAN 10
Router1	eth0.20	10.0.20.1/24	Gateway für VLAN 20
Router1	eth1	10.0.1.1/30	Verbindung zu Router2
Router2	eth0	10.0.1.2/30	Verbindung zu Router1
Router2	eth1	10.0.30.1/24	Gateway für das Webserver-Netz
Dozent1 & 2	eth0	10.0.10.10 – 10.0.10.50	Client im Dozenten-Netz
Student1 & 2	eth0	10.0.20.10 – 10.0.20.50	Client im Studenten-Netz
Webserver	eth0	10.0.30.2/24	Dateiserver im Webserver-netz

6.3.2 Konfiguration

Konfiguration der Topologiedatei

Die Konfiguration der Topologie erfolgt in der Datei `lab.conf`. In dieser Datei werden die Gräte des Labors sowie deren Verbindungen festgelegt. Die Konfiguration der Topologie ist im folgenden Ausschnitt dargestellt.

Codeauszug 6.11: Definition der Netzwerkverbindungen

```

1  #VLAN10 Dozenten
2  dozent1[0]="CD1_dozenten"
3  dozent2[0]="CD2_dozenten"
4
5
6  #VLAN20 Studenten
7  student1[0]="CD1_studenten"
8  student2[0]="CD2_studenten"
9
10 #Switch
11 switch1[0]="CD1_dozenten"
12 switch1[1]="CD2_dozenten"
13 switch1[2]="CD1_studenten"
14 switch1[3]="CD2_studenten"
15 switch1[4]="CD_trunk"
16
17
18 #Router1
19 router1[0]="CD_trunk"
20 router1[1]="wan"

```

```

21
22
23 #Router2
24 router2[0]="wan"
25 router2[1]="CD_webserver"
26
27 #Webserver
28 webserver[0]="CD_webserver"

```

Konfiguration des virtuelles Switches

Für die Umsetzung der VLANs wird auf `switch1` eine Linux Bridge mit dem Namen `br0` erstellt. Mit `vlan_filtering 1` wird die VLAN-Filterung der Linux Bridge aktiviert. Durch `vlan_default_pvid 0` wird festgelegt, dass die Bridge-Ports nicht automatisch einem VLAN zugeordnet werden.

```

1 ip link add br0 type bridge
2 #Aktivierung der VLAN-Filterung
3 ip link set dev br0 type bridge vlan_filtering 1
4 #Standard PVID deaktivieren
5 ip link set dev br0 type bridge vlan_default_pvid 0

```

Danach werden die fünf Schnittstellen Switches der Bridge `br0` zugeordnet und aktiviert.

```

1 ip link set eth0 master br0
2 ip link set eth0 up
3 ip link set eth1 master br0
4 ip link set eth1 up
5 ip link set eth2 master br0
6 ip link set eth2 up
7 ip link set eth3 master br0
8 ip link set eth3 up
9 ip link set eth4 master br0
10 ip link set eth4 up
11 ip link set br0 up

```

Die Schnittstellen `eth0` und `eth1` werden VLAN 10 für die Dozenten zugeordnet, während die Schnittstellen `eth2` und `eth3` zu VLAN 20 für die Studenten gehören.

Codeauszug 6.12: Konfiguration der Access-Ports und den Trunk-Port

```

1 #Access-Ports VLAN 10 (Dozenten)
2 bridge vlan add dev eth0 vid 10 pvid untagged
3 bridge vlan add dev eth1 vid 10 pvid untagged
4
5 #Access-Ports VLAN 20 (Studenten)
6 bridge vlan add dev eth2 vid 20 pvid untagged

```

```

7 bridge vlan add dev eth3 vid 20 pvid untagged
8
9 #Trunk-Port
10 bridge vlan add dev eth4 vid 10
11 bridge vlan add dev eth4 vid 20

```

Mit `pvid` werden die Daten, die ohne VLAN-Tag am Switch-Port ankommen, dem jeweiligen VLAN zugeordnet. Der Befehl `untagged` ermöglicht, dass die Daten beim Verlassen des Switch-Ports ohne VLAN-Tag an die Clients gesendet werden. Die Schnittstelle `eth4` verbindet den Switch mit Router1. Über diese Verbindung werden die beide VLANs übertragen.

Routerkonfiguration

Die beiden Router übernehmen die Weiterleitung zwischen den verschiedenen Netzwerken des Szenarios. Die Erstellung der VLAN-Subinterfaces auf Router erfolgt in der `startup`-Datei.

Router1 ist über die Schnittstelle `eth0` mit dem Trunk-Port des Switches verbunden. Über diese Verbindung werden VLAN 10 und VLAN 20 übertragen.

Codeauszug 6.13: Erstellung der VLAN-Subinterfaces auf Router1

```

1 #VLAN 10
2 ip link add link eth0 name eth0.10 type vlan id 10
3 ip link set eth0.10 up
4
5 #VLAN 20
6 ip link add link eth0 name eth0.20 type vlan id 20
7 ip link set eth0.20 up

```

Das Subinterface `eth0.10` wird VLAN 10 und `eth0.20` VLAN 20 zugeordnet. Dadurch kann Router1 die über den Trunk ankommenden Daten anhand ihrer VLAN-ID dem richtigen VLAN zuordnen.

Router2 verbindet Router1 mit dem Netz des Webservers. In der Datei `router2.startup` wird FRRouting (FRR) gestartet.

Codeauszug 6.14: Startup-Konfiguration von Router2

```

1 #FRR starten
2 service frr start

```

Nach der Konfiguration der Schnittstellen wird auf beiden Routern FRRouting gestartet. Die dynamische Weitergabe der Routinginformationen zwischen den beiden Routern erfolgt über OSPF.

FRRouting- und OSPF-Konfiguration

Für das dynamische Routing zwischen Router1 und Router2 wird FRRouting (FRR) mit OSPF eingesetzt. Hierfür wurden im Verzeichnis `/etc/frr/` die Dateien `daemons` und `frr.conf` konfiguriert.

Die IP-Adressen der VLAN-Subinterfaces und der Verbindung zu Router2 werden in der Datei `/etc/frr/frr.conf` konfiguriert. Die Angaben `frr version 10.5.3` und `frr defaults traditional` bezeichnet die verwendete FRRouting-Version sowie das eingesetzte Standardprofil [63].

Codeauszug 6.15: OSPF-Konfiguration von Router1

```
1 frr version 10.5.3
2 frr defaults traditional
3
4 hostname router1
5 interface eth0.10
6     ip address 10.0.10.1/24
7
8 interface eth0.20
9     ip address 10.0.20.1/24
10
11 interface eth1
12     ip address 10.0.1.1/30
13
14 router ospf
15     ospf router-id 1.1.1.1
16     network 10.0.10.0/24 area 0.0.0.0
17     network 10.0.20.0/24 area 0.0.0.0
18     network 10.0.1.0/30 area 0.0.0.0
```

Die Subinterfaces `eth0.10` und `eth0.20` erhalten die Gateway-Adressen der beiden VLAN-Netze. Die Schnittstelle `eth1` verbindet Router1 mit Router2. Mit `router ospf` wird OSPF aktiviert. Die Router-ID `1.1.1.1` dient zur eindeutigen Identifikation von Router1 innerhalb von OSPF. Über die `network`-Befehle werden die drei Netze in OSPF eingebunden.

Codeauszug 6.16: OSPF-Konfiguration von Router2

```
1 frr version 10.5.3
2 frr defaults traditional
3
4 hostname router2
5
6 interface eth0
7     ip address 10.0.1.2/30
8
9 interface eth1
```

```
10 ip address 10.0.30.1/24
11
12 router ospf
13     ospf router-id 2.2.2.2
14     network 10.0.1.0/30 area 0.0.0.0
15     network 10.0.30.0/24 area 0.0.0.0
```

Die Schnittstelle `eth0` erhält die Adresse `10.0.1.2/30` und stellt die Verbindung zu Router1 her. Über `eth1` ist Router2 mit dem Webserver-Netz `10.0.30.0/24` verbunden. Für OSPF wird Router2 die Router-ID `2.2.2.2` zugewiesen. Die Netze `10.0.1.0/30` und `10.0.30.0/24` werden anschließend in OSPF eingebunden.

In der Datei `daemons` werden die für das Szenario benötigten FRR-Dienste aktiviert. Dabei werden `zebra` und `ospfd` verwendet. Hierfür werden `zebra` und `ospfd` auf `yes` gesetzt [64].

Codeauszug 6.17: Aktivierte FRRouting-Daemons

```
1 zebra=yes
2 bgpd=no
3 ospfd=yes
4 ospf6d=no
5 ripd=no
6 ripngd=no
7 isisd=no
8 pimd=no
9 ldpd=no
10 nhrpd=no
11 eigrpd=no
12 babeld=no
13 sharpd=no
14 pbrd=no
15 bfdd=no
16 fabricd=no
17 vrrpd=no
18 pathd=no
```

Der Daemon `zebra` fungiert innerhalb von FRRouting als IP-Routing-Manager. Es bietet Kernel-Routing Tabellen-Updates, Interface-Lookups und Umverteilung von Routen zwischen verschiedene Routing-Protokolle [65]. Der Daemon `ospfd` stellt die OSPFv2-Funktionalität für dynamische Routing bereit [66].

Durch diese Konfigurationen können die beiden Router OSPF-Nachbarschaften aufbauen und Routinginformationen über die jeweils angeschlossenen Netze dynamisch austauschen.

DHCP-Konfiguration

Router1 vergibt dem Dozenten- und Studenten-Netz durch DHCP-Konfiguration beim Start automatisch IPv4 Adressen. Als DHCP-Server wird `dnsmasq` eingesetzt. Für beide Netzwerkbereiche werden getrennte Adressbereiche definiert.

Codeauszug 6.18: DHCP-Konfiguration auf Router1

```
1 port=0
2 interface=eth0.10
3 interface=eth0.20
4 dhcp-range=10.0.10.10,10.0.10.50,255.255.255.0,210d
5 dhcp-range=10.0.20.10,10.0.20.50,255.255.255.0,210d
```

Die Schnittstelle `eth0.10` gehört zu VLAN 10 und `eth0.20` zu VLAN 20. Für jedes VLAN wird ein eigener DHCP-Bereich festgelegt. Dadurch erhalten die Dozenten IP-Adressen aus dem Netz `10.0.10.0/24` und die Studenten aus dem Netz `10.0.20.0/24`. Nach der Konfiguration wird der DHCP-Server mit dem Befehl `dnsmasq` gestartet.

Konfiguration der Endgeräte

Die Endgeräte werden beim Start des Kathara-Labors über die jeweiligen Startup-Dateien konfiguriert. Die Dozenten- und Studenten-Clients beziehen ihre Netzwerkkonfiguration automatisch über DHCP. Hierzu wird mit `dhclient eth0` der DHCP-Client für die Netzwerkschnittstelle `eth0` gestartet.

Die Dozenten-Clients sind mit dem Netz `10.0.10.0/24` verbunden und erhalten ihre IP-Adressen aus dem auf Router1 konfigurierten DHCP-Adressbereich `10.0.10.10-10.0.10.50`. Die Studenten-Clients befinden sich im Netz `10.0.20.0/24` und erhalten entsprechend Adressen aus dem Bereich `10.0.20.10-10.0.20.50`.

Für die Erreichbarkeit der anderen Netzwerke des Szenarios wird auf den Dozenten-Clients eine Route über Router1 mit der Gateway-Adresse `10.0.10.1` eingerichtet. Auf den Studenten-Clients wird entsprechend die Gateway-Adresse `10.0.20.1` verwendet.

Codeauszug 6.19: Netzwerkkonfiguration der Endgeräte

```
1 #Dozent1 und Dozent2
2 dhclient eth0
3 ip route add 10.0.0.0/8 via 10.0.10.1 dev eth0 || true
4 echo 'nameserver 8.8.8.8' > /etc/resolv.conf
5 apt-get update -q && apt-get install -y -q firefox-esr fonts-dejavu
6
7 #Student1 und Student2
8 dhclient eth0
9 ip route add 10.0.0.0/8 via 10.0.20.1 dev eth0 || true
10 echo 'nameserver 8.8.8.8' > /etc/resolv.conf
```

```
11 apt-get update -q && apt-get install -y -q firefox-esr fonts-dejavu
```

Für die Namensauflösung wird auf den Endgeräten der DNS-Server 8.8.8.8 in der Datei `/etc/resolv.conf` eingetragen. Außerdem werden Firefox ESR und die Schriftarten DejaVu installiert.

WebDAV-Konfiguration

Die Konfiguration des WebDAV-Dienstes erfolgt in der Datei `webdav.conf`. Diese Datei wird auf dem Host innerhalb des gerätespezifischen Verzeichnisses `webserver` abgelegt. Kathara übernimmt die dort angelegte Verzeichnisstruktur beim Start in das Dateisystem des Webserver-Containers[7]. Dadurch wird die Datei unter `/etc/apache2/sites-available/webdav.conf` im Container bereitgestellt [67].

Codeauszug 6.20: Konfiguration des WebDAV-Dientes

```
1 <VirtualHost *:80>
2 DocumentRoot /var/www/webdav
3   <Directory /var/www/webdav>
4     DAV On
5     <RequireAny>
6       Require method GET POST OPTIONS PROPFIND HEAD
7       Require ip 10.0.10.0/24
8     </RequireAny>
9   </Directory>
10 </VirtualHost>
```

Der WebDAV-Dienst wird anschließend innerhalb eines `VirtualHost` auf Port 80 konfiguriert. Die Angabe `*:80` bedeutet, dass der `VirtualHost` HTTP-Anfragen auf Port 80 für alle lokalen IP-Adressen des Webservers annimmt.

Die Direktive `DavLockDB` musste in Kathara nicht zusätzlich angegeben werden, da der Speicherort der WebDAV-Lock-Datenbank bereits über die Apache-Konfiguration in `dav_fs.conf` festgelegt war.

Mit `DocumentRoot /var/www/webdav` wird das Verzeichnis festgelegt, aus dem Apache die WebDAV-Ressourcen bereitstellt. Das Verzeichnis `/var/www/webdav` wurde für dieses Szenario separat angelegt, um die WebDAV-Ressourcen von standardmäßigen Apache-Dokumentationsverzeichnis `/var/www/html` zu trennen[40].

Innerhalb des `<Directory>`-Blocks werden die Einstellungen für das WebDAV-Verzeichnis festgelegt. Mit `DAV On` werden die WebDAV-Methoden für dieses Verzeichnis aktiviert. Dadurch können die für das Szenario benötigten WebDAV-Funktionen genutzt werden.

Die Zugriffssteuerung erfolgt über den Block `<RequireAny>`. Dabei wird der Zugriff erlaubt, wenn mindesten eine der definierten Bedingungen erfüllt ist. Die Methoden `GET`, `HEAD`,

OPTIONS und PROPFIND sind für alle Clients zugelassen. Dadurch können die Studierenden auf die bereitgestellten Dateien zugreifen und Informationen über WebDAV-Ressourcen abrufen. Die zweite Bedingung `Require ip 10.0.10.0/24` erlaubt zusätzlich Clients aus dem Dozenten-Netzwerk den Zugriff unabhängig von der verwendeten Methode. Dadurch können Dozenten auch schreibende WebDAV-Operationen wie PUT zum Hochladen und DELETE zum löschen von Dateien ausführen, während diese Operationen für Studierende nicht freigegeben sind [41, 42].

Konfiguration der startup-Datei des Webservers

Die Netzwerkkonfiguration des Webservers erfolgt beim Start über die Datei `webserver.startup`. Zunächst wird der Schnittstelle `eth0` mit `ip addr add 10.0.30.2/24 dev eth0` die statische IPv4-Adresse `10.0.30.2/24` zugewiesen. Damit ist der Webserver innerhalb des Servernetzes `10.0.30.0/24` erreichbar.

Für die Kommunikation mit den anderen Teilnetzen wird zusätzlich die Route `10.0.0.0/8 via 10.0.30.1` eingerichtet. Als Next Hop dient dabei Router 2 mit der Adresse `10.0.30.1`. Dadurch kann der Webserver antworten an die Dozenten- und Studentennetze über Router 2 weiterleiten.

Anschließend wird mit `mkdir -p /var/www/webdav` das für die WebDAV-Ressourcen vorgesehene Verzeichnis angelegt. Mit `chown -R www-data:www-data /var/www/webdav` wird dieses Verzeichnis dem Benutzer und der Gruppe `www-data` zugeordnet, unter denen der Apache-Prozess im verwendeten Container ausgeführt wird. Dadurch erhält Apache die notwendigen Zugriffsrechte, um WebDAV-Ressourcen in diesem Verzeichnis bereitzustellen und bei entsprechend autorisierten Anfragen zu verändern.

Für die WebDAV-Funktionalität werden anschließend die Apache-Module `dav` und `dav_fs` mit `a2enmod` aktiviert. Während `mod_dav` die grundlegende WebDAV-Funktionalität bereitstellt, ermöglicht `mod_dav_fs` die Verwaltung der WebDAV-Ressourcen im Dateisystem des Servers. Die standardmäßig aktivierte Apache-Site wird mit `a2dissite 000-default` deaktiviert, da für das Szenario ausschließlich die eigene WebDAV-Site verwendet werden soll. Anschließend wird die WebDAV-Konfigurationsdatei `webdav.conf` mit dem Befehl `a2ensite webdav` aktiviert [68, 69].

Es wird mit `nameserver 8.8.8.8` ein DNS-Server in `/etc/resolv.conf` eingetragen[60]. Anschließend werden die Paketinformationen aktualisiert und Wireshark für die spätere Aufzeichnung und Analyse des Netzwerkverkehrs installiert. Abschließend wird der Apache-Dienst mit `service apache2 start` gestartet.

Codeauszug 6.21: Konfiguration der Datei `webserver.startup`

```
1 #IP Adresse
2 ip addr add 10.0.30.2/24 dev eth0
3
4 #Default Route zu Router 2
```

```
5 ip route add 10.0.0.0/8 via 10.0.30.1 dev eth0 || true
6
7 mkdir -p /var/www/webdav
8 chown -R www-data:www-data /var/www/webdav
9 #WebDAV Konfiguration
10 a2enmod dav
11 a2enmod dav_fs
12 a2dissite 000-default
13 a2ensite webdav
14
15 #DNS-Server
16 echo 'nameserver 8.8.8.8' > /etc/resolv.conf
17
18 #Wireshark installieren
19 apt-get update -q && apt-get install -y -q wireshark
20
21 #Apache WebDAV starten
22 service apache2 start
```

X11-Anbindung für grafische Anwendungen

Grafische Anwendungen mittels X11

Für die grafische Nutzung von Firefox und Wireshark innerhalb der Kathara-Container wird eine X11-Anbindung zum Host-System eingerichtet. Dazu wird mit `env` die Umgebungsvariable `DISPLAY=:0` gesetzt. Zusätzlich bindet `volume` den X11-Socket `/tmp/.x11-unix` des Hosts in den jeweiligen Container ein. Dadurch können die Anwendungen im Container ausgeführt und ihre grafischen Oberflächen auf dem Host-System angezeigt werden [57].

Codeauszug 6.22: X11-Konfiguration der Kathara-Geräte

```
1 dozent1[env]="DISPLAY=:0"
2 student1[env]="DISPLAY=:0"
3 webserver[env]="DISPLAY=:0"
4
5 dozent1[volume]="/tmp/.X11-unix:/tmp/.X11-unix:rw"
6 student1[volume]="/tmp/.X11-unix:/tmp/.X11-unix:rw"
7 webserver[volume]="/tmp/.X11-unix:/tmp/.X11-unix:rw"
```

Auf den Dozenten- und Studenten-Endgeräten wird Firefox ESR für den grafischen Zugriff auf den WebDAV-Server installiert. Auf dem Webserver wird Wireshark für die spätere Aufzeichnung und Analyse des Netzwerkverkehrs eingesetzt. Die Installation erfolgt mit `apt-get` in den jeweiligen Startup-Dateien.

6.3.3 Durchführung

Überprüfung der VLAN-Trennung

Laut der erstellten Datei `lab.conf` gehören die Schnittstelle `eth0` und `eth1` den Dozenten, `eth2` und `eth3` den Studenten.

Codeauszug 6.23: Überprüfung der VLAN-Zuordnung auf Switch1

```
1 root@switch1:/# bridge vlan show
2 port vlan-id
3 eth0 10 PVID Egress Untagged
4 eth1 10 PVID Egress Untagged
5 eth2 20 PVID Egress Untagged
6 eth3 20 PVID Egress Untagged
7 eth4 10 20
```

Die Ausgabe bestätigt, dass die Endgeräte den vorgesehenen VLANs zugeordnet wurden.

OSPF-Test

Codeauszug 6.24: Überprüfung der OSPF-Nachbarschaft in Kathara

```
1 root@router1:/# vtysh
2
3 Hello, this is FRRouting (version 10.5.3).
4 Copyright 1996-2005 Kunihiro Ishiguro, et al.
5
6 router1# show ip ospf neighbor
7
8 Neighbor ID Pri State Up Time Dead Time Address Interface RXmtL RqstL DBsmL
9 2.2.2.2 1 Full/DR 7h57m38s 37.668s 10.0.1.2 eth1:10.0.1.1 0 0 0
10
11
12 root@router2:/# vtysh
13
14 Hello, this is FRRouting (version 10.5.3).
15 Copyright 1996-2005 Kunihiro Ishiguro, et al.
16
17 router2# show ip ospf neighbor
18
19 Neighbor ID Pri State Up Time Dead Time Address Interface RXmtL RqstL DBsmL
20 1.1.1.1 1 Full/Backup 7h58m12s 34.470s 10.0.1.1 eth0:10.0.1.2 0 0 0
```

Der Zustand `Full` bestätigt, dass die OSPF-Nachbarschaft erfolgreich aufgebaut wurde und beide Router ihre Routinginformationen vollständig ausgetauscht haben.

WebDAV-Dienst

Zur Überprüfung der Zugriffsrechte wurde Firefox innerhalb des Client-Containers gestartet und die IP-Adresse des Webserver aufgerufen.

Codeauszug 6.25: Starten von Firefox auf dem Client

```
1 xhost +local:
2 root@dozent1:/# firefox esr http://10.0.30.2
```



Abbildung 6.5: Grafische Darstellung von Firefox

Die Startseite des WebDAV-Servers wird im Browser angezeigt. Die angezeigte Verzeichnisübersicht bestätigt, dass der Apache-Webserver erreichbar ist und die Grafische Anwendung von Browser ordnungsgemäß funktioniert. Zur Überprüfung der Schreib- und Leseberechtigungen wurde zunächst eine Testdatei mit `tmp` im Client-Container erstellt. Anschließend wurde die Datei auf den Apache-Webserver hochgeladen, heruntergeladen und wieder gelöscht.

Codeauszug 6.26: Befehle von HTTP-Anfragen

```
1
2 root@dozent1:/# echo "Hallo, hier ist eine Testdatei fuer Dozenten." > ↵
   /tmp/testdatei.txt
3
4 root@dozent1:/# curl -T /tmp/testdatei.txt http://10.0.30.2/testdatei.txt
5
6 root@dozent1:/# curl http://10.0.30.2/testdatei.txt
7
8 root@dozent1:/# curl -X DELETE http://10.0.30.2/testdatei.txt
```

No.	Time	Source	Destination	Protocol	Length	Info
10	30.280679863	10.0.30.2	10.0.10.11	HTTP	91	HTTP/1.1 100 Continue
12	30.340110563	10.0.10.11	10.0.30.2	HTTP	112	PUT /testdatei.txt HTTP/1.1
13	30.340527589	10.0.30.2	10.0.10.11	HTTP	571	HTTP/1.1 201 Created (text/html)
26	48.221586450	10.0.10.11	10.0.30.2	HTTP	152	GET /testdatei.txt HTTP/1.1
28	48.222247646	10.0.30.2	10.0.10.11	HTTP	340	HTTP/1.1 200 OK (text/plain)
46	121.908521038	10.0.30.2	10.0.20.19	HTTP	560	HTTP/1.1 403 Forbidden (text/html)
61	149.620043331	10.0.20.19	10.0.30.2	HTTP	152	GET /testdatei.txt HTTP/1.1
63	149.620807469	10.0.30.2	10.0.20.19	HTTP	340	HTTP/1.1 200 OK (text/plain)
73	168.989748062	10.0.20.19	10.0.30.2	HTTP	155	DELETE /testdatei.txt HTTP/1.1
75	168.990500373	10.0.30.2	10.0.20.19	HTTP	541	HTTP/1.1 403 Forbidden (text/html)
88	196.569220309	10.0.10.11	10.0.30.2	HTTP	155	DELETE /testdatei.txt HTTP/1.1
90	196.571012167	10.0.30.2	10.0.10.11	HTTP	162	HTTP/1.1 204 No Content

Frame 10: 91 bytes on wire (728 bits), 91 bytes captured (728 b	0000	56 23 b9 96 f8 58 b2 1f 1e 9b 27 a5 08 00 45 00	V#...X..
▶ Ethernet II, Src: b2:1f:1e:9b:27:a5 (b2:1f:1e:9b:27:a5), Dst: 5	0010	00 4d 3f 98 40 00 40 06 bf 06 0a 00 1e 02 0a 00	.M?@.@.
▶ Internet Protocol Version 4, Src: 10.0.30.2, Dst: 10.0.10.11	0020	0a 0b 00 50 9c 94 96 a9 fa 50 ca a6 1a da 80 18	...P.....
▶ Transmission Control Protocol, Src Port: 80, Dst Port: 40084, S	0030	00 40 27 8a 00 00 01 01 08 0a 51 4c 59 af 03 0c	@.....
▶ Hypertext Transfer Protocol	0040	1b 4b 48 54 54 50 2f 31 2e 31 20 31 30 30 20 43	..KHTTP/1
	0050	6f 6e 74 69 6e 75 65 0d 0a 0d 0a	ontinue.

Abbildung 6.6: Wireshark-Mitschnitt von HTTP-Methoden

Mit der Abbildung 6.6 wird bestätigt, dass die in der Apache-Konfiguration definierten Zugriffsrechte korrekt umgesetzt wurden.

Der Dozent-client mit der IP-Adresse 10.0.10.11 sendete zunächst eine PUT-Anfrage für die Datei `testdatei.txt`. Der Webserver bestätigte die Erstellung mit 201 `Created`. Anschließend wurde die Datei mit GET abgerufen, worauf der Server mit 200 `OK` antwortete. Beim Student-Client mit der IP-Adresse 10.0.20.19 wurde dagegen die PUT- und DELETE-Anfrage mit 403 `Forbidden` abgelehnt. Bei GET-Anfragen antwortete der Webserver mit 200 `OK`.

Dies zeigt, dass die konfigurierte Zugriffsbeschränkung wirksam war.

6.3.4 Testergebnisse

Tabelle 6.6: Zusammenfassung der Testergebnisse des Szenario2

Test	Status	Ergebnis
VLAN-Zuordnung (bridge vlan show)	✓	Alle Endgeräte wurden den vorgesehenen VLANs zugeordnet.
Kommunikation innerhalb VLAN 10	✓	Ping zwischen den Clients erfolgreich.
Kommunikation innerhalb VLAN 20	✓	Ping zwischen den Clients erfolgreich.
Inter-VLAN-Routing	✓	Kommunikation zwischen VLAN 10 und VLAN 20 über Router1 erfolgreich.
OSPF-Nachbarschaft	✓	Nachbarschaft im Zustand FULL; Routinginformationen wurden erfolgreich ausgetauscht.
Erreichbarkeit des Webservers	✓	Webserver über Router2 erfolgreich erreichbar.
Grafische Anwendungen über X11	✓	Firefox und Wireshark konnten innerhalb der Container gestartet und auf dem Desktop des Host-Systems grafisch dargestellt werden.
WebDAV-Zugriff (Dozent)	✓	Datei konnte erfolgreich hochgeladen, heruntergeladen und gelöscht werden.
WebDAV-Zugriff (Student)	✓	Herunterladen möglich; Schreib- und Löschzugriffe wurden mit 403 Forbidden verweigert.

6.4 Umsetzung Szenario 3

6.4.1 Topologie

Die Topologie in Kathara besteht aus zwei lokalen Netzen. PC1 und PC2 befinden sich im Netzwerk 10.0.1.0/24, während PC3 und PC4 zum zweiten Netzwerk 10.0.2.0/24 gehören. Das erste Netz ist mit dem Router1 und das zweite Netz mit Router2 verbunden. Zwischen Router1 und Router2 existieren zwei mögliche Übertragungswege. Der Primärpfad verläuft direkt zwischen Router1 und Router2. Als alternativer Pfad dient die Verbindung

über Router3 und anschließend Router2. Dadurch können neben der Zugriffskontrolle auch das Failover und die Redundanz des Netzwerks untersucht werden.

Tabelle 6.7: Adressierung des dritten Szenarios in Kathara

Komponente	Schnittstelle	IP-Adresse	Beschreibung
Router1	eth0	10.0.1.1/24	Gateway für Netzwerk 1 mit PC1 und PC2
Router1 (Primärpfad)	eth1	10.0.12.1/30	Verbindung zwischen Router1 und Router2
Router1	eth2	10.0.13.1/30	Verbindung zu Router3
Router2	eth0	10.0.2.1/24	Gateway für Netzwerk 2 mit PC3 und PC4
Router2 (Primärpfad)	eth1	10.0.12.2/30	Verbindung des Primärpfades zu Router1
Router2	eth2	10.0.23.2/30	Verbindung des Backuppfades zu Router3
Router3	eth0	10.0.13.2/30	Verbindung des Backuppfades zu Router1
Router3 (zu Router2)	eth1	10.0.23.1/30	Verbindung des Backuppfades zu Router2
PC1		10.0.1.20 – 10.0.1.100	Multicast-Sender
PC2		10.0.1.11	Multicast-Sender
PC3		10.0.2.20 – 10.0.2.100	Multicast-Empfänger
PC4		10.0.2.11	Multicast-Empfänger
Multicast-Bereich		224.0.0.0/4	Adressbereich für den Multicast-Verkehr

6.4.2 Konfiguration

6.4.3 Konfiguration der Topologiedatei

Die Topologie des dritten Szenarios wurde in der Datei `lab.conf` erstellt. Sie besteht aus vier PCs und drei Router. Die PCs wurden mit dem Image `kathara/base` und die Router mit `kathara/frr` gestartet.

Für die Verbindung der Geräte wurden mehrere Kollisionsdomänen verwendet. `CD1` verbindet PC1 und PC2 mit Router1, während `CD2` PC3 und PC4 mit Router2 verbindet. Router1 und Router2 sind über der Kollisionsdomäne `wan` verbunden. Router1 und Router3 sind über `CD3`, während Router2 und Router3 über `CD4` verbunden sind.

Routerkonfiguration

Für das Routing zwischen den drei Routern wird FRRouting verwendet. Die IP-Adressen der Schnittstellen sowie OSPF und PIM Dense Mode werden in der Datei `/etc/frr/frr.conf` konfiguriert.

Codeauszug 6.27: Konfiguration von OSPF- und PIM-Daemon

```
1 zebra=yes
2 ospfd=yes
3 pimd=yes
```

Router1 Router1 ist direkt mit dem Netz des Multicast-Senders verbunden. Mit `ip pim dm` wird PIM Dense Mode auf den benötigten Schnittstellen aktiviert. Der Block `Router ospf` konfiguriert den dynamischen Netzwerkverkehr.

Codeauszug 6.28: Multicast-Konfiguration von Router1

```
1 frr version 10.5.3
2 frr defaults traditional
3 hostname router1
4
5 interface eth0
6     ip address 10.0.1.1/24
7     ip pim dm
8
9 interface eth1
10    ip address 10.0.12.1/30
11    ip pim dm
12
13 interface eth2
14    ip address 10.0.13.1/30
15    ip pim dm
16
17 router ospf
18     ospf router-id 1.1.1.1
19     network 10.0.1.0/24 area 0.0.0.0
20     network 10.0.12.0/30 area 0.0.0.0
21     network 10.0.13.0/30 area 0.0.0.0
```

Router2 Router2 ist hingegen mit dem Netz der Multicast-Empfänger verbunden. Deshalb wird auf `eth0` zusätzlich IGMP aktiviert. Dadurch kann Router2 die Gruppenmitgliedschaften der Empfänger im lokalen Netz erkennen.

Codeauszug 6.29: Multicast-Konfiguration von Router2

```
1 frr version 10.5.3
```

```
2 frr defaults traditional
3 hostname router2
4
5 interface eth0
6     ip address 10.0.2.1/24
7     ip pim dm
8     ip igmp
9
10 interface eth1
11     ip address 10.0.12.2/30
12     ip pim dm
13
14 interface eth2
15     ip address 10.0.23.2/30
16     ip pim dm
17
18 router ospf
19     ospf router-id 2.2.2.2
20     network 10.0.2.0/24 area 0.0.0.0
21     network 10.0.12.0/30 area 0.0.0.0
22     network 10.0.23.0/30 area 0.0.0.0
```

Router3 Router3 stellt einen zusätzlichen Pfad zwischen Router1 und Router2 bereit. Auf beiden Schnittstellen werden auch PIM Dense Mode aktiviert, damit Multicast-Verkehr auch bei Ausfall über den Pfad weitergeleitet werden kann.

Codeauszug 6.30: Multicast-Konfiguration von Router3

```
1 frr version 10.5.3
2 frr defaults traditional
3 hostname router3
4
5 interface eth0
6     ip address 10.0.13.2/30
7     ip pim dm
8
9 interface eth1
10     ip address 10.0.23.1/30
11     ip pim dm
12
13 router ospf
14     ospf router-id 3.3.3.3
15     network 10.0.13.0/30 area 0.0.0.0
16     network 10.0.23.0/30 area 0.0.0.0
```

Konfiguration der Startup-Datei von Router1

In der Datei `router1.startup` wird `dnsmasq` installiert und als DHCP-Server für das Netz `10.0.1.0/24` eingerichtet. Für einen Client mit der festgelegten MAC-Adresse wird zusätzlich die feste Adresse `10.0.1.11` reserviert. Außerdem wird in der Konfiguration zwei `iptables`-Regeln gesetzt, die den Verkehr zwischen `10.0.1.11` und `10.0.2.11` blockieren. Am Ende wird das FRRouting gestartet.

Codeauszug 6.31: Konfiguration der startup-Datei von Router1

```
1 echo 'nameserver 8.8.8.8' > /etc/resolv.conf
2 apt-get update -q && apt-get install -y -q dnsmasq
3
4 #dnsmasq fuer DHCP-Server
5 cat > /etc/dnsmasq.conf << 'EOF'
6 port=0
7 interface=eth0
8 dhcp-range=10.0.1.20,10.0.1.100,255.255.255.0,210d
9 dhcp-host=00:e0:4c:68:3e:e7,10.0.1.11
10 EOF
11 dnsmasq
12
13
14 iptables -A FORWARD -s 10.0.1.11 -d 10.0.2.11 -j DROP
15 iptables -A FORWARD -s 10.0.2.11 -d 10.0.1.11 -j DROP
16
17 service frr start
```

Konfiguration der Startup-Datei von Router2

In der Datei `router2.startup` wird `dnsmasq` installiert und als DHCP-Server für das Netz `10.0.2.0/24` eingerichtet. Für einen Client mit der festgelegten MAC-Adresse wird zusätzlich die feste Adresse `10.0.2.11` reserviert. Außerdem wird in der Konfiguration zwei `iptables`-Regeln gesetzt, die den Verkehr zwischen `10.0.1.11` und `10.0.2.11` blockieren. Am Ende wird das FRRouting gestartet.

Codeauszug 6.32: Konfiguration der startup-Datei von Router2

```
1 echo 'nameserver 8.8.8.8' > /etc/resolv.conf
2
3 apt-get update -q && apt-get install -y -q dnsmasq
4
5 #dnsmasq fuer DHCP-Server
6 cat > /etc/dnsmasq.conf << 'EOF'
7 port=0
8 interface=eth0
9 dhcp-range=10.0.2.20,10.0.2.100,255.255.255.0,210d
```

```

10 dhcp-host=00:e0:4c:68:3e:fe,10.0.2.11
11 EOF
12 dnsmasq
13
14
15 iptables -A FORWARD -s 10.0.1.11 -d 10.0.2.11 -j DROP
16 iptables -A FORWARD -s 10.0.2.11 -d 10.0.1.11 -j DROP
17
18 service frr start

```

Konfiguration der startup-Datei von Router3

Router3 übernimmt hier nur Routing- und Multicast-Funktionen. In dieser Datei wird nur FRRouting gestartet.

Codeauszug 6.33: Konfiguration der Startup-Datei von Router3

```

1 service frr start

```

6.4.4 Durchführung

Nach Abschluss der Konfiguration wurde die Funktionsfähigkeit des Netzwerks schrittweise überprüft.

Überprüfung der Firewall-Regeln

Codeauszug 6.34: Überprüfung der Kommunikation zwischen Endgeräten

```

1 root@pc2:/# ping -c 4 10.0.2.11
2 PING 10.0.2.11 (10.0.2.11) 56(84) bytes of data.
3
4 --- 10.0.2.11 ping statistics ---
5 4 packets transmitted, 0 received, 100% packet loss, time 3069ms
6
7 root@pc2:/# ping -c 4 10.0.2.92 (IP-Adresse von PC3)
8 PING 10.0.2.92 (10.0.2.92) 56(84) bytes of data.
9 64 bytes from 10.0.2.92: icmp_seq=1 ttl=62 time=2.48 ms
10 64 bytes from 10.0.2.92: icmp_seq=2 ttl=62 time=1.85 ms
11 64 bytes from 10.0.2.92: icmp_seq=3 ttl=62 time=2.20 ms
12 64 bytes from 10.0.2.92: icmp_seq=4 ttl=62 time=2.50 ms
13
14 --- 10.0.2.92 ping statistics ---
15 4 packets transmitted, 4 received, 0% packet loss, time 3004ms
16 rtt min/avg/max/mdev = 1.851/2.258/2.503/0.263 ms

```

Überprüfung des Failovers

Zur Überprüfung der Redundanz wurde der Primärpfad zwischen Router1 und Router2 gezielt unterbrochen.

Vor dem Ausfall

Codeauszug 6.35: Vor dem Ausfall

```

1
2 router1# show ip ospf neighbor
3
4 Neighbor ID Pri State Up Time Dead Time Address Interface RXmtL RqstL DBsmL
5 2.2.2.2 1 Full/DR 14m02s 32.896s 10.0.12.2 eth1:10.0.12.1 0 0 0
6 3.3.3.3 1 Full/DR 14m04s 30.369s 10.0.13.2 eth2:10.0.13.1 0 0 0
7
8 router1# show ip route ospf
9 Codes: K - kernel route, C - connected, L - local, S - static,
10 R - RIP, O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
11 T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
12 f - OpenFabric, t - Table-Direct,
13 > - selected route, * - FIB route, q - queued, r - rejected, b - backup
14 t - trapped, o - offload failure
15
16 IPv4 unicast VRF default:
17 0 10.0.1.0/24 [110/10] is directly connected, eth0, weight 1, 00:17:29
18 O>* 10.0.2.0/24 [110/20] via 10.0.12.2, eth1, weight 1, 00:16:39
19 0 10.0.12.0/30 [110/10] is directly connected, eth1, weight 1, 00:16:44
20 0 10.0.13.0/30 [110/10] is directly connected, eth2, weight 1, 00:16:47
21 O>* 10.0.23.0/30 [110/20] via 10.0.12.2, eth1, weight 1, 00:16:39
22 * via 10.0.13.2, eth2, weight 1, 00:16:39

```

Nach dem Ausfall

Codeauszug 6.36: Nach dem Ausfall

```

1 router1# conf ter
2 router1(config)# interface eth1
3 router1(config-if)# shutdown
4 router1(config-if)# exit
5 router1(config)# exit
6 router1# show ip ospf neighbor
7
8 Neighbor ID Pri State Up Time Dead Time Address Interface RXmtL RqstL DBsmL
9 3.3.3.3 1 Full/DR 7h27m01s 34.539s 10.0.13.2 eth2:10.0.13.1 0 0 0
10
11 router1# show ip route ospf
12 Codes: K - kernel route, C - connected, L - local, S - static,
13 R - RIP, O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,

```

```

14 T - Table, v - VNC, V - VNC-Direct, A - Babel, F - PBR,
15 f - OpenFabric, t - Table-Direct,
16 > - selected route, * - FIB route, q - queued, r - rejected, b - backup
17 t - trapped, o - offload failure
18
19 IPv4 unicast VRF default:
20 0 10.0.1.0/24 [110/10] is directly connected, eth0, weight 1, 07:27:59
21 0>* 10.0.2.0/24 [110/30] via 10.0.13.2, eth2, weight 1, 00:01:59
22 0>* 10.0.12.0/30 [110/30] via 10.0.13.2, eth2, weight 1, 00:01:59
23 0 10.0.13.0/30 [110/10] is directly connected, eth2, weight 1, 07:27:17
24 0> 10.0.23.0/30 [110/20] via 10.0.12.2, eth1 inactive, weight 1, 00:01:59
25 * via 10.0.13.2, eth2, weight 1, 00:01:59

```

Nach der Deaktivierung des Primärpfades wurde das Netzwerk 10.0.2.0/24 nicht mehr über Router2 (10.0.12.2), sondern über Router3 (10.0.13.2) erreicht. Gleichzeitig erhöhte sich die OSPF-Metrik von 20 auf 30. Die OSPF-Nachbarschaft zu Router2 verschwand, während die Nachbarschaft zu Router3 weiterhin den Zustand FULL aufwies. Damit konnte die automatische Umschaltung auf den Backup-Pfad nachgewiesen werden.

Überprüfung der Multicast-Kommunikation

Zur Überprüfung der Multicast-Weiterleitung wurde zunächst `iperf` verwendet. PC1 diene dabei als Sender und PC4 als Empfänger. Als Multicast-Gruppe wurde die Adresse 239.1.2.3 mit dem UDP-Port 1234 verwendet.

Codeauszug 6.37: Start des Multicast-Empfängers auf PC4

```

1 iperf -s -B 239.1.2.3 -u -p 1234 -i 1

```

Codeauszug 6.38: Erzeugung des Multicast-Datenverkehrs auf PC1

```

1 iperf -c 239.1.2.3 -u -b 1M -t 30 -T 11 -p 1234

```

Der Parameter `-T 11` setzt den TTL-Wert der Multicast-Pakete auf 11. Dies ist erforderlich, da der Datenverkehr mehrere Router durchläuft. Bei einem TTL-Wert von 1 würde das Multicast-Paket bereits nach dem ersten Router verworfen werden.

Codeauszug 6.39: Kontrolle der Multicast-Weiterleitung

```

1 router1# show ip mroute
2 IP Multicast Routing Table
3 Flags: S - Sparse, D - Dense, C - Connected, P - Pruned
4 R - SGRpt Pruned, F - Register flag, T - SPT-bit set
5 Source Group Flags Proto Input Output TTL Uptime
6 10.0.1.47 239.1.2.3 TD eth0 eth1 1 14:56:23
7
8

```

```

9 router1# show ip pim state
10 Codes: J -> Pim Join, I -> IGMP Report, S -> Source, * -> Inherited from ←
    (*,G), V -> VxLAN, M -> Muted
11 Active Source Group RPT IIF OIL
12 1 10.0.1.47 239.1.2.3 n eth0 eth1( ), eth2( )
13
14 router2# show ip mroute
15 IP Multicast Routing Table
16 Flags: S - Sparse, D - Dense, C - Connected, P - Pruned
17 R - SGRpt Pruned, F - Register flag, T - SPT-bit set
18 Source Group Flags Proto Input Output TTL Uptime
19 10.0.1.47 239.1.2.3 TD eth1 eth0 1 00:00:26
20
21
22 router2# show ip igmp groups
23 Total IGMP groups: 3
24 Watermark warn limit(Not Set): 0
25 Interface Group Mode Timer Srcs V Uptime
26 eth0 224.0.1.39 EXCL 00:03:03 1 3 22:35:37
27 eth0 224.0.1.40 EXCL 00:03:03 1 3 22:35:37
28 eth0 239.1.2.3 EXCL 00:03:17 1 3 00:01:08
29
30 router2# show ip pim state
31 Codes: J -> Pim Join, I -> IGMP Report, S -> Source, * -> Inherited from ←
    (*,G), V -> VxLAN, M -> Muted
32 Active Source Group RPT IIF OIL
33 0 * 224.0.1.39 y <iif?> pimreg(I ), eth0(I )
34 0 * 224.0.1.40 y <iif?> pimreg(I ), eth0(I )
35 0 * 239.1.2.3 y <iif?> pimreg(I ), eth0(I )
36 1 10.0.1.47 239.1.2.3 n eth1 eth0( ), eth2( )
37
38 router2# show ip rpf 10.0.1.47
39 (10.0.1.47, *) --- Nexthop 10.0.12.1 Interface eth1

```

Die Multicast-Routing-Tabelle von Router1 zeigt einen (S,G)-Eintrag für die Quelle 10.0.1.47 und die Multicast-Gruppe 239.1.2.3. Der Datenverkehr wird über Ethernet0 empfangen und über Ethernet1 weitergeleitet. Auf Router2 ist ebenfalls ein (S,G)-Eintrag für 10.0.1.47 und 239.1.2.3 vorhanden. Zusätzlich zeigt die IGMP-Tabelle, dass sich die Empfänger PC3 und PC4, die über das eth0 von Router2 verbunden sind, erfolgreich für die Multicast-Gruppe 239.1.2.3 registriert haben.

6.4.5 Testergebnisse

Tabelle 6.8: Testergebnisse von Szenario 3

Test	Status	Beobachtung
Netzwerkcommunication	✓	Kommunikation zwischen den beiden Netzen Über OSPF möglich.
Firewall-Filterung	✓	Kommunikation zwischen PC2 und PC4 erfolgreich blockiert.
OSPF-Failover	✓	Nach Ausfall des Primärpfades erfolgte die Umschaltung über Router3.
Firewall nach Failover	✓	Die Filterung zwischen PC2 und PC4 blieb auch auf dem Backup-Pfad wirksam.
Multicast-Routing	✓	Der (S,G)-Eintrag wurde aufgebaut und der Multicast-Verkehr weitergeleitet.
Multicast-Datenübertragung	✓	Die UDP-Multicast-Übertragung mit <code>iperf</code> wurde erfolgreich nachgewiesen.

Kapitel 7: Vergleich und Evaluation der drei Umgebungen

7.1 Benutzerfreundlichkeit und Implementierungsaufwand

Die Installationsschritte sind bei beiden Umgebungen Kathara und Containerlab gut dokumentiert und konnten ohne größere Schwierigkeiten durchgeführt werden. Größere Unterschiede zeigten sich dagegen bei der Erstellung und Konfiguration der Topologien.

Bei Containerlab wurde die Topologie über eine YAML-Datei erstellt. Die Erstellung dieser Datei war teilweise Zeitaufwendig, da kleiner Fehler, besonders bei der Einrückung, zu Fehlermeldungen führen konnten. Dadurch wurde während der Umsetzung mehr Zeit für die Fehlersuche benötigt. Auch Fehler in der Konfiguration einzelner Geräte konnte den Start des Labs beeinflussen. Der Startvorgang blieb wegen einer Fehlermeldung hängen bzw. wurde nicht vollständig abgeschlossen. Ein Beispiel dafür war eine fehlerhafte Konfiguration des DHCP-Servers. In anderen Fällen wurde das Lab trotz einer Fehlermeldung gestartet.

Ein Vorteil von Containerlab ist die Unterstützung in Visual Studio Code. Dort kann die aufgebaute Topologie grafisch dargestellt werden und die Container einfacher verwaltet werden, sodass nicht alle Schritte nur über die Kommandozeile durchgeführt werden müssen. Außerdem ist die Dokumentation von Containerlab sehr ausführlich und enthält viele kleine Beispiele für die verschiedenen Funktionen.

Bei Kathara war die Erstellung der Topologie mit der `lab.conf`-Datei einfacher. Hilfreich waren dabei auch die von der Kathara-Community auf Github bereitgestellten Beispiel-Labore, die als Orientierung für den eigenen Aufbau verwendet werden konnten. Während der Umsetzung traten bei der Topologiedateien weniger Fehler auf als bei Containerlab. Eine grafische Oberfläche wie bei der Arbeit mit Containerlab in Visual Studio Code steht jedoch nicht zur Verfügung. Die Bedienung erfolgt daher hauptsächlich über die Kommandozeile. Auch die Dokumentation von Kathara bietet ausreichend Informationen für die Erstellung eigener Netzwerklabore.

7.2 Ressourcenverbrauch

Die Ergebnisse in der Tabelle 7.1 zeigen deutliche Unterschiede beim Ressourcenverbrauch. Besonders bei den Routern und den Switches war der Verbrauch in Containerlab höher. Router1 benötigte beispielsweise etwa 1124 MiB RAM, während der gleiche Knoten in Kathara nur etwa 23 MiB verwendete. Auch bei der CPU-Auslastung waren bei den Netzwerkgeräten deutliche Unterschiede zu erkennen.

Die Unterschiedlichen Werte können unter anderem durch die verwendeten Images erklärt werden. In Containerlab wurden für die Router und den Switch cEOS-Images eingesetzt, während in Kathara leichtgewichtiger Container verwendet wurden. Die Ergebnisse beziehen sich deshalb auf die in dieser Arbeit verwendeten Konfigurationen und Images. Ein allgemeiner Rückschluss, dass Containerlab grundsätzlich mehr Ressourcen als Kathara benötigt, kann daraus nicht gezogen werden.

Tabelle 7.1: Ressourcenverbrauch in Kathara und Containerlab

Knoten	Umgebung	CPU[%]	RAM [MiB]
Router1	Kathara	0.02	23.08
Router1	Containerlab	35.91	1124
Switch1	Kathara	0.00	1.69
Switch1	Containerlab	43.50	1085
Student1	Kathara	0.00	19.07
Student1	Containerlab	0.00	27.64

Ein weiterer Unterschied zeigte sich beim Starten der Labore. Bei Containerlab dauerte der Start von Szenario 2 ungefähr drei Minuten, während das gleiche Szenario in Kathara bereits nach ungefähr einer Minute gestartet war. Dieser Unterschied war auch bei den anderen Szenarien deutlich bemerkbar.

7.3 Realitätsnahe der Simulationsumgebungen

Durch die Umsetzung der gleichen Szenarien im physischen Cisco-Labor konnte untersucht werden, wie realitätsnah Kathara und Containerlab die verwendeten Netzwerkfunktionen abbilden. Grundlegende Funktionen wie Routing, VLANs, Firewall-Regeln, Failover und Multicast konnten in beiden Emulatoren umgesetzt und getestet werden.

Containerlab kam dem physischen Cisco-Labor besonders bei der Konfiguration der Netzwerkgeräte näher. Durch die Verwendung von cEOS konnten viele Konfigurationen und Show-Befehle ähnlich wie auf den realen Cisco-Geräten verwendet werden. Unterschiede zeigten sich hauptsächlich bei den Bezeichnungen und der Zuordnung der Interfaces.

Bei Kathara wurden die Router dagegen mit FRRouting und die Netzwerkfunktionen teilweise mit Linux-Befehlen konfiguriert. Dadurch unterschied sich die Konfiguration stärker von den physischen Cisco-Geräten. Das grundlegende Netzwerkverhalten konnte trotzdem nachgebildet werden. Dies zeigte sich beispielsweise beim OSPF-Routing, beim Failover und bei der Multicast-Kommunikation.

Ein deutlicher Unterschied zum physischen Labor bestand bei der Hardware. Dort mussten Router, Switches und PCs tatsächlich verkabelt und die richtigen physischen Interfaces verwendet werden. In Kathara und Containerlab wurden die Verbindungen dagegen virtuell über die jeweilige Topologiedatei erstellt.

7.4 Stärken und Schwächen im Vergleich

Aus der praktischen Umsetzung der drei Szenarien konnten verschiedene Stärken und Schwächen der untersuchten Umgebungen festgestellt werden. Tabelle 7.2 fasst die wichtigsten Beobachtungen zusammen.

Tabelle 7.2: Stärken und Schwächen der untersuchten Umgebungen

Umgebung	Stärken	Schwächen
Kathara	- Keine Abhängigkeiten von Laboröffnungszeiten. - Geringer Ressourcenverbrauch. - Mehrfache Nutzung derselben Topologiedatei nach dem Beenden und erneuten Starten des Labs. - Automatische Überprüfung von Netzwerkszenarien mit Hilfe des Werkzeuges Kathará Labor Checker, besonders hilfreich für die automatische Korrektur von Prüfungen und Netzwerkszenariokonfigurationen der Studenten.[70]	- Keine grafische Darstellung der Topologie. - Konfiguration teilweise mit Linux und FRRouting und dadurch weniger ähnlich zu physischen Cisco-Geräten.
Containerlab	- Keine Abhängigkeiten von Laboröffnungszeiten. - Mehrfache Nutzung derselben Topologiedatei nach dem Beenden und erneuten Starten des Labs. - Grafische Darstellung der Topologie und Verwaltung der Container in Visual Studio Code möglich, ausführliche Dokumentation. - Verwendung von unterschiedlichen Netzwerkbetriebssystem wie beispielsweise Arista cEOS möglich.	- Höherer Ressourcenverbrauch der verwendeten cEOS-Geräte. - Längere Startzeit des untersuchten Szenarios.
Physisches Cisco-Labor	- Arbeit mit realen Routern und Switches. - Reale Verkabelung möglich. - Direkte praktische Erfahrung mit realen Netzwerkgeräten.	- Abhängigkeit von den Laboröffnungszeiten. - Höherer Zeitaufwand beim erneuten Aufbau und bei der Verkabelung der Topologie, da die Geräte nach dem Verlassen des Labors wieder umgebaut wurden.

Die drei Umgebungen besitzen damit unterschiedliche Vorteile. Kathara zeigte sich bei der Umsetzung als ressourcenschonend und vergleichsweise einfacher zu konfigurieren. Containerlab benötigte mit den verwendeten cEOS-Images mehr Ressourcen, bot dafür aber eine Konfiguration, die näher an den physischen Cisco-Geräten lag. Ein gespeichertes virtuelles Lab kann mehrmals erneut gestartet werden, während das physische Netzwerk nach einem

Umbau erneut verkabelt werden muss.

Kapitel 8: Fazit

Ziel dieser Bachelorarbeit war, die beiden Netzwerkvirtualisierungsumgebungen Kathara und Containerlab zu untersuchen und miteinander zu vergleichen. Im Mittelpunkt standen dabei die Realitätsnähe, die Benutzerfreundlichkeit, die Performance sowie die praktische Nutzbarkeit der beiden Umgebungen.

Die erste Forschungsfrage beschäftigte sich mit der Realitätsnähe der beiden Simulationsumgebungen. Die durchgeführten Szenarien zeigten, dass grundlegende Netzwerkfunktionen wie Routing, VLANs, Firewall-Regeln, Failover und Multicast sowohl mit Kathara als auch mit Containerlab umgesetzt werden konnten. Containerlab kam durch die Verwendung von cEOS bei der Konfiguration der Netzwerkgeräte näher an die physischen Cisco-Geräte heran. Bei Kathara wurde dagegen teilweise Linux und FRRouting verwendet. Die grundlegende Funktionsweise der untersuchten Netzwerke konnte trotzdem in beiden Umgebungen nachgebildet werden.

Auch hinsichtlich Benutzerfreundlichkeit und Performance zeigten sich Unterschiede. Die Erstellung der Topologie war mit Kathara vergleichsweise einfacher und der Ressourcenverbrauch war in den durchgeführten Versuchen gering. Bei Containerlab nahm insbesondere die Fehlersuche in den YAML-Topologiedateien teilweise mehr Zeit in Anspruch. Potenzielle Fehler in den YAML-Dateien, wie beispielsweise falsche Einrückungen, und falsche Parameter können durch die Verwendung der VS Code Plugins schneller erkannt werden. Einige Fehler, wie nicht in der Topologie vorhandene Knoten, doppelte Endpoints oder falsche Images, werden jedoch erst beim Deployment durch Containerlab erkannt. Die Erweiterung kann die Fehlersuche erleichtern, ersetzt aber nicht die Prüfung beim Deployment. Außerdem bietet Containerlab eine grafische Darstellung der aufgebauten Topologie in Visual Studio Code und ermöglicht die Verwendung unterschiedlicher echte Netzwerkbetriebssysteme. Die Messung des Ressourcenverbrauchs zeigte bei den verwendeten cEOS-Geräten einen deutlich höheren CPU- und RAM-Verbrauch als bei den in Kathara verwendeten Geräten. Diese Ergebnisse sind jedoch von den verwendeten Images abhängig.

Die Aufzeichnung und Untersuchung des Datenverkehrs mit Wireshark war ebenfalls in den simulierten Netzwerken möglich. Dadurch konnten Pakete an unterschiedlichen Stellen der aufgebauten Topologien beobachtet und für die Überprüfung der Kommunikation verwendet werden.

Neben den eigentlichen Netzwerkfunktionen konnten auch konkrete Anwendungen innerhalb der Container ausgeführt werden. In den Versuchen wurden sowohl ein Webserver als auch ein Browser innerhalb der Container gestartet und verwendet.

Für den selbständigen Einsatz in der Hochschullehre bieten beide Umgebungen Vorteile. Kathara eignet sich besonders für einen einfachen ressourcenschonenden Einstieg im Netzwerk. Containerlab bietet dagegen Vorteile, wenn die Arbeit mit unterschiedlichen echten Netzwerkbetriebssystemen und eine stärkere Annäherung an die Konfiguration realer Netzwerkgeräte im Vordergrund stehen. Beide Umgebungen ermöglichen es den Studierenden, Versuche unabhängig von den Öffnungszeiten eines physischen Labors durchzuführen und bereits erstellte Topologien erneut zu verwenden.

Zusammenfassend lässt sich daher keine der beiden Umgebungen allgemein als die bessere Lösung bewerten. Die Wahl hängt von jeweiligen Einsatzzweck ab. Kathara bietet Vorteile bei einer einfachen und ressourcenschonenden Umsetzung, während Containerlab insbesondere durch die Unterstützung verschiedene echte Netzwerkbetriebssysteme und die Nähe zu realen Netzwerkgeräte mehr Vorteile bietet. Beide können ein physisches Netzwerklabor sinnvoll ergänzen, da sie die Arbeit mit realer Hardware, beispielsweise die Verkabelung und der Umgang mit physischen Interfaces jedoch nicht vollständig ersetzen können.

Anlage

Die beigefügte ZIP-Datei enthält ergänzende Materialien, zu den in dieser Arbeit untersuchten Netzwerkszenarien. Sie umfasst unter anderem:

- Konfigurationsdateien der Router und Switches in Containerlab
- Konfigurationsdateien der Router, Switches, Endgeräte und des Webserver in Kathara

Diese bereitgestellten Dateien dienen als praktische Ergänzung zur Arbeit und unterstützen das Verständnis der durchgeführten Tests.

Abbildungsverzeichnis

4.1	Netzwerktopologie des ersten Szenarios im Labor	13
4.2	Netzwerktopologie des zweiten Szenarios im Labor	18
4.3	Wireshark-Mitschnitt von HTTP-Methoden	24
4.4	Netzwerktopologie des dritten Szenarios im Labor	26
5.1	Netzwerktopologie des ersten Containerlab-Szenarios	35
5.2	Erfolgreicher Start der Containerlab-Topologie	41
5.3	Aufzeichnung des Netzwerkverkehrs mit Wireshark	44
5.4	Netzwerktopologie des zweiten Containerlab-Szenarios	46
5.5	Grafische Darstellung von Firefox	53
5.6	Wireshark-Mitschnitt von HTTP-Methoden	54
5.7	Containerlab-Topologie des dritten Szenarios	56
5.8	Offizielle Bestätigung - Arista Community	58
5.9	Graphische Benutzeroberfläche von Containerlab	66
5.10	Live-Paketaufzeichnung an Router2 in Szenario1 über die VS Code-Erweiterung	67
6.1	Terminal-Emulator xterm	68
6.2	Erfolgreicher Start des Kathara-Netzwerkszenarios	75
6.3	Empfehlung vom Kathara-Team	78
6.4	Aufzeichnung des Netzwerkverkehrs mit Wireshark auf dem Container PC1	79
6.5	Grafische Darstellung von Firefox	90
6.6	Wireshark-Mitschnitt von HTTP-Methoden	91

Tabellenverzeichnis

4.1	Verwendete Hardware und Software im physischen Cisco-Labor	12
4.2	Adressierung der Geräte im physischen Labor für Szenario 1	13
4.3	Aufbau des Hexadezimalwertes der DHCP-Option 121	15
4.4	Vergebene IP-Adressen in Szenario 1	16
4.5	Ergebnisse der Ping-Tests von PC1	16
4.6		17
4.7	Adressierung des zweiten Containerlab-Szenarios	19
4.8	Zusammenfassung der Testergebnisse des Szenario2	25
4.9	Adressierung des dritten Szenarios	27
4.10	Testergebnisse von Szenario 3	31
5.1	Verwendete Software für die Containerlab-Testumgebung	34
5.2	Adressierung der Geräte in Containerlab – Szenario 1	35
5.3	Bestandteile der Containerlab-Topologie	36
5.4	Wesentliche Konfigurationsparameter der Linux-Endgeräte	40
5.5	Adressierung des zweiten Containerlab-Szenarios	46
5.6	Zusammenfassung der Testergebnisse des Szenario2	55
5.7	Adressierung des dritten Containerlab-Szenarios	57
5.8	Testergebnisse von Szenario 3	65
6.1	Systemumgebung für Kathara	69
6.2	Adressierung der Geräte in Kathara – Szenario 1	70
6.3	Verwendete Geräteoptionen in der <code>lab.conf</code>	72
6.4	Ergebnisse der iPerf3-Durchsatzmessung mit unterschiedlichen MSS-Werten	77
6.5	Adressierung des zweiten Kathara-Szenarios	80
6.6	Zusammenfassung der Testergebnisse des Szenario2	92
6.7	Adressierung des dritten Szenarios in Kathara	93
6.8	Testergebnisse von Szenario 3	101
7.1	Ressourcenverbrauch in Kathara und Containerlab	103
7.2	Stärken und Schwächen der untersuchten Umgebungen	105

Quelltextverzeichnis

4.1	Interface-Konfiguration der Router in Szenario 1	14
4.2	Konfiguration der statistischen Routen	14
4.3	DHCP-Konfiguration auf der Router	14
4.4	Durchsatzmessung zwischen PC1 und PC3	17
4.5	Konfiguration auf Router1	19
4.6	DHCP-Konfiguration auf Router1	20
4.7	Konfiguration auf Router2	20
4.8	VLAN-Konfiguration auf Switch1	21
4.9	Netzwerkkonfiguration des physischen Webservers	22
4.10	Vorbereitung des WebDAV-Servers im physischen Labor	22
4.11	WebDAV-Konfiguration im physischen Labor	23
4.12	OSPF-Konfiguration auf Router1	27
4.13	ACL-Konfiguration auf Router1	28
4.14	ACL-Konfiguration auf Router2	28
4.15	PIM-Konfiguration auf Router1	29
5.1	Download des Docker-Installationsskripts	32
5.2	Ausführen des Docker-Installationsskripts	32
5.3	Überprüfung der Docker-Installation	32
5.4	Download und Installation von Containerlab	33
5.5	Überprüfung der Containerlab-Installation	33
5.6	Import des Arista-cEOS-Images	33
5.7	Überprüfung des importierten Images	33
5.8	Grundstruktur der Containerlab-Topologiedatei	36
5.9	Beispiel der definition eines Gerätes in Containerlab	37
5.10	Definition der Netzwerkverbindungen	38
5.11	Konfiguration von PC1	39
5.12	Das Starten des Labs	40
5.13	DHCP-Adressvergabe auf PC1	41
5.14	Ping-Test von PC1 zu PC4	42
5.15	Traceroute von PC1 nach PC4	42
5.16	Aufzeichnung des Netzwerkverkehrs auf dem Container PC1	44
5.17	Auszug aus der Konfiguration von Router1	47
5.18	Auszug aus der VLAN-Konfiguration von Switch1	48
5.19	Auszug aus der VLAN-Konfiguration des Webserver-Containers	48

5.20	Auszug aus der WebDAV-Konfigurationsdatei	49
5.21	Konfiguration des X11-Forwardings auf dem Docker-Container	51
5.22	Überprüfung der VLAN-Zuordnung auf Switch1	51
5.23	Überprüfung der OSPF-Nachbarschaft auf Router1	52
5.24	Starten von Firefox auf dem Client	52
5.25	Befehle von HTTP-Anfragen	53
5.26	ACL Warnung	57
5.27	Konfiguration der transparenten Firewall in Containerlab	59
5.28	Multicast-Konfiguration am Beispiel von Router1	59
5.29	Konfiguration der globalen Aktivierung von IGMP Snooping auf den Switches	60
5.30	Aktivierung von IGMP Snooping auf den Switches	60
5.31	Überprüfung der Kommunikation zwischen Endgeräten	61
5.32	Vor dem Ausfall	61
5.33	Nach dem Ausfall	62
5.34	Start des Multicast-Empfängers auf PC3	62
5.35	Erzeugung des Multicast-Datenverkehrs auf PC1	62
5.36	Kontrolle der Multicast-Weiterleitung	62
6.1	Installation von Kathara unter Ubuntu	69
6.2	Definition der Netzwerkverbindungen in der Datei lab.conf	70
6.3	Zuordnung des Docker-Images	71
6.4	Starten des Kathara-Labs	74
6.5	Überprüfung der IP-Adresse auf PC1	75
6.6	Ping-Test zwischen PC1 und PC2	76
6.7	Ping-Test zwischen PC1 und PC3	76
6.8	Überprüfung des Routing-Pfades zwischen PC1 und PC3	76
6.9	Starten des iperf3-Servers auf PC3	77
6.10	Starten des iperf3-Client auf PC1	77
6.11	Definition der Netzwerkverbindungen	80
6.12	Konfiguration der Access-Ports und den Trunk-Port	81
6.13	Erstellung der VLAN-Subinterfaces auf Router1	82
6.14	Startup-Konfiguration von Router2	82
6.15	OSPF-Konfiguration von Router1	83
6.16	OSPF-Konfiguration von Router2	83
6.17	Aktivierte FRRouting-Daemons	84
6.18	DHCP-Konfiguration auf Router1	85
6.19	Netzwerkkonfiguration der Endgeräte	85
6.20	Konfiguration des WebDAV-Dientes	86
6.21	Konfiguration der Datei webserver.startup	87
6.22	X11-Konfiguration der Kathara-Geräte	88
6.23	Überprüfung der VLAN-Zuordnung auf Switch1	89
6.24	Überprüfung der OSPF-Nachbarschaft in Kathara	89
6.25	Starten von Firefox auf dem Client	90

6.26	Befehle von HTTP-Anfragen	90
6.27	Konfiguration von OSPF- und PIM-Daemon	94
6.28	Multicast-Konfiguration von Router1	94
6.29	Multicast-Konfiguration von Router2	94
6.30	Multicast-Konfiguration von Router3	95
6.31	Konfiguration der startup-Datei von Router1	96
6.32	Konfiguration der startup-Datei von Router2	96
6.33	Konfiguration der Startup-Datei von Router3	97
6.34	Überprüfung der Kommunikation zwischen Endgeräten	97
6.35	Vor dem Ausfall	98
6.36	Nach dem Ausfall	98
6.37	Start des Multicast-Empfängers auf PC4	99
6.38	Erzeugung des Multicast-Datenverkehrs auf PC1	99
6.39	Kontrolle der Multicast-Weiterleitung	99

Literaturverzeichnis

- [1] *Docker-Engine*. URL: <https://docs.docker.com/engine/network/>.
- [2] *Container*. URL: <https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/>.
- [3] *Container-Image*. URL: <https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-an-image/>.
- [4] *DockerHub*. URL: https://hub.docker.com/?_gl=1*495rdd*_gcl_au*MTI2NjA1OTgwNy4xNzg1NDIzNTc5*_ga*MTIyNjEwMTI4OS4xNzg1NDIzNTc4*_ga_XJWPQMjYHQ*cZ30Dc4NjI3NTMkbzIyJGcxJHQxNzg3ODYyODYzJGozNSRsMCRoMA...
- [5] *YAML-Language*. URL: <https://yaml.org/spec/1.2.2/>.
- [6] *Containerlab Topology Definition*. URL: <https://containerlab.dev/manual/topo-def-file/>.
- [7] *Kathara-lab-dirs*. URL: <https://www.kathara.org/man-pages/kathara-lab-dirs.7.html>.
- [8] *DHCP*. URL: <https://datatracker.ietf.org/doc/html/rfc2131#section-3>.
- [9] *DHCP Option 121 rfc3442*. URL: <https://datatracker.ietf.org/doc/rfc3442/>.
- [10] *RFC 4918 : WebDAV*. URL: <https://www.rfc-editor.org/info/rfc4918/#section-1>.
- [11] *Topology Definition*. URL: <https://containerlab.dev/manual/topo-def-file/>.
- [12] *Kinds*. URL: <https://containerlab.dev/manual/kinds/srl/>.
- [13] *Kathara manpages*. URL: <https://www.kathara.org/man-pages/kathara.1.html>.
- [14] *Docker Images Kathara*. URL: <https://github.com/KatharaFramework/Docker-Images>.
- [15] *Cisco Modeling Labs - Free*. URL: <https://developer.cisco.com/docs/modeling-labs/cml-free/>.
- [16] *Arista Portal*. URL: <https://www.arista.com/en/support/software-download>.
- [17] *Kathara Framework*. URL: <https://github.com/KatharaFramework/Docker-Images>.
- [18] *FRRouting User Guide*. URL: <https://docs.frrouting.org/en/latest/>.

- [19] *IGMP Snooping Cisco*. URL: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2960x/software/15_2_6_e/configuration_guide/b_1526e_consolidated_2960x_cg/b_1526e_consolidated_2960x_cg_chapter_01010.html.
- [20] *Install Docker Engine on Ubuntu*. URL: <https://docs.docker.com/engine/install/ubuntu/>.
- [21] *Install Containerlab*. URL: <https://containerlab.dev/install/>.
- [22] *Arista cEOS: Container Image*. URL: <https://containerlab.dev/manual/kinds/ceos/>.
- [23] *Arista cEOS - Schnittstelle*. URL: <https://containerlab.dev/manual/kinds/ceos/#additional-interface-naming-considerations>.
- [24] *Network-Multitool*. URL: <https://github.com/srl-labs/network-multitool>.
- [25] *Network-Multitool - Container Image for Network Testing*. URL: <https://github.com/wbitt/Network-MultiTool>.
- [26] *Containerlab Configuration Management*. URL: <https://containerlab.dev/manual/config-mgmt/>.
- [27] *Containerlab deploy Command*. URL: <https://containerlab.dev/cmd/deploy/>.
- [28] *MTU Settings*. URL: <https://oneuptime.com/blog/post/2026-02-08-how-to-configure-docker-containers-with-custom-mtu-settings/view>.
- [29] *xhost Manual Page*. URL: <https://manpages.ubuntu.com/manpages/resolute/man1/xhost.1.html>.
- [30] *Docker exec command*. URL: <https://docs.docker.com/reference/cli/docker/container/exec/>.
- [31] *Wireshark auf dem Host-Containerlab*. URL: <https://containerlab.dev/manual/wireshark/#local-capture>.
- [32] *Arista vEOS Router Subinterface Support*. URL: <https://www.arista.com/en/cg-veos-router/veos-router-subinterface-support>.
- [33] *Arista EOS Virtual LANs*. URL: <https://www.arista.com/en/um-eos/eos-virtual-lans-vlans>.
- [34] *X11 Passthrough*. URL: <https://briefcase.beeware.org/en/v0.3.21/how-to/internal/x11passthrough.html>.
- [35] *Docker httpd*. URL: <https://github.com/docker-library/httpd/blob/4da43810a4429c552e19c43f8e513248c34cfc84/2.4/Dockerfile>.
- [36] *Apache Module mod-dav*. URL: https://httpd.apache.org/docs/current/mod/mod_dav.html.
- [37] *Apache Modul mod-dav-fs*. URL: https://httpd.apache.org/docs/current/mod/mod_dav_fs.html.

- [38] *DavLockDB*. URL: https://httpd.apache.org/docs/2.4/mod/mod_dav_fs.html#davlockdb.
- [39] *Configuring WebDAV on IBM*. URL: <https://www.ibm.com/docs/en/cognos-analytics/12.1.x?topic=server-configuring-webdav-http-apache-http>.
- [40] *Apache-Documentroot*. URL: <https://httpd.apache.org/docs/2.4/mod/core.html#documentroot>.
- [41] *Apache require method*. URL: https://httpd.apache.org/docs/2.4/mod/mod_authz_core.html#requireany.
- [42] *Apache require*. URL: https://httpd.apache.org/docs/2.4/mod/mod_authz_host.html.
- [43] *Access Control Apache*. URL: <https://httpd.apache.org/docs/current/howto/access.html>.
- [44] *The x.Org Foundation*. URL: <https://www.x.org/>.
- [45] *Running GUI Applications in Docker*. URL: <https://oneuptime.com/blog/post/2026-01-16-docker-gui-apps-x11-vnc/view>.
- [46] *Arista Community:cEOS*. URL: <https://arista.my.site.com/AristaCommunity/s/topic/0T02I000000DacxWAC/ceos>.
- [47] *Transparente Linux Bridge*. URL: <https://oneuptime.com/blog/post/2026-03-20-transparent-bridge-no-ip-linux/view>.
- [48] *ip-Link Manual Page*. URL: <https://man7.org/linux/man-pages/man8/ip-link.8.html>.
- [49] *iptables Linux Manual Page*. URL: <https://man7.org/linux/man-pages/man8/iptables.8.html>.
- [50] *Linux Ethernet Bridging Documentation*. URL: <https://docs.kernel.org/networking/bridge.html>.
- [51] *Bridge Netfilter Documentation*. URL: <https://ebtables.netfilter.org/documentation/bridge-nf.html>.
- [52] *Arista EOS Protocol Independent Multicast*. URL: <https://www.arista.com/en/um-eos/eos-protocol-independent-multicast>.
- [53] *Arista EOS Multicast Architekture*. URL: https://www.arista.com/en/um-eos/eos-multicast-architecture#xxz_VCTmgf5Slo.
- [54] *Arista Community cEOS*. URL: <https://arista.my.site.com/AristaCommunity/s/topic/0T02I000000DaWfWAK/eosceosveos>.
- [55] *Containerlab GUI*. URL: <https://containerlab.dev/manual/gui/>.
- [56] *Kathara Installation*. URL: <https://github.com/KatharaFramework/Kathara/wiki/Linux>.

- [57] *Kathara labconf*. URL: <https://www.kathara.org/man-pages/kathara-lab.conf.5.html>.
- [58] *Kathara dnsmasq*. URL: <https://thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html>.
- [59] *Bridge network driver*. URL: <https://docs.docker.com/engine/network/drivers/bridge/>.
- [60] *resolv.conf(5)*. URL: <https://man7.org/linux/man-pages/man5/resolv.conf.5.html>.
- [61] *Kathara lstart*. URL: <https://www.kathara.org/man-pages/kathara-lstart.1.html>.
- [62] *Kathara-Team*. URL: <https://github.com/KatharaFramework/Kathara/issues>.
- [63] *FRRouting Grundlagen*. URL: <https://docs.frrouting.org/en/latest/basic.html>.
- [64] *FRRouting setup*. URL: <https://docs.frrouting.org/en/latest/setup.html>.
- [65] *Daemon Zebra*. URL: <https://docs.frrouting.org/en/latest/zebra.html>.
- [66] *Daemon ospfd*. URL: <https://docs.frrouting.org/en/latest/ospfd.html>.
- [67] *Web Server Apache2*. URL: <https://ubuntu.com/server/docs/how-to/web-services/install-apache2/>.
- [68] *a2enmod, a2dismod*. URL: <https://manpages.ubuntu.com/manpages/jammy/man8/a2dismod.8.html>.
- [69] *a2ensite, a2dissite*. URL: <https://manpages.ubuntu.com/manpages/bionic/man8/a2ensite.8.html>.
- [70] *Kathará Labor Checker*. URL: <https://github.com/KatharaFramework/kathara-lab-checker>.

Eigenständigkeitserklärung und Erklärung zur Nutzung von KI

Eigenständigkeitserklärungen

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 07.09.2026

Brinda Olivia Ngangya Ndoutchou

Erklärung zur Nutzung von KI

KI-Verzeichnis

Verwendetes KI-Werkzeug

KI-Tool	Zweck
ChatGPT	Unterstützung bei der sprachlichen Überarbeitung selbst verfasster Text insbesondere zur Verbesserung von Grammatik, Rechtschreibung, Ausdruck und Lesbarkeit. Zusätzlich wurde ChatGPT zur Übersetzung von Fehlermeldung aus Visual Studio Code und Linux-Terminal verwendet.

Im Rahmen der Erstellung dieser Arbeit wurde KI-gestützte Werkzeuge unterstützend verwendet. Diese dienten insbesondere zur sprachlichen Überarbeitung (Optimierung von Stil und Grammatik des von mir verfassten Textes) und Übersetzung. Darüber hinaus wurde sie zur Übersetzung von Fehlermeldungen verwendet, die aus Englisch waren und während des Debuggings in Visual Studio Code und Linux-Terminal auftraten. Die inhaltliche Verantwortung sowie die fachliche Ausarbeitung dieser Arbeit liegen vollständig beim Verfasser. Alle technischen Inhalte, Analysen und Bewertungen wurden eigenständig erarbeitet.

Berlin, den 07.09.2026

Brinda Olivia Ngangya Ndoutchou