

Projektarbeit

Software-defined Networking mit Openflow

Mücahit Sagiroglu

Matrikelnummer: 1228852

James Belmonte

Matrikelnummer: 1340604

Naghmeh Ghavidel Rostami

Matrikelnummer: 1249307

Tung Trinh

Matrikelnummer:

Vorgelegt am: 10. Februar 2022

Dozent: Maurizio Petrozziello

Modul 25: Informatik Projekt

Software-defined Networking mit Openflow

Wintersemester 2021/2022

Eigenständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit eigenständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie die aus fremden Quellen direkt oder indirekt übernommenen Stellen/Gedanken als solche kenntlich gemacht haben. Diese Arbeit wurde noch keiner anderen Prüfungskommission in dieser oder einer ähnlichen Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Hiermit stimmen wir zu, dass die vorliegende Arbeit von der Prüferin/ dem Prüfer in elektronischer Form mit entsprechender Software auf Plagiate überprüft wird.

X

James Belmonte

X

Mücahit Sagiroglu

X

Naghmeh Ghavidel

X

Tung Trinh

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Abkürzungsverzeichnis	viii
1 Abstract	1
1.1 Software-defined Networking	1
1.2 Motivation	2
1.3 Problemstellung	3
1.4 Aufbau der Arbeit	4
2 Projekt	5
2.1 Projektziel	5
2.2 Vorgehen	5
2.3 Festlegen von Meilensteinen	6
2.4 Verwendete Werkzeuge	7
3 Durchführung des Projektes	17
3.1 Netzwerkplan	17
3.2 Aufbau des Netzwerkgerüstes in Mininet	19
3.3 Verschlüsselung der Netzwerkverbindung zwischen den Lokationen	23
3.4 Auswahl des Service-Providers	27
3.5 Einrichtung des NAT-Firewalls	29
3.6 Implementierung der Webproxy-Funktion	32
3.7 Aufbau eines zentralen Topologie-Viewers und einer Monitoring-Lösung	34
3.8 Realisierung einer Quality of Service Funktion	38
3.9 Priorisierung der Datenübertragung über API	43
3.10 Analyse weiterer Netzwerkfunktionen	46
4 Zusammenfassung	50
4.1 Analyse der Ergebnisse	50
4.2 Kritische Betrachtung	50
5 Fazit	51
5.1 Zukunftsaussichten	51

Inhaltsverzeichnis

Literaturverzeichnis

52

Abbildungsverzeichnis

2.1	Zeitplan des Projektes	6
2.2	Erstellung eines Standardnetzwerkes mit Mininet	10
2.3	Webbenutzeroberfläche vom Floodlight-Controller	12
2.4	Ausführung von Floodlight über den Linux-Terminal	15
2.5	Mininet Controller Verbindung und Ping-Test	16
3.1	Netzwerkplan aller Lokationen	18
3.2	Netzwerk-Klasse zur Topologie Erstellung	19
3.3	Erstellung und Verbindung von Router und Switch	20
3.4	Erstellung und Verbindung von Hosts und Switch	20
3.5	Verbindung und Konfigurierung der Router	21
3.6	Erstellung eines RemoteController's	22
3.7	Erstellung des Mininet-Objektes	22
3.8	Ausführung und Testen vom Mininet-Skript	23
3.9	Aufstellen der GRE-Tunnel	24
3.10	Wandlung der Pakete bei IPSEC over GRE	25
3.11	Konfiguration der Routen	25
3.12	Erstellung der States für die Ver- und Entschlüsselung	26
3.13	Erstellung der Policies für die Ver- und Entschlüsselung	26
3.14	Verschlüsselter Verkehr zwischen Standort Frankfurt und Berlin	27
3.15	VirtualBox NAT-Adapter	30
3.16	Einbindung und Konfigurierung des NAT-Adapters	30
3.17	Anfrage von H1 wird über Public IP des Routers durchgeführt	31
3.18	Topologie des Netzwerkes auf der Webbenutzeroberfläche von Floodlight	34
3.19	Hinzufügen der Switch IP für s1	35
3.20	Definition der Switch für Nagios	37
3.21	Definition der Services für die Switch	37
3.22	Webbenutzeroberfläche von Nagios Core	38
3.23	Ausgabe der QoS und Queue der Switches aller Lokationen	41
3.24	Sprachanruf zwischen Berlin und Frankfurt	42
3.25	Ausgabe der Flows von s1	42
3.26	UDP-Pakete während dem Sprachanruf	43
3.27	Flows für die Priorisierung und die Drosselung	44
3.28	Abfrage der Daten für den Dateitransfer	45
3.29	Übermittlung der Datei über SSH	45
3.30	Statische IP-Adressenverwaltung	47

Abbildungsverzeichnis

3.31 Dynamische IP-Adressenverwaltung	47
3.32 DHCP-Konfiguration für alle Standorte	47
3.33 Datenverkehr zwischen h1 und h11	48
3.34 Wireshark-Aufnahme der Switch s1 (Frankfurt)	49

Tabellenverzeichnis

3.1	Vergabe von IP-Adressen im Netzwerk	18
3.2	DSL-Angebote verschiedener Internet-Service-Provider	28

Abkürzungsverzeichnis

- ADSL** Asymmetric Digital Subscriber Line
bzw. beziehungsweise
CLI Command Line Interface
DHCP Dynamic Host Configuration Protocol
DNS Domain Name System
ESP Encapsulating Security Payload
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
ISP Internet Service Provider
LAN Local Area Network
MTU Maximum Transmission Unit
NAT Network Address Translation
OSI-Modell Open Systems Interconnection model
SDN Software Defined Network
SDSL Symmetric Digital Subscriber Line
usw. und so weiter
VPN Virtual Private Network
WAN Wide Area Network

1 Abstract

Der vorliegende Projektbericht dient als Dokumentation des Informatikprojekts „Software-Defined Network mit OpenFlow“ an der Frankfurt University of Applied Sciences im Bachelorstudiengang Informatik im Wintersemester 2021/2022.

Das Aufkommen des Internets hat eine Revolution in der Informationstechnologie geschaffen. Durch eine neue Art der Kommunikation kann der Mensch auf nationaler wie auch auf internationaler Ebene effizienter und effektiver Informationen weitervermitteln. Dies bildet die Grundlage für die heutige Wissensökonomie.

Die traditionelle Netzwerkarchitektur ist jedoch seit einem halben Jahrhundert unverändert geblieben und wird für die Geschäftsanforderungen von Unternehmen, Netzwerkbetreibern und Endbenutzern zunehmend ungeeignet. Gegenwärtig werden die Geschäftsanforderungen von Unternehmen immer komplexer und die Anwendungsvielfalt der Endbenutzer nimmt zu, was zu unterschiedlichen Anforderungen der Benutzer an Verbindungsnetzwerke führt. Das Netzwerk muss auf sich schnell ändernde Parameter von Latenz, Bandbreite, Routing, Sicherheit und so weiter (usw.) entsprechend den Anforderungen der Anwendungen reagieren [1].

In den letzten Jahren hat die dramatische Zunahme der Netzwerkkomplexität Schwierigkeiten bei der traditionellen Netzwerkadministration mit sich gebracht. Das Konfigurieren von Computernetzwerksystemen unter Verwendung vordefinierter Richtlinien, das Rekonfigurieren von Netzwerken, um auf Änderungen zu reagieren, die Fehlerkorrektur und der Lastausgleich sind zu gewaltigen Aufgaben geworden. Wenn die Parameter des Netzwerks neu konfiguriert wurden, musste jedes Gerät manuell vollständig neu konfiguriert werden, anstatt einfach nur den Teil der Steuerungsebene zu ändern (vgl. Kim/Feamster 2013: 114f). Dies führte zu einem revolutionären Wandel in der Netzwerktechnologie durch die Zentralisierung der Netzwerkadministration. Seitdem wurde das Konzept des Software-Defined Network (SDN) geboren [2, S. 114–115].

1.1 Software-defined Networking

asd

1 Abstract

1.1.1 Einleitung von James

asddsa

1.1.2 Einleitung von Naghmeh

asddsa

1.1.3 Einleitung von Tung

asddsa

1.1.4 Einleitung von Mücahit

asddsa

1.2 Motivation

Das Modul “Informatik Projekt” wird im 5. Semester des Bachelorstudiengangs Informatik durchgeführt. Nach erfolgreichem Abschluss des Moduls sollten Studierende gewisse Kompetenzen erlernt haben, wie beispielsweise den Software-Engineering Prozess planen und durchführen, als auch auf einem vertieften Niveau gemeinsam programmieren zu können. Außerdem sollten Studierende fähig sein, gemeinsam ein Team zu bilden und einen selbsterstellten Zeitplan einzuhalten sowie auf einem technisch hohen Niveau zu kommunizieren, um als Team auf Ergebnisse zu kommen. Falls unerwartete Komplikationen sowohl technischer als auch sozialer Art entstehen, sollte als Team diese Hürde überwunden werden. Infolgedessen entstand dieser Projektbericht, der als Projektergebnis und als Dokumentation dient, um die erlernten Kompetenzen widerzuspiegeln.

Durch das Thema “Software-Defined Networking mit Openflow” konnte Freizeit mit Studium verbunden werden, da viele selbsterlernte Kenntnisse und Vorkenntnisse aus anderen Modulen praktisch angewendet werden konnten. Zugleich dient die Dokumentation durch ausführliche Erklärungen und Abbildungen auch als Tutorial, dass den Einstieg in das Thema SDN durch Praxis vereinfachen soll.

1.3 Problemstellung

Innerhalb des Informatikprojekts muss sich folgendes Szenario vorgestellt werden:

Ein Unternehmen plane eine Netzwerkkommunikation zwischen vier Standorten mittels Software-Defined Networking Funktionen. Die Hauptverwaltung befindet sich in Frankfurt am Main, die drei weiteren Niederlassungen seien in München, Berlin, Hamburg. Zudem sollte jede Lokation einen Asymmetric Digital Subscriber Line Zugang (ADSL) zum Internet haben.

Darüber hinaus müssen im Netzwerk bestimmte Funktionen und Aufgaben realisiert werden. Es solle nicht nur für jede Lokation jeweils ein privater IP-Adressenbereich genutzt werden, sondern auch ein Netzwerkplan vom gesamten Netzwerk erstellt werden. Außerdem solle jede Kommunikation zwischen den einzelnen Lokationen über eine Virtual Private Network Verbindung (VPN) laufen, somit sei der gesamte Datenverkehr über das Internet und zwischen den Lokationen verschlüsselt. Anschließend müsse ein Service-Provider gefunden werden, der die gewünschte Konfiguration und Anforderungen realisiere. Jedoch sollen der Preis und die benötigte Bandbreite nicht nur für den Internetzugang, sondern auch für die Wide Area Network-Verbindungen (WAN) beachtet und verglichen werden. Ebenfalls solle durch SDN sowohl eine Network Address Translation-Firewall-Funktion (NAT) als auch eine Webproxy-Funktion in allen Lokationen implementiert werden. Ergänzend dazu solle mithilfe des SDN-Controllers sowohl eine graphische Darstellung der Netzwerkstruktur durch einen Topologieviewer realisiert werden als auch eine Monitoring-Lösung. Zudem müsse eine Quality of Service -Funktion implementiert werden, die genügend Bandbreite für Audio beziehungsweise (bzw.) Video-Konferenzen habe, auch wenn diese über die WAN-Verbindung mit Symmetric Digital Subscriber Line (SDSL) 20 Megabit begrenzt sei. Anschließend wird ein weiteres Szenario beschrieben:

“Für eine Spezialanwendung muss eine Software in Berlin wichtige Daten an einem Server in der Zentrale senden, dazu kann diese Software über die API mit dem Controller kommunizieren und diesem dies mitteilen. Dadurch wird der Controller nun alle Knoten auf diesem Weg durchs Netzwerk anweisen, diesen Flow zu priorisieren und alle anderen Datenströme zu drosseln.” Schließlich sollen die Netzwerkfunktionen Hub (Repeater), Bridge, Layer-2-Switch, Layer-3-Switch, Dynamic Host Configuration Protocol (DHCP) und Domain Name System (DNS) analysiert und realisiert werden.

In Kapitel 3 wird für die Implementierungen der Netzwerk Funktionalitäten Screenshots von Mininet in VirtualBox gezeigt, die als Nachweis der einzelnen Funktionalitäten dienen sollen.

1.4 Aufbau der Arbeit

In Kapitel 2 dieser Projektarbeit wird über das generelle Vorgehen in dem Projekt geschrieben. Nach einer kurzen Vorstellung unseres Projektziels, wird das konkrete Vorgehen innerhalb der Gruppe erläutert. Weiterhin wird sowohl über die Festlegung der Meilensteine, als auch über die genutzten Werkzeuge eingegangen. In Kapitel 3 wird die Projektdurchführung erklärt und in den Unterkapiteln werden die erreichten Ergebnisse vorgestellt, die auch die einzelnen Thematiken der jeweiligen Funktionalitäten ergänzen. Anschließend dient Kapitel 4 mit einer kurzen Analyse aller Ergebnisse, als auch eine kritische Betrachtung der Projektanforderung, als Auswertung und Selbstreflektion, was im Rahmen der Projektarbeit nicht funktionierte und umgesetzt werden konnte, in Bezug auf Netzwerkanforderungen, sowie intern zwischen allen Gruppenmitgliedern. Kapitel 5 bildet mit dem Fazit einen Ausblick in die mögliche Zukunft für SDN.

2 Projekt

2.1 Projektziel

Ziel des Projekts war es, ein Netzwerk für ein Unternehmen mit vier Lokationen aufzubauen. Dabei war es besonders wichtig, dass das gesamte Netzwerk mit SDN Funktionen realisiert wurde.

Ein stabiles Netzwerk vom ersten Tag an wird die Grundlage für den Erfolg von Unternehmen sein. Damit Unternehmen gut funktionieren, muss auch das Netzwerksystem gut funktionieren. Das Netzwerk arbeitet mit der richtigen Kapazität und bringt Effizienz. Die Aufgabe war es, ein gutes stabiles Netzwerk aufzubauen, dass zu 100 Prozent bei Datenverkehr funktionierte und unerwartete Sicherheitsprobleme vermeidete.

Das Netzwerksystem musste eng verwaltet und überwacht werden. Zudem musste es leicht unterstützt werden, um Probleme auf die effektivste Weise behandeln und beheben zu können. Es war zwingend erforderlich, dass das Netzwerksystem sicher und verschlüsselt war. Denn Unternehmensdaten sind das Wichtigste. Bei der Netzwerksicherheit ging es auch um den Schutz von Unternehmensressourcen. Je nach Verwendungszweck und Anzahl der Nutzer sollten genügend Bandbreite zur Verfügung gestellt werden.

Im Laufe des Projektberichtes werden die erfolgreichen sowie erfolglosen Ergebnisse des Projekts dokumentiert und dargestellt. Am Ende des Projektes wird ein lauffähiges Produkt entstehen, dass alle benötigten Funktionalitäten erfüllt.

2.2 Vorgehen

Angemessene Aufgabenverteilung im Kollektiv, brachte viele Vorteile für die Arbeitssituation und den Teammitgliedern. Die Nutzung der maximalen Kapazität jedes Teammitglieds war ein effektiver Weg, um die Arbeitseffizienz zu verbessern.

Um die Wünsche und Fähigkeiten jedes einzelnen Mitglieds zu verstehen, wurden Gespräche und Diskussionen frühzeitig durchgeführt. Die Zuweisung von Aufgaben, die der Produktivität jeder Person entsprachen, half den Mitgliedern, effektiver und mit einem angenehmeren Geist zu arbeiten. Den Mitgliedern wurden bestimmte Aufgaben mit

Fristen zugewiesen. Es wurde jede Woche ein permanentes Treffen über Discord gehalten. Spontane Treffen konnten mit dem höchsten Geist und der höchsten Konzentration ebenfalls durchgeführt werden.

Die Analyse von Aufgabenzuweisungen war wichtig, um zu verstehen, was getan werden muss und welche Tools notwendig seien. Der Wissensaustausch half den Mitgliedern, sich Wissen sofort anzueignen und effektiv zu nutzen. Nach einer erfolgreichen Analyse begann unser Team mit der Ausarbeitung eines Plans. Die Arbeit wurde vom Projektleiter aufgeteilt und kontrolliert.

2.3 Festlegen von Meilensteinen

In dem ersten Treffen der Gruppe wurde entschieden, drei zentrale Meilensteine zu definieren (siehe Abbildung 2.1). Grund dafür sei, einen klaren Faden in der Projektarbeit zu konstruieren, um mit der Menge an Informationen strukturiert umgehen zu können. Die Meilensteine wurden mithilfe der Aufgabenstellungen konkretisiert:

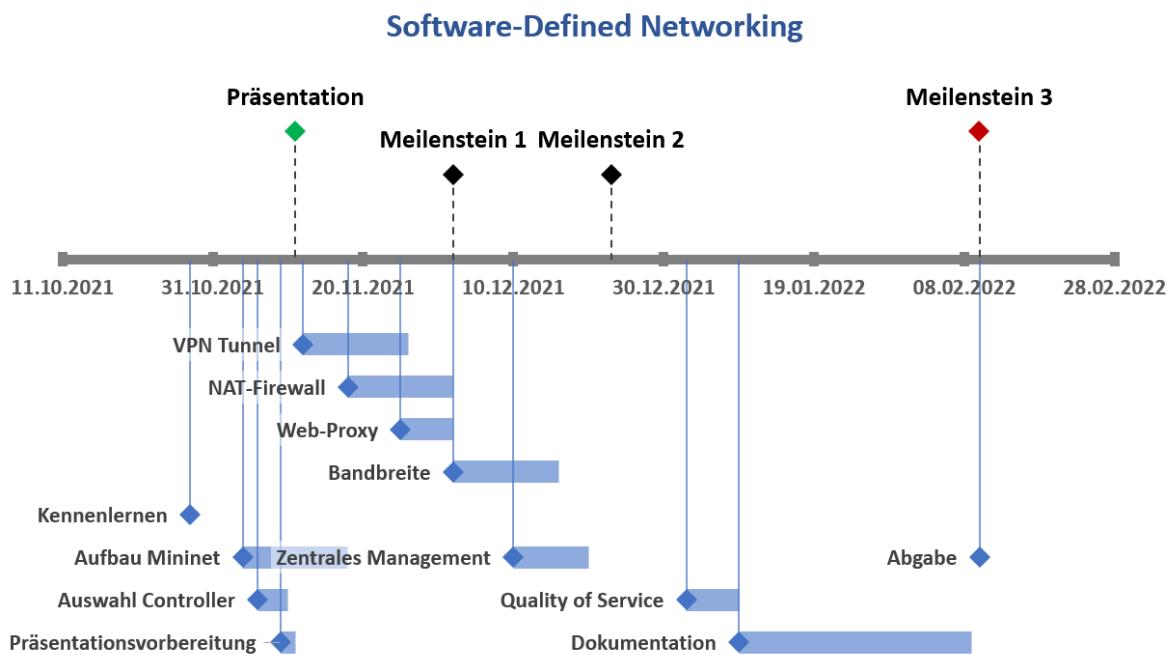


Abbildung 2.1: Zeitplan des Projektes

Meilenstein 1

- Erstellung eines Netzwerkplans für das gesamte Netzwerk

2 Projekt

- Kommunikation zwischen Lokationen soll über eine VPN Verbindung realisiert werden
- Produktauswahl bei einem ISP zur Realisierung des Netzwerkes
- Implementierung einer NAT-Firewall-Funktion in allen Lokationen
- Deadline: 02.12.2021

Meilenstein 2

- Implementierung einer Webproxy-Funktion für den Internet-Zugang in den einzelnen Lokationen
- Implementierung eines Topologie-Viewers und einer Monitoring-Lösung
- Implementierung einer Quality of Service Funktion für Audio- und Videokonferenzen
- Deadline: 23.12.2021

Meilenstein 3

- Priorisierung von einem Datenflow mithilfe des Controllers
- Analyse und Umsetzung der Netzwerkfunktionen von Hub, Bridge, Layer-2-Switch, Layer-3-Switch, DHCP und DNS
- Deadline: 10.02.2022

Durch gängige IT-Projektmanagementmethoden, wie beispielsweise die Scrum-Methode, konnten frühzeitig Ergebnisse erzielt werden. Infolgedessen gab es am Ende der Projektarbeit mehr Zeit, um über Kleinigkeiten zu reflektieren.

2.4 Verwendete Werkzeuge

Im Folgenden werden die für die Implementierung und Evaluierung verwendeten Hardware- und Softwareumgebungen kurz beschrieben.

Dieses Projekt wurde auf VirtualBox Oracle VM Version 6.1 durchgeführt. Unter der Verwaltung von VirtualBox wurde Mininet-Emulator Version 2.3 und Floodlight Controller Version 1.2 installiert. Zur Ausführung von Programmen zur Evaluation wurde außerdem Python3 installiert. Weitere Programme sind auch installiert und sie werden im Ablauf von Kapitel 3 bekannt gegeben und ausführlicher erklärt.

2.4.1 Mininet

Der Mininet-Emulator implementiert die Verbindung zwischen Switches und Controllern. Diese ermöglicht es Entwicklern, die an der Erstellung und dem Testen von Controller-Ressourcen interessiert sind, Mininet zur Durchführung ihrer Simulationen zu nutzen.

2.4.1.1 Einführung

Mininet ist ein Netzwerk Emulator, mit der Netzwerke simuliert werden können. Bei Mininet handelt es sich um eine kostenlose Open-Source-Software, die die virtuelle Maschine und dem Controller die Recherche in SDN und OpenFlow ermöglicht. Mininet ermöglicht eine sehr groß angelegte Topologie, wodurch ein Netzwerk von Hosts, Switches, virtuellen Links und einem Controller erstellt wird. Das Ausführen von Tests mit den Komponenten ist unkompliziert und kann über Python-Schnittstelle erledigt werden. Benutzer können ihre eigene Netzwerktopologie-Struktur nach ihren eigenen Bedürfnissen aufbauen.

2.4.1.2 Funktionalität

Mininet:

- stellt ein einfaches Netzwerk Testbed dar, welches aber auch gleichzeitig günstig ist. Da der OpenFlow Switch in Mininet alle Eigenschaften wie ein echter OpenFlow Switch hat, ist die Anwendung von einem Netzwerkemulator mit Mininet praktisch sinnvoll.
- ermöglicht das Debuggen und Ausführen von Tests größerer Netzwerke mithilfe von Command Line Interface (CLI).
- unterstützt das Einrichten beliebiger benutzerdefinierter Diagramme. Die Anwendungen im Mininet können im echten Netzwerk realisiert werden, ohne dass der Code geändert werden muss.
- bietet eine benutzerfreundliche und erweiterbare Python-API.
- ermöglicht mehreren gleichzeitigen Entwicklern, unabhängig voneinander an derselben Topologie zu arbeiten.
- ermöglicht komplexe Topologietests, ohne dass ein physisches Netzwerk verkabelt werden muss.

2.4.1.3 Nachteile

Aktuell ist Mininet nur unter Linux lauffähig. Nutzer eines anderen Betriebssystems müssen auf Linux entweder durch Simulierung oder Installation zurückgreifen. Zudem könnte der Sourcecode effizienter und sauberer implementiert werden.

Mininet schreibt Ihren OpenFlow-Controller nicht für Benutzer. Wenn Benutzer benutzerdefiniertes Routing- oder Schaltverhalten benötigen, müssen Benutzer einen Controller mit den erforderlichen Funktionen finden oder entwickeln.

Standardmäßig ist Mininet-Netzwerk von Local Area Network (LAN) und vom Internet isoliert - das ist normalerweise eine gute Sache! Benutzer können jedoch das NAT-Objekt und/oder die Option –nat verwenden, um Ihr Mininet-Netzwerk über Network Address Translation mit Ihrem LAN zu verbinden. Benutzer können Ihrem Mininet-Netzwerk auch eine echte (oder virtuelle) Hardware-Schnittstelle hinzufügen (siehe Beispiele/hwintf.py für Details).

Standardmäßig teilen sich alle Mininet-Hosts das Host-Dateisystem und den PID-Speicherplatz. Das bedeutet, dass Benutzer möglicherweise vorsichtig sein müssen, wenn sie Daemons ausführen, die eine Konfiguration in /etc erfordern, und Benutzer müssen darauf achten, dass sie nicht versehentlich die falschen Prozesse beenden.

Im Gegensatz zu einem Simulator hat Mininet keine starke Vorstellung von virtueller Zeit. dies bedeutet, dass Timing-Messungen auf Echtzeit basieren und dass Ergebnisse schneller als Echtzeit (z. B. 100-Gbit/s-Netzwerke) nicht einfach emuliert werden können.

2.4.1.4 Komponenten

Ein Mininet-Netzwerk besteht aus den folgenden Komponenten:

- **Link:** Links sind virtuelle Ethernets, die zwei virtuelle Schnittstellen verbinden. Jeder Link verhält sich für das gesamte System wie ein echter funktionsfähiger Ethernet-Anschluss. Die Datenrate jedes Links wird von Linux Traffic Control (TC) festgelegt.
- **Hosts:** Ein emulierter Host ist eine Reihe von Prozessen auf Benutzerebene, die in einen Netzwerk-Namespace verlagert wird. Netzwerk-Namespaces bieten Prozessgruppen privaten Besitz von Schnittstellen, Ports und Routing-Tabellen.
- **Switch:** Mininet verwendet Open vSwitches, die im Kernelmodus ausgeführt werden, um Pakete zwischen verschiedenen virtuellen Netzwerkschnittstellen auszutauschen. Open vSwitches sind OpenFlow-fähig und bieten die gleiche Semantik für das Senden von Paketen wie einen realen Switch.

- **Controller:** Ein Controller ist in der Mininet-Simulation ein Knoten, der einen OpenFlow-Controller darstellt. Mininet bietet die Möglichkeit einen internen oder externer Controller zu benutzen. Für den externen Controller wird die IPv4-Adresse und der Port benötigt.

2.4.1.5 Installation

Mininet kann auf verschiedene Weisen installiert werden. In unserer Arbeit wurde die Option: Native Installation from Source ausgewählt. Die Installation wird Schritt für Schritt aufgeführt:

1. Git wird über das Linux-Terminal installiert

```
$ sudo apt-get install git
```

2. Über das git Kommando wird die aktuellste Version von Mininet installiert

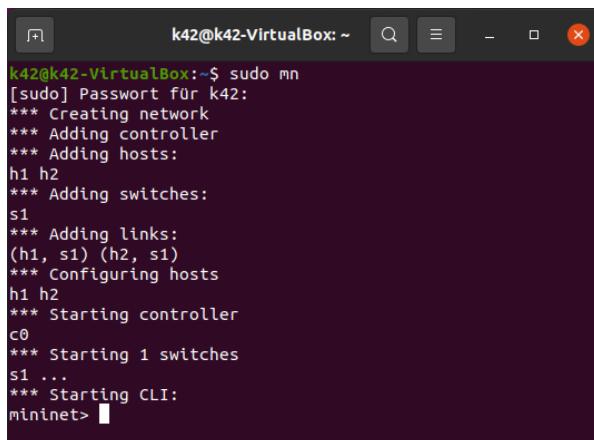
```
$ git clone git://github.com/mininet/mininet
```

3. Mit install.sh die Installation starten

```
$ sudo mininet/util/install.sh -a
```

2.4.1.6 Aufbau

Durch die Eingabe von dem Befehl **sudo mn** wird ein Standardnetzwerk mit zwei Hosts, einer Switch und einem Controller gestartet (siehe Abbildung 2.2).



The screenshot shows a terminal window titled 'k42@k42-VirtualBox: ~'. The command 'sudo mn' is run, and the terminal displays the following output:

```
k42@k42-VirtualBox:~$ sudo mn
[sudo] Passwort für k42:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> █
```

Abbildung 2.2: Erstellung eines Standardnetzwerkes mit Mininet

2.4.2 Floodlight

Floodlight ist ein sogenannter Controller zur Steuerung einer oder mehreren programmierbaren Switches. Die Kommunikation zwischen Switch und Controller erfolgt über ein Kommunikationsprotokoll namens OpenFlow. Die Kontrolle der Switches erfolgt entweder durch Programmierung oder einer vom Controller zur Verfügung gestellten und benutzerfreundlichen Schnittstelle.

2.4.2.1 Einführung

In den letzten Jahren wurde eine Vielzahl unterschiedlicher SDN-Controller entwickelt. Aus diesem Grund gibt es mittlerweile eine riesige Auswahl an SDN-Controllern für die breit gefächerten Einsatzzwecke, wo unter anderem OpenDaylight, Ryu, POX, NOX und Floodlight dazugehören. Mit allen Controllern sind alle Projektziele der Projektarbeit erreichbar.

Floodlight wurde als Controller ausgewählt, da einige Punkte und das daraus resultierende Gesamtprodukt die Gruppe überzeugen konnte. Dazu gehört unter anderem die einfache und gut beschriebene Installation. Die moderne Webbenutzeroberfläche und die verständliche, gut dokumentierte REST-API sind sehr benutzerfreundlich und leicht zu verstehen. Daraus resultiert auch die Option, die REST-API über ein Python-Skript zu benutzen. Die Einbindung des Floodlight-Controllers in Eclipse ermöglicht die Implementierung, Untersuchung und das Debuggen verschiedenster Controller-Funktionen. Die gute Dokumentation des in Java geschriebenen Controllers und einige mit der Installation mitgelieferten Module geben dem Entwickler einen guten Start zur Entwicklung von Netzwerkfunktionen.

2.4.2.2 Funktionalität

Die Funktionalitäten des Floodlight-Controllers unterscheiden sich anhand der Ausführung und der Implementierung. Funktionen können über die Webbenutzeroberfläche per Eingabe ausgeführt werden (siehe Abbildung 2.3). Das Einstellen der Switch-Firewall und der Access Control List sind zwei dieser Funktionen. Nach Aktivierung der Firewall werden alle Pakete, die nicht in der Liste eingetragen sind, fallen gelassen. Die Access Control List arbeitet ähnlich wie die Firewall, wohingegen nur eine Liste mit erwünschten und nicht erwünschten Quellen existiert. Die Quellen werden anhand der Paket-Informationen angegeben. Bei einem Treffer wird die Quelle je nach Einstellung zugelassen oder verweigert. Folglich verweigert die Firewall jegliche Verbindung nach Aktivierung, wohingegen die Access Control List nur bestimmte Zugriffe auf ein Netzwerk zulässt oder ablehnt. Auf der Webbenutzeroberfläche sind Informationen zu dem vom Controller gesteuerten Netzwerk einsehbar. Dazu gehört die Anzahl der Switches, Hosts und Links sowie der verbrauchten Ressourcen des Controllers und der Netzwerktopologie. Die Statistikfunktion des Controllers kann auf der Benutzeroberfläche aktiviert werden.

Dieser dient zur ausführlichen Weiterverarbeitung und der Anzeige der vom Controller gesammelten Statistik. Dazu gehören die Flow, meter, queue, aggregate, table und port Statistiken. Die Sammlung benutzerdefinierter Statistiken sind ebenfalls möglich und müssen vom Entwickler nachimplementiert werden. Über die REST-API können sogenannte Flows eingetragen werden, die zur Steuerung des Netzwerkes beitragen. Dabei können Datenpakete modifiziert, zwischengespeichert und umgeleitet werden.

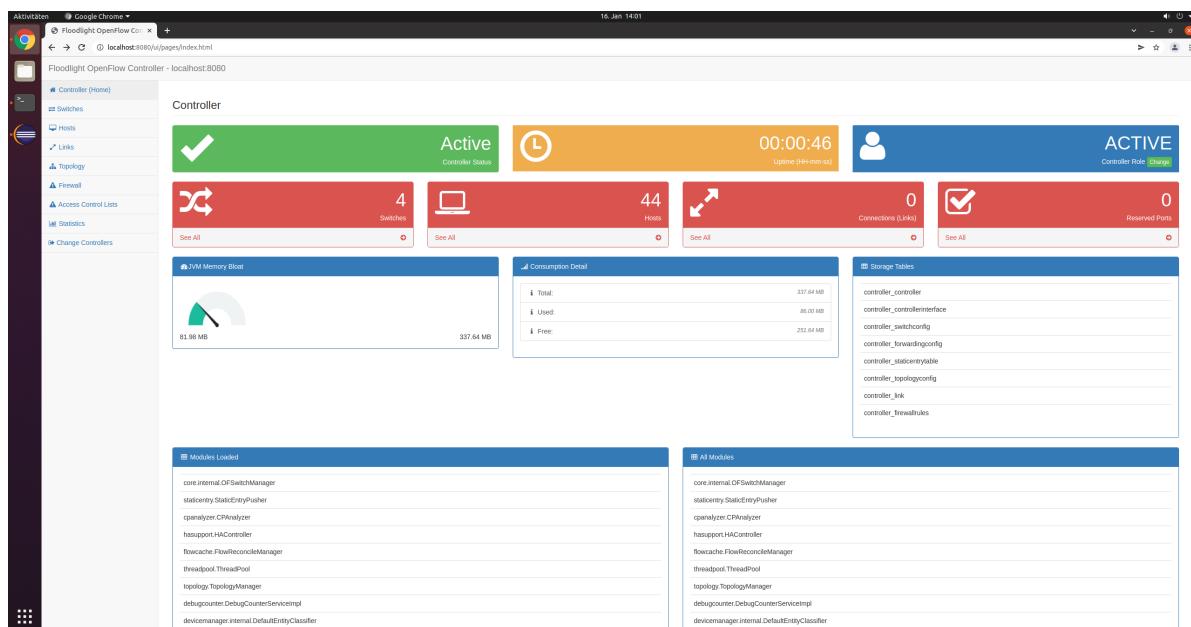


Abbildung 2.3: Webbenutzeroberfläche vom Floodlight-Controller

2.4.2.3 Nachteil

Das ganze Netzwerk ist betroffen, wenn Floodlight ausfällt oder nicht erreichbar ist. Dieser Single Point of Failure gilt für alle Controller und stellt ein Risiko für Netzwerke, die eine hohe Verfügbarkeit und Zuverlässigkeit fordern. Die Wartung und Konfiguration eines Controllers für komplexe Netzwerkstrukturen erfordern viele geschulte Angestellte.

2.4.2.4 Komponenten

Mit der Installation von Floodlight kommen sogenannte Module zum Einsatz. Die meisten der Module sind bereits aktiviert und stellen bestimmte Funktionen zur Verfügung. Einer davon ist die Learning Switch, welcher für die Speicherung der Routen zu den Hosts zuständig ist. Wenn ein Host einen anderen Host im gleichen Netzwerk erreichen will und der Switch die Route nicht kennt, wird ein Broadcast ausgeführt, der die Anfrage auf allen Ports ausgibt. Wenn der Host antwortet, speichert der Switch die MAC-Adresse des

jeweiligen Hosts ab und muss somit keinen Broadcast durchführen. Weitere Beispieldomäne wären der Load Balancer und der Topology Manager. Der Load Balancer sorgt für einen Ausgleich des Datenverkehrs im gesamten Netzwerk. Der Topology Manager stellt die Netzwerktopologie über die Webbenutzeroberfläche grafisch dar. Es existieren noch weitere Module, wobei auch eigene programmiert werden können.

2.4.2.5 Installation

Die Installation des Floodlight-Controllers kann auf den Betriebssystemen Linux, Mac oder Windows erfolgen. Es wird das Java Development Kit 8, Maven, Git, build-essential und das Python Development Paket benötigt. Da Floodlight in Java geschrieben wurde, wird auch zur Ausführung Java verwendet. Maven wird zum sogenannten Builden benutzt, bei dem die Software Floodlight aus mehreren Dateien zusammengestellt wird. Das Python Development Paket wird zur Ausführung und Git zum Herunterladen von Floodlight vorausgesetzt. Build-essential werden zum Kompilieren einiger Software verwendet. Im Folgenden wird die Installation auf Linux Schritt für Schritt erklärt. Befehle müssen im Linux-Terminal zeilenweise eingegeben werden.

1. Alle benötigten Abhängigkeiten installieren.

```
$ sudo apt-get install build-essential git maven python-dev openjdk-8-jdk
```

2. Java Compiler als Alternative festlegen. Befehl eingeben und JDK 8 Auswählen.

```
$ sudo update-alternatives --config javac
```

3. Programmcode per Github herunterladen und aktualisieren

```
$ git clone git://github.com/floodlight/
```

```
$ floodlight.git
```

```
$ cd floodlight
```

```
$ git submodule init
```

```
$ git submodule update
```

4. Floodlight Ordnerrechte zuweisen und Builden

```
$ cd ..
```

```
$ sudo chown -hR Benutzername:Gruppenname floodlight/
```

```
$ cd floodlight/
```

```
$ mvn package -DskipTests
```

5. Floodlight im Terminal ausführen (siehe Abbildung 2.4)

\$ java -jar target/floodlight.jar

Es besteht die Möglichkeit, den Floodlight Controller mithilfe von Eclipse zu starten, somit muss Floodlight nicht im Terminal ausgeführt werden. Außerdem ist Floodlight durch die Entwicklungsumgebung Eclipse leichter auszuführen und eine Bearbeitung ist übersichtlicher. Zudem hat man eine schnelle Übersicht vom kompletten Controller, da alle Klassen in einem Eclipse Ordner einzusehen sind. Mit sudo mvn package –Declipse werden mehrere Dateien erstellt. Mit den neu erstellten Dateien kann ein neues Eclipse Projekt importiert werden. Anschließend wird Eclipse gestartet und eine neue Arbeitsumgebung erstellt. **File -> import -> General -> Existing Projects into Workspace** und auf **Next** klicken. Von **Select root directory** auf **Browser** klicken und Verzeichnis das Floodlight enthält auswählen. Das Projekt mit **finish** ausführen und damit sollte Floodlight auf Eclipse importiert sein.

Um Floodlight auf Eclipse auszuführen, wählt man **run configuration** aus, rechts klickt auf **java application** und **new**. Anschließend wird die neue Java Application **FloodlightLaunch** genannt, es nutzt das Projekt Floodlight und **net.floodlight.controller.core.Main** in der Main-Klasse. Nachdem dies konfiguriert wurde, kann das Programm in Eclipse ausgeführt werden.

2.4.2.6 Aufbau

Nach erfolgreicher Installation und Ausführung von Floodlight läuft der Controller standardmäßig auf Port 6653. Im Terminal werden alle informativen Ereignisse ausgegeben. Um den Controller zu stoppen, wird die Tastenkombination Steuerung und C gleichzeitig gedrückt (siehe Abbildung 2.4).

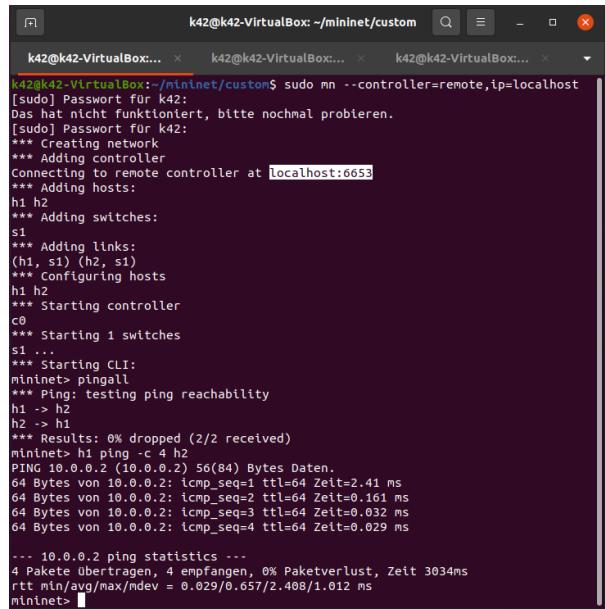
2 Projekt

```
k42@k42-VirtualBox:~/floodlight/floodlight$ java -jar target/floodlight.jar
2022-01-31 20:43:21.406 INFO [n.f.c.m.FloodlightModuleLoader] Loading modules from src/main/resources/floodlightdefault.properties
2022-01-31 20:43:21.503 WARN [n.f.r.RestApiServer] HTTPS disabled; HTTPS will not be used to connect to the REST API.
2022-01-31 20:43:21.504 WARN [n.f.r.RestApiServer] HTTP enabled; Allowing unsecure access to REST API on port 8080.
2022-01-31 20:43:21.504 WARN [n.f.r.RestApiServer] CORS access control allow ALL origins: true
2022-01-31 20:43:21.636 WARN [n.f.c.i.OFSwitchManager] SSL disabled. Using unsecure connections between Floodlight and switches.
2022-01-31 20:43:21.636 INFO [n.f.c.i.OFSwitchManager] Clear switch flow tables on initial handshake as master: TRUE
2022-01-31 20:43:21.636 INFO [n.f.c.i.OFSwitchManager] Clear switch flow tables on each transition to master: TRUE
2022-01-31 20:43:21.636 INFO [n.f.c.i.OFSwitchManager] Setup default rules for all tables on switch connect: true
2022-01-31 20:43:21.641 INFO [n.f.c.i.OFSwitchManager] Setting 0x1 as the default max tables to receive table-miss flow
2022-01-31 20:43:21.672 INFO [n.f.c.i.OFSwitchManager] OpenFlow version OF_15 will be advertised to switches. Supported fallback versions [OF_10, OF_11, OF_12, OF_13, OF_14, OF_15]
2022-01-31 20:43:21.676 INFO [n.f.c.i.OFSwitchManager] Listening for OpenFlow switches on [192.168.1.20]:6653
2022-01-31 20:43:21.677 INFO [n.f.c.i.OFSwitchManager] Openflow socket config: 1 boss thread(s), 16 worker thread(s), 60000 ms TCP connection timeout, max 1000 connection backlog, 4194304 byte TCP send buffer size
2022-01-31 20:43:21.678 INFO [n.f.c.i.Controller] ControllerId set to 1
2022-01-31 20:43:21.678 INFO [n.f.c.i.Controller] Shutdown when controller transitions to STANDBY HA role: true
2022-01-31 20:43:21.678 WARN [n.f.c.i.Controller] Controller will automatically deserialize all Ethernet packet-in messages. Set 'deserializeEthPacketIn' to 'FALSE' if this feature is not required or when benchmarking core performance
2022-01-31 20:43:21.678 INFO [n.f.c.i.Controller] Controller role set to ACTIVE
2022-01-31 20:43:21.706 INFO [n.f.l.i.LinkDiscoveryManager] Link latency history set to 10 LLDP data points
2022-01-31 20:43:21.706 INFO [n.f.l.i.LinkDiscoveryManager] Latency update threshold set to +/-0.5 (50.0%) of rolling historical average
2022-01-31 20:43:21.709 INFO [n.f.t.TopologyManager] Path metrics set to LATENCY
2022-01-31 20:43:21.709 INFO [n.f.t.TopologyManager] Will compute a map of 3 paths upon topology updates
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default hard timeout not configured. Using 0.
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default idle timeout set to 5.
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default table ID not configured. Using 0x0.
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default priority not configured. Using 1.
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default flags will be set to SEND_FLOW_REM false.
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default flow matches set to: IN_PORT=true, VLAN=true, MAC=true, IP=true, FLAG=true, TPPT=true
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Default detailed flow matches set to: SRC_MAC=true, DST_MAC=true, SRC_IP=true, DST_IP=true, SRC_TPPT=true, DST_TPPT=true
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Not flooding ARP packets. ARP flows will be inserted for known destinations
2022-01-31 20:43:21.717 INFO [n.f.f.Forwarding] Flows will be removed on link/port down events
2022-01-31 20:43:21.718 INFO [n.f.s.StatisticsCollector] Statistics collection disabled
2022-01-31 20:43:21.718 INFO [n.f.s.StatisticsCollector] Port statistics collection interval set to 10s
2022-01-31 20:43:21.719 INFO [n.f.h.HAController] Configuration parameters: {serverPort=192.168.1.20:4242, nodeId=1}
2022-01-31 20:43:21.741 INFO [o.s.s.i.SyncManager] [1] Updating sync configuration ClusterConfig [allNodes=[1=Node [hostname=192.168.56.1, port=6642, nodeId=1, domainId=1], 2=Node [hostname=192.168.56.1, port=6643, nodeId=2, domainId=1], 3=Node [hostname=192.168.56.1, port=6644, nodeId=3, domainId=1], 4=Node [hostname=192.168.56.1, port=6645, nodeId=4, domainId=1]], authScheme=CHALLENGE_RESPONSE, keyStorePath=/etc/floodlight/myKey.jceks, keyStorePassword is set]
```

Abbildung 2.4: Ausführung von Floodlight über den Linux-Terminal

2.4.3 Ergebnis

Nach der Ausführung von Floodlight, wurde dieser mit einer OpenFlow-fähigen Switch verbunden. Der Switch wurde mit Mininet mit der Angabe des Controllers simuliert. Die Konsole zeigt die erfolgreiche Verbindung mit dem Controller. Die Konnektivität im Netzwerk kann mit dem Befehl ***pingall*** überprüft werden. Die Konnektivität zwischen Host 1 und Host 2 wird durch den Befehl ***h1 ping h2*** getestet. Durch Wireshark kann der ausgelöste Datenverkehr betrachtet werden (siehe Abbildung 2.5).



```
k42@k42-VirtualBox:~/mininet/custom$ sudo mn --controller=remote,ip=localhost
[sudo] Passwort für k42:
Das hat nicht funktioniert, bitte nochmal probieren.
[sudo] Passwort für k42:
*** Creating network
*** Adding controller
Connecting to remote controller at localhost:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> h1 ping -c 4 h2
PING 10.0.0.2 (10.0.0.2) 56(84) Bytes Daten.
64 Bytes von 10.0.0.2: icmp_seq=1 ttl=64 Zeit=2.41 ms
64 Bytes von 10.0.0.2: icmp_seq=2 ttl=64 Zeit=0.161 ms
64 Bytes von 10.0.0.2: icmp_seq=3 ttl=64 Zeit=0.032 ms
64 Bytes von 10.0.0.2: icmp_seq=4 ttl=64 Zeit=0.029 ms
--- 10.0.0.2 ping statistics ---
4 Pakete übertragen, 4 empfangen, 0% Paketverlust, Zeit 3034ms
rtt min/avg/max/mdev = 0.029/0.657/2.408/1.012 ms
mininet>
```

Abbildung 2.5: Mininet Controller Verbindung und Ping-Test

3 Durchführung des Projektes

Die in Kapitel 1.3 dargestellten Problemstellungen werden in diesem Kapitel behandelt und realisiert. Mit den im vorherigen Kapitel erläuterten Werkzeugen wird dieses Projekt umgesetzt. Durch Abbildungen, Code Ausschnitte und Erläuterungen soll die Dokumentation die Vorgehensweise und Überlegungen der Gruppe wiedergeben.

3.1 Netzwerkplan

Die topologische Struktur des Netzwerks, die in Mininet erstellt wird, ist in Abbildung 3.1 dargestellt. Das Netzwerkdiagramm enthält 40 Hosts, 4 Switches, 4 Router und einen Controller. Der Floodlight-Controller hat einen globalen Überblick über die physikalische Topologie. Die 4 Switches sind mit dem Controller verbunden. Zehn Hosts in den jeweiligen Lokationen sind mit einem Switch verbunden. Um die Hosts in jeder Lokation mit dem Internet und mit anderen Lokationen zu verbinden, wird ein Router benötigt. Durch die grüne Linie und den Tunnel wird gezeigt, dass die Kommunikation zwischen Lokationen verschlüsselt sind. Ebenfalls sind alle Router der Lokationen mit dem Internet verbunden, welches durch eine Cloud visualisiert wird.

3 Durchführung des Projektes

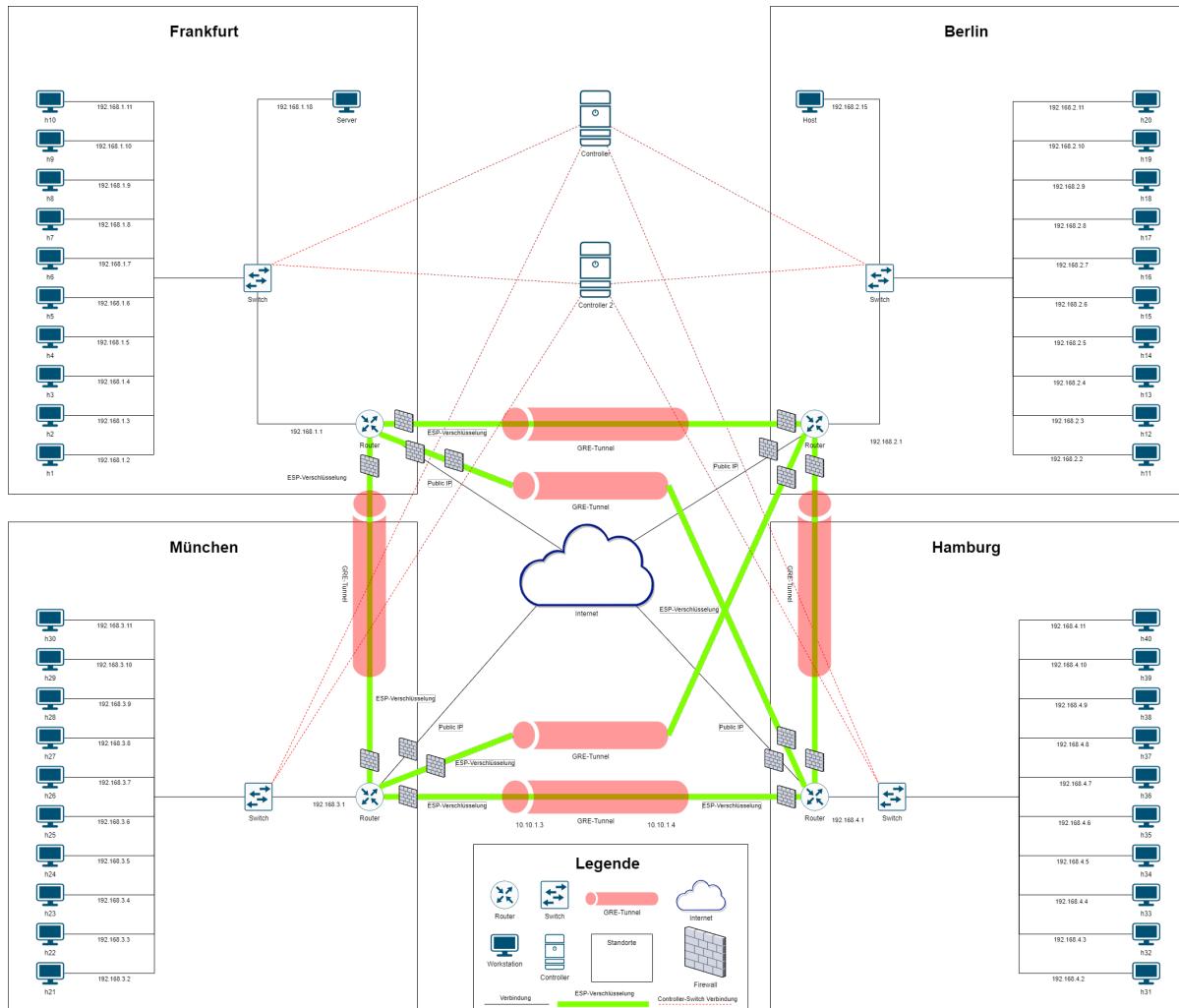


Abbildung 3.1: Netzwerkplan aller Lokationen

Tabelle 3.1: Vergabe von IP-Adressen im Netzwerk

Ort	Frankfurt	Hamburg	München	Berlin
Subnetz	192.168.1.0/24	192.168.2.0/24	192.168.3.0/24	192.168.4.0/24
Netzwerkmaske	255.255.255.0	255.255.255.0	255.255.255.0	255.255.255.0
Broadcast	192.168.1.255	192.168.2.255	192.168.3.255	192.168.4.255
Router	192.168.1.1	192.168.2.1	192.168.3.1	192.168.4.1
Switch	192.168.1.20	192.168.2.20	192.168.3.20	192.168.4.20
erster Host	192.168.1.2	192.168.2.2	192.168.3.2	192.168.4.2
letzter Host	192.168.1.11	192.168.2.11	192.168.3.11	192.168.4.11
Controller	192.168.1.20			
Server in Frankfurt	192.168.1.18			
Berliner Host				192.168.2.15

3.2 Aufbau des Netzwerkgerüstes in Mininet

In diesem Abschnitt wird die beschriebene Topologie unter Verwendung von Mininet simuliert und erklärt.

3.2.1 Durchführung

In der Main-Funktion werden die Komponenten eines Netzwerks deklariert und aufgerufen. Das sind eine Topologie, ein Controller mit zugewiesenen Port und ein Mininet Objekt mit der deklarierten Topologie. Anschließend wurde für unsere Router Routing-Regeln und Informationen gegeben.

3.2.2 Aufbau der Topologie

Die Klasse Netzwerk bildet die Netzwerktopologie. Diese befindet sich in der Main-Funktion des Mininet-Skripts. Das Mininet-Skript besteht aus einer Main-Funktion. Die Main-Funktion enthält eine definierte Klasse **Netzwerk()**. Mithilfe dieser Klasse wird das Netzwerk beziehungsweise eine Topologie erstellt (siehe Abbildung 3.2). Die Klasse übernimmt ein Topo-Objekt an dem er mit der in ihm definierten **build()** Methode die Konfiguration des Netzwerkes vornimmt. In der **build()** Methode wird zuerst ein String definiert, der den privaten-IP-Bereich der vier Lokationen enthält. Der **defaultIP** String bleibt in unvollständiger Form **192.168.%s.1/24**. Somit kann er später durch passende Stellen ersetzt und genutzt werden. Lediglich ist hier im dritten Block ein Platzhalter eingesetzt der beim Erstellen der Router in einer Schleife durch die Zahl der Iteration ersetzt wird. Zunächst wird ein leeres Array/Liste unter dem Namen Routers deklariert. Dies wird dann später genutzt und mit den Router-Objekten gefüllt. Später für die Verlinkung der Router mit dem jeweiligen Switch wird die Liste aufgerufen. Für größere Anzahl von Router ist die Bedeutung der Liste sehr praktisch.

```
# Hier implementieren wir unseren Netzwerkplan (Topologie)
class Netzwerk(Topo):
    def build(self, n=10, **_opts):

        # IP Adresse für die Router r1-r4
        defaultIP = '192.168.%s.1/24'

        # Leere Liste. Gebraucht für später
        routers = []

        # Schleife um 4 Router zu erstellen
        for i in range(1, n+1):
            # Router erstellen
            router = self.addSwitch('r' + str(i))

            # Router mit IP konfigurieren
            router.config(ip=defaultIP % i, defaultGW=None)
```

Abbildung 3.2: Netzwerk-Klasse zur Topologie Erstellung

Im nächsten Teil der Klasse Netzwerk, werden die Komponenten des Netzwerks implementieren. Dies ist mit Hilfe einer Schleife mit 4 Durchläufen ausgeführt worden. Jeder

3 Durchführung des Projektes

Durchlauf entspricht einer Lokation, der jeweils einen Switch, einen Router und zehn Hosts erstellt. Dabei wird bei jeder Iteration erst ein Router-Objekt mit der Methode **self.addNode()** erstellt, bei dem der Name, der private IP-Adressen-Bereich, die MAC-Adresse und der benutzerdefinierte Parameter für die Konfiguration, dass der Router IP-Forwarding aktiviert bekommt, übergeben. Danach wird der Router der vorher erstellten Liste eingefügt. Mit der Methode **self.addSwitch()** wird ein Switch erstellt der einen Namen erhält. Anschließend wird mit der Methode **self.addLink()** eine Verbindung zwischen dem Router und der Switch erstellt. Dabei wird auch die Netzwerkschnittstelle des Routers benannt und der private-IP-Adressenbereich vergeben (siehe Abbildung 3.3).

```
# Erstellen der 4 Router, welche jeweils eine Site darstellen
for r in range(4):
    router = self.addNode(
        'r%s' % (r+1), cls=Router, ip=defaultIP % (r+1), mac='00:00:00:00:00:0%s' % (r+1))
    routers.append(router)

# Erstellen der 4 Switch's für die vier Sites
switch = self.addSwitch('s%s' % (r+1))

# Erstellen der Verlinkung zwischen dem Router und der Switch pro Site
self.addLink(switch, router, intfName2='r%s-eth1' % (r+1),
            params2={'ip': defaultIP % (r+1)})
```

Abbildung 3.3: Erstellung und Verbindung von Router und Switch

Danach folgt noch eine Schleife, bei der insgesamt n Hosts erstellt und mit dem Switch verbunden werden (siehe Abbildung 3.4). Die Hosts erhalten für den jeweiligen privaten-IP-Bereich eine IP, eine MAC-Adresse und die IP des jeweiligen Routers als Standard-Route zugewiesen.

```
# Erstellen der 40 Host's (10 pro Site) mit anschließender Verlinkung
for h in range(n):
    name = ((r)*10)+(h+1)
    host = self.addHost(name='h%s' % (name), ip='192.168.%s.%s/24' % (r+1, h+2),
                        defaultRoute='via 192.168.%s.1' % (r+1), mac='00:00:00:00:00:s0' % (r+1, h))
    self.addLink(host, switch)
```

Abbildung 3.4: Erstellung und Verbindung von Hosts und Switch

Nachdem für alle Lokationen der Rumpf erstellt worden ist, werden die Verbindungen zwischen den Routern mit dem Befehl **self.addLink()** hergestellt. Dabei wird jeder Router mit allen anderen Routern verbunden. Dieser Vorgang wird das Internet simulieren, worauf ebenfalls der Tunnel und die Verschlüsselung implementiert wird. Dabei wird der Netzwerkschnittstellen-Name für beide Router und die jeweilige öffentliche-IP-Adresse definiert. Zusätzlich wird per **bw=20** Befehl die Bandbreite der Leitung auf 20 Megabit gesetzt, wobei dies die geforderte SDSL-Leitung im Kapitel 3.8 darstellen soll (siehe Abbildung 3.5).

3 Durchführung des Projektes

```
# Hinzufügen von Interfaces für die Router und Verlinkung der Router untereinander
# Das stellt unser "Internet" da
self.addLink(routers[0],
             routers[1],
             intfName1='r1-eth2',
             intfName2='r2-eth2',
             params1={'ip': '10.100.12.1/24'},
             params2={'ip': '10.100.12.2/24'},
             bw=20
            )
```

Abbildung 3.5: Verbindung und Konfigurierung der Router

3.2.3 Controller Implementierung

Nach der Erstellung der Topologie geht es weiter bei der Main-Funktion. Es wird ein ***RemoteController-Objekt*** erstellt, der einen Namen, die Konfiguration, um was für ein Controller es sich handelt, die IP-Adresse und den Port, wo er zu erreichen ist, bekommt (siehe Abbildung 3.6). Hier ist wichtig zu erwähnen, dass der Controller auf Ubuntu läuft und zurzeit per ***localhost*** zu erreichen ist. Der Controller könnte auf einer anderen VirtualBox-Maschine laufen und per ***internes Netzwerk*** verbunden werden. Ebenfalls kann der Controller auf dem Hostsystem laufen und per ***Host-Only-Adapter*** in Mininet eingebunden werden.

```
# We create our controller and define the properties
c0 = RemoteController('c0', controller=RemoteController,
                      ip='localhost', port=6653)
```

Abbildung 3.6: Erstellung eines RemoteController's

Anschließend wird ein Mininet-Objekt erstellt, bei dem die erstellte Topologie, der erstellte Controller, ein ***TCLink-Objekt*** für die Einstellung der Bandbreite der Netzwerkadapter und ein ***OVSKernelSwitch-Objekt*** für die Erstellung der Switches als ***Open vSwitches*** (siehe Abbildung 3.7). Diese werden später verwendet, um den Quality of Service zu implementieren. Dabei werden per ***ovs-vsctl-Befehle*** Queues an den Ports der Switches erstellt und die Priorisierung der Pakete vorgenommen.

```
# Initialize a Mininet with our topo object, a controller, the link and switch-version
net = Mininet(topo=topo, controller=c0,
              link=TCLink, switch=OVSKernelSwitch)
```

Abbildung 3.7: Erstellung des Mininet-Objektes

3.2.4 Ergebnis

Wenn das Mininet Skript ausgeführt wird, ist eine Topologie mit 40 Hosts über 4 Switches und 4 Routers zu sehen. Durch ***pingall*** kann es festgestellt werden, dass alles richtig funktioniert hat (siehe Abbildung 3.8).

3 Durchführung des Projektes

```

k42@k42-VirtualBox: ~/mininet/custom
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1) (h9, s1) (h10, s1) (h11, s2) (h12, s2) (h13, s2) (h14, s2) (h15, s2) (h16, s2) (h17, s2) (h18, s2) (h19, s2) (h20, s2) (h21, s3) (h22, s3) (h23, s3) (h24, s3) (h25, s3) (h26, s3) (h27, s3) (h28, s3) (h29, s3) (h30, s3) (h31, s4) (h32, s4) (h33, s4) (h34, s4) (h35, s4) (h36, s4) (h37, s4) (h38, s4) (h39, s4) (h40, s4) (20.00Mbit) (20.00Mbit) (r1, r2) (20.00Mbit) (r1, r3) (20.00Mbit) (r1, r4) (20.00Mbit) (r2, r3) (20.00Mbit) (r2, r4) (20.00Mbit) (r3, r4)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
*** Starting controller
c9
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Waiting for switches to connect
s1 s2 s3 s4
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h3 -> h1 h2 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h16 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h17 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h18 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h19 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h20 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h21 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h22 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h23 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h24 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h25 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h26 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h27 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h28 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h29 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h30 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h31 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h32 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h33 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h34 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h35 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h36 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h37 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h38 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h39 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
h40 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
r1 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
r2 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
r3 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
r4 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 r1 r2 r3 r4
*** Results: 0% dropped (1892/1892 received)

```

Abbildung 3.8: Ausführung und Testen vom Mininet-Skript

3.3 Verschlüsselung der Netzwerkverbindung zwischen den Lokationen

Zwischen den vier Lokationen soll ein Tunnel über das Internet konfiguriert werden. Dieser wird auch Virtual Private Network (VPN) genannt. Der gesamte Datenverkehr durch den Tunnel soll verschlüsselt und für unbeteiligte nicht einsehbar sein.

3.3.1 Vorüberlegung

Der Tunnel wird zwischen den Routern r1-r4 entstehen und für eine Verbindung der Netzwerke der Lokationen sorgen. Damit soll eine **Site-to-Site-VPN** Verbindung zwischen allen Lokationen hergestellt werden. Dieser zeichnet sich durch die Verschlüsselung

ab den Schnittstellen, also den Routern der Lokationen, aus. Zudem werden aus dem privaten Netzwerk eingehende Pakete an den Routern verschlüsselt und weiterverschickt. Der Router an der Ziellokation wird das Paket entschlüsseln und an die Zieladresse weiterleiten. Die Methode, die implementiert werden soll, heißt ***IPSEC over GRE*** und soll alle angestellten Vorüberlegungen ermöglichen.

3.3.2 Durchführung

Für die Durchführung folgt in der Main-Funktion des Mininet-Skripts die Einrichtung der Tunnel zwischen den Routern beziehungsweise den Lokationen. Dafür baut jeder Router mit jedem Router einen GRE-Tunnel auf, für den der Befehl ***ip tunnel add Tunnel-Name mode gre local Schnittstelle-Router-Lokation-A remote Schnittstelle-Router-Lokation-B ttl 255*** bei jedem Router über die Mininet-Methode ***info(net['Router-Name'].cmd(Befehl))*** ausgegeben und ausgeführt wird (siehe Abbildung 3.9). Nachdem die Verbindung definiert wurde, wird der Tunnel-Adapter per ***ip link set Tunnel-Name up*** hochgefahren. Anschließend wird dem Tunnel Adapter mit dem Befehl ***ip addr add Tunnel-IP dev Tunnel-Name*** die Tunnel IP vergeben. Hier ist wichtig, dass der IP-Adressbereich zwischen zwei Lokationen im selben Bereich liegt. Hier erfolgt das gleiche Prinzip wie bei der Erstellung und Simulierung des Internets zwischen den Lokationen.

```
# Setting up GRE-Tunnels between the routers
info(net['r1'].cmd(
    "ip tunnel add gre12 mode gre local 10.100.12.1 remote 10.100.12.2 ttl 255"))
info(net['r1'].cmd("ip link set gre12 up"))
info(net['r1'].cmd("ip addr add 10.10.12.1/24 dev gre12"))

info(net['r1'].cmd(
    "ip tunnel add gre13 mode gre local 10.100.13.1 remote 10.100.13.3 ttl 255"))
info(net['r1'].cmd("ip link set gre13 up"))
info(net['r1'].cmd("ip addr add 10.10.13.1/24 dev gre13"))

info(net['r1'].cmd(
    "ip tunnel add gre14 mode gre local 10.100.14.1 remote 10.100.14.4 ttl 255"))
info(net['r1'].cmd("ip link set gre14 up"))
info(net['r1'].cmd("ip addr add 10.10.14.1/24 dev gre14"))
```

Abbildung 3.9: Aufstellen der GRE-Tunnel

Nachdem der Tunnel aufgesetzt worden ist, sind alle Pakete, die durch den Tunnel versendet werden, nun als Payload eines neuen Paketes, wo der neue IP-Header der IP des Tunnels entspricht (siehe Abbildung 3.10). Daraus folgt, dass nun Pakete, die die Maximum Transmission Unit (MTU) erreichen, jetzt eine geringere Länge annehmen müssen, da der neue Payload aus dem ursprünglichen Payload und IP-Header besteht. Wenn der Fall eintritt, schickt der Router eine Aufforderung an den Absender zurück, das Paket kleiner zu gestalten.

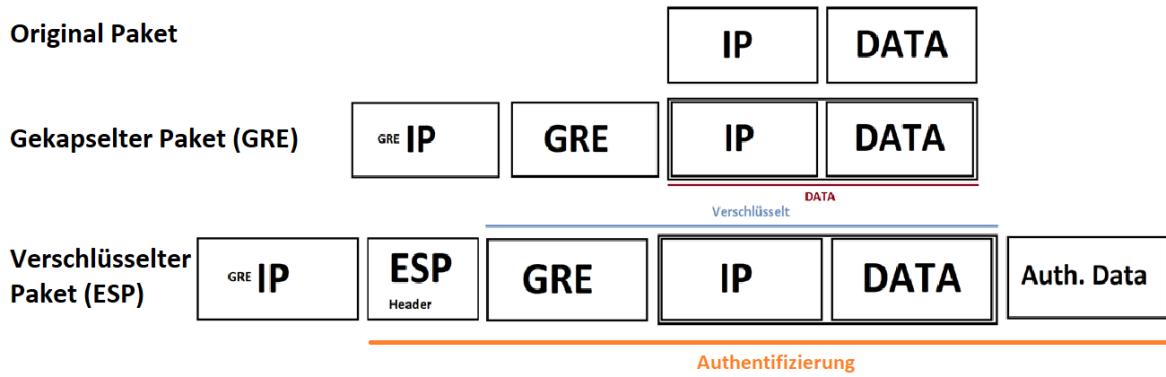


Abbildung 3.10: Wandlung der Pakete bei IPSEC over GRE

Im nächsten Schritt wird die Route zum jeweils anderen Subnetzwerkadressenbereich per ***ip route add IP-anderen-Lokation via IP-Adapter dev Adapter-Name*** Befehl in die Routing-Tabelle eingefügt (siehe Abbildung 3.11). Da es sich hierbei um eine Simulation handelt, sollten alle Lokationen einen jeweils anderen privaten Adressenbereich besitzen. Bei Überschneidungen könnte es zu Problemen führen. Bei einer echten Umgebung mit echten öffentlichen IP-Adressen könnten dieselben privaten IP-Adressen für verschiedene Lokationen genutzt werden. Dies wird in der Realität üblicherweise so gemacht.

```
# Add routing for reaching networks that aren't directly connected through GRE-Tunnel
# route from r1 to r2 and r2 to r1
info(net['r1']).cmd("ip route add 192.168.2.0/24 via 10.10.12.2 dev gre12")
info(net['r2']).cmd("ip route add 192.168.1.0/24 via 10.10.12.1 dev gre21")
```

Abbildung 3.11: Konfiguration der Routen

Der Tunnel von und zu den Lokationen ist nun aufgesetzt. Als Nächstes sollen alle Pakete aus dem privaten IP-Adressenbereich bei der Übermittlung durch den Router verschlüsselt und weitergeleitet werden. Dazu sollen alle Pakete von außen entschlüsselt und zum Zielort weitergeleitet werden. Für diese Methode wird die Verschlüsselung und Entschlüsselung über IPSEC im Transport-Modus mit dem Encapsulating Security Payload (ESP) Protokoll benutzt. Dafür wird zuerst der Befehl ***ip xfrm state add src IP-Adresse-des-Routers dst IP-Adresse-des-Ziel-Routers proto esp Security-Parameter-Index-Key enc 'cbc(aes)' 256bit-Key mode transport*** ausgeführt, um Regeln für die Ver- und Entschlüsselung auf dem Router einzufügen. Jeder Router bekommt zwei Regeln für jeweils einen anderen Router. Eine Regel ist für die Entschlüsselung bei ankommenden Paketen und die andere Regel für die Verschlüsselung bei abgehenden Paketen (siehe Abbildung 3.12).

3 Durchführung des Projektes

```
# Setting up ipsec in Transport mode
info(net['r1'].cmd("ip xfrm state add src 10.100.12.1 dst 10.100.12.2 proto esp spi " +
    "spi12 + " enc 'cbc(aes)' " + key12 + " mode transport"))
info(net['r1'].cmd("ip xfrm state add src 10.100.12.2 dst 10.100.12.1 proto esp spi " +
    "spi21 + " enc 'cbc(aes)' " + key21 + " mode transport"))
```

Abbildung 3.12: Erstellung der States für die Ver- und Entschlüsselung

Der Befehl legt für Pakete mit bestimmter IP-Quelladresse, IP-Zieladresse, einem Security-Parameter-Index Key (SPI) und einer 256 Bit Verschlüsselung fest, bei einer Übereinstimmung das Paket zu verschlüsseln oder zu entschlüsseln. Auf Abbildung 3.10 kann der Wandel von der ursprünglichen Paketstruktur auf die verschlüsselte Paketstruktur nachvollzogen werden. Es wird für eingehende und ausgehende Pakete jeweils eine Regel festgelegt. Der Unterschied ist, dass die IP-Ziel- und Quelladresse, der Security-Parameter-Index-Key und die 256 Bit Verschlüsselung verschieden sind. Durch die States wurden die Ver- und Entschlüsselungen auf den Routern installiert. Die Keys und SPI's wurden zufällig gewählt.

Nun muss den Routern angewiesen werden, auf welchem Datenverkehr die installierten States angewandt werden sollen. Dies geschieht über Policies. Dort wird über den **ip xfrm policy add dir out src IP-Adresse-des-Routers dst IP-Adresse-des-Ziel-Routers tmpl proto esp mode transport** Befehl die Anweisung erteilt, dass auf den IP-Ziel- und Quelladressen die Ver- und Entschlüsselung stattfinden soll. Dabei muss der Befehl zweimal eingegeben werden, wobei die IP-Adressen vertauscht werden (siehe Abbildung 3.13). Es wird nicht die Tunnel-IP, sondern die Router-IP angegeben.

```
info(net['r1'].cmd(
    "ip xfrm policy add dir out src 10.100.12.1 dst 10.100.12.2 tmpl proto esp mode transport"))
info(net['r1'].cmd(
    "ip xfrm policy add dir in src 10.100.12.2 dst 10.100.12.1 tmpl proto esp mode transport"))
```

Abbildung 3.13: Erstellung der Policies für die Ver- und Entschlüsselung

3.3.3 Ergebnis

Durch die Erstellung der Tunnel und der Verschlüsselung der Pakete wurde die komplette Kommunikation zwischen den Lokationen sicherer. Dazu gehört unter anderem die Authentifizierung des Kommunikationspartners und der Verhinderung unautorisierter Veränderungen von Paketen. Zwischen den Lokationen war vorher der gesamte Datenverkehr ersichtlich, wohingegen jetzt alles verschlüsselt ist (siehe Abbildung 3.14).

3 Durchführung des Projektes

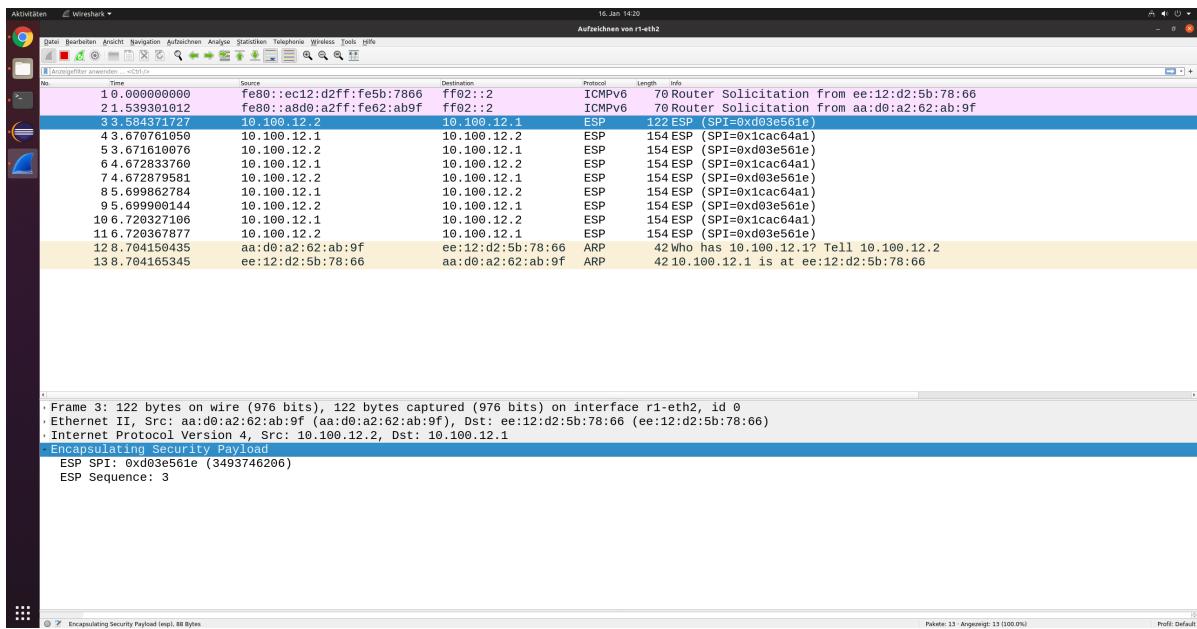


Abbildung 3.14: Verschlüsselter Verkehr zwischen Standort Frankfurt und Berlin

3.4 Auswahl des Service-Providers

Alle Lokationen sind über einen Tunnel durch das Internet miteinander verbunden. Deswegen benötigen alle Lokationen eine DSL-Verbindung mit der richtigen Konfiguration zur Realisierung. Der Preis und die Bandbreite der DSL-Leitung zählen bei der Auswahl als primäre Faktoren. Zusätzlich können weitere Leistungen betrachtet werden. Im Folgenden sind in einer Tabelle die Internet-Service-Provider O2, Vodafone, 1&1 und Telekom mit verschiedenen Bandbreiten und den dazugehörigen Nettopreisen aufgelistet (siehe Tabelle 3.2). Die Angebote sind aktuell und aus den originalen Webseiten der Provider entnommen worden.

3 Durchführung des Projektes

Tabelle 3.2: DSL-Angebote verschiedener Internet-Service-Provider

Anbieter	Tarife	Download in Mbit/s (min/normal/max)	Upload in Mbit/s (min/normal/max)	Monatl. Preis (Netto, 24 Monate)
Telekom	Company Start 16	6,3 / 9,5 / 16	0,7 / 1,5 / 2,4	37 Euro
	Company Start 50	27,9 / 47 / 50	2,7 / 9,4 / 10	42 Euro
	Company Start 100	54 / 83,8 / 100	20 / 33,4 / 40	47 Euro
	Company Start 250	175 / 200 / 250	20 / 35 / 40	57 Euro
	Company Start 500	400 / 500 / 500	80 / 100 / 100	70 Euro
	Company Start 1000	700 / 850 / 1000	200 / 200 / 200	100 Euro
	DSL 16	6,6 / 11 / 16	0,128 / 0,983 / 1	20 Euro
	DSL 50	16,7 / 44 / 50	1,6 / 9,6 / 10	22,5 Euro
1&1	DSL 100	54 / 88,6 / 100	20 / 36,9 / 40	25 Euro
	DSL 250	175 / 200 / 250	20 / 35 / 40	30 Euro
	Glasfaser 500	431 / 250 / 480	215 / 225 / 240	200 Euro
	Glasfaser 1.000	860 / 900 / 1000	430 / 450 / 500	350 Euro
	Plus 16 Regio DSL	6 / 9,5 / 16	0,7 / 0,9 / 1	20 Euro
	Plus 50 Regio DSL	28 / 38 / 50	2,7 / 7,5 / 10	22,5 Euro
Vodafone	Plus 100 Regio DSL	54 / 87 / 100	20 / 37 / 40	25 Euro
	Plus 250 Regio DSL	175 / 210 / 250	20 / 37 / 40	30 Euro
	O2 MyOffice S	0,3 / 8 / 10	0,3 / 1,5 / 2	25 Euro
	O2 MyOffice M	3 / 38 / 50	0,7 / 8 / 10	27,5 Euro
O2	O2 MyOffice L	50 / 83 / 100	10 / 33 / 40	30 Euro
	O2 MyOffice XL	105 / 200 / 250	12 / 33 / 40	35 Euro

Die notwendige Bandbreite für den Download und Upload wurde in der Gruppe abgestimmt. Vorher wurden kleine Tests zur Belegung und Stärkung der Argumente für die Abstimmung gemacht. Bei der Abstimmung wurde pro Arbeitsplatz mindestens 10 Megabit als Download und 5 Megabit als Upload für einen flüssigen Arbeitsrhythmus als notwendig gesehen. Demnach sind bei zehn Arbeitsplätzen ein Upload von mindestens 50 Megabit und ein Download von mindestens 100 Megabit nötig. Nach dem Preis-

3 Durchführung des Projektes

leistungsverhältnis ist der Telekom Company Start 500 Angebot mit einer garantierten Uploadrate von mindestens 80 Megabit und einer garantierten Downloadrate von mindestens 400 Megabit die beste Wahl. Zusätzlich besitzt die Telekom jahrelange Erfahrung und beweist dadurch gute Qualitäten, welches die Entscheidung für das Produkt noch einmal gestärkt hat.

Neben einer DSL-Leitung benötigen die Lokationen eine Standleitung. Eine Standleitung ist sicherer als das Internet, weil der Datenverkehr über eine private Leitung verläuft. Es ist im Rahmen des Möglichen, dass bei der Leitung physikalisch von außen Pakete abgefangen werden können. Standleitungen sind meistens symmetrisch ausgelegt und besitzen die gleiche Uploadrate wie die Downloadrate. Die Preise für Standleitungen gibt es bei Service-Providern erst nach einer Anfrage mit der Angabe der Adressen der Lokationen. Demnach ist eine genauere Preisangabe nicht möglich. Die Kosten für eine Standleitung aller vier Lokationen würde nach Recherchen bei einer symmetrischen Geschwindigkeit von 100 Megabit ungefähr zwischen 500 und 1100 Euro liegen. Die Spanne zwischen den Preisen ist groß, da Preise sich selbst von Gebäude zu Gebäude ändern. Infolgedessen ist für einen genaueren Preis eine Anfrage unabdingbar.

3.5 Einrichtung des NAT-Firewalls

Durch die Network Address Translation (NAT) Firewall Funktion werden private IPv4-Adressen beziehungsweise Geräte geheim gehalten. Es wird durch die Netzwerkadressenübersetzung keine Informationen über die privaten IPv4-Adressen ins World Wide Web geschickt. Dazu wird der IPv4-Header von IP-Paketen aus dem privaten Netzwerkbereich auf die öffentliche IPv4-Adresse verändert. Hinter jeder öffentlichen IPv4-Adresse können mehrere Tausende Geräte stehen und auf das Internet zugreifen. Des Weiteren ist die Anzahl der IPv4-Adressen durch den eigenen Aufbau begrenzt. Für die vier Lokationen würde der Internet-Service-Provider vier öffentliche IPv4-Adressen vergeben. Die Adressen werden vom Internet-Service-Provider in bestimmten Zeitintervallen immer wieder neu vergeben, welches ebenfalls zu einer gewissen Sicherheit beiträgt.

3.5.1 Vorüberlegung

Die NAT-Firewall Funktion muss am Router einer Lokation implementiert werden, da alle Hosts über ihn Anfragen ins Internet verschicken werden. Dazu sollte jede Anfrage ins World Wide Web mit der öffentlichen IPv4-Adresse des Routers durchgeführt werden. Zusätzlich sollte der Router beziehungsweise die Hosts den Dynamic-Name-Server (DNS) konfiguriert bekommen, damit die Hosts nicht nur über die IPv4-Adressen aufs Internet zugreifen.

3.5.2 Durchführung

Um den Hosts der Lokationen den Internetzugang zu ermöglichen, muss zuerst den Routern der Zugang zum Internet möglich sein. Dafür wurde in VirtualBox die vier Netzwerk-Adapter für die vier vorgesehenen Lokationen aktiviert und als NAT konfiguriert (siehe Abbildung 3.15). Die vier Schnittstellen wurden jeweils an die Router r1, r2, r3 und r4 per *Intf(„Schnittstellenbezeichnung“, node=Router-Objekt)* Befehl zugewiesen. Die Schnittstellenbezeichnung kann vorher mit dem Befehl *ifconfig -a* angezeigt werden. Die Router-Objekte werden vorher per *variabel = net.getNodeByName('Router-Bezeichnung')* Befehl instanziert. Jeder Router bekommt eine individuelle Schnittstelle zugewiesen. Die Router führen anschließend den *info(net['Router-Bezeichnung'].cmd("dhclient Schnittstellenbezeichnung"))* Befehl aus, um eine IPv4-Adresse des VirtualBox NAT-Services zu erhalten (siehe Abbildung 3.16). Der IPv4-Adressenbereich, der vom NAT-Service vergeben wird, liegt bei **10.0.X.X/24** und ändert sich je nach virtueller Maschine.

```
r1 = net.getNodeByName('r1')
Intf('enp0s16', node=r1)
info(net['r1'].cmd("dhclient enp0s16"))
info(net['r1'].cmd("sudo iptables -t nat -A POSTROUTING -o enp0s16 -j MASQUERADE"))
```

Abbildung 3.16: Einbindung und Konfigurierung des NAT-Adapters

Da nun eine Internetverbindung für alle Router besteht, muss der Domain Name System Server auf dem Ubuntu-Host festgelegt werden, damit die Hosts nicht nur per IPv4 auf das Internet zugreifen können. Eine Möglichkeit besteht darin, die Datei */etc/resolv.conf* per Admin-Rechte zu bearbeiten und dort den Domain Name System Server festzulegen. Es kann der Domain Name Server von Google mit der IPv4 **8.8.8.8** und/oder **8.8.4.4** eingetragen werden. Ein Nachteil bei der Variante ist, dass nach jedem Neustart des Betriebssystems dieser Vorgang erneut durchgeführt werden muss, da der Eintrag nur temporär bis zum Ausschalten des Betriebssystems erhalten bleibt. Um dem entgegenzuwirken wurde das Paket Resolvconf per *sudo apt install resolvconf* installiert. Dadurch wurden neue Dateien in der Konfigurationsebene von Ubuntu erstellt, wodurch ein permanenter Eintrag des DNS Servers möglich war. Dafür wurde die IPv4 des DNS Server in die Datei */etc/resolvconf/resolv.conf.d/head* eingetragen und gespeichert. Anschließend wurde der Netzwerkmanager per *sudo systemctl restart network-manager* Befehl neu gestartet, um die Einstellungen zu übernehmen. Jetzt ist es den Routern möglich, das Internet auch per Domainnamen zu erreichen.

Die Default-Route der Hosts ist der jeweilige Router in der Lokation. Wenn ein Host eine Website aufruft, schickt er eine Anfrage an und über den Router. Im Moment wird die Anfrage beim Router fallengelassen, da der Router noch keine Regeln bezüglich solcher Anfragen besitzt. Die Regel wird über das im Ubuntu vorhandene Programm *iptables* eingetragen, welches den Linux-Kernel umkonfiguriert. Dazu wird auf allen Routern der Befehl ***sudo iptables -t nat -A POSTROUTING -o Schnittstellenbezeichnung -j MASQUERADE*** ausgeführt. Der Befehl vergibt jedem eingehenden Paket als Quelladresse die IPv4-Adresse der NAT-Schnittstelle des Routers (siehe Abbildung 3.15). Es wird **MASQUERADE** genutzt, weil zum Zeitpunkt der Ausführung des Befehls die IPv4-Adresse der Schnittstelle unbekannt sein beziehungsweise sich ändern kann. Bei einer statischen IPv4-Adresse der NAT-Schnittstelle würde statt MASQUERADE direkt die IP eingegeben werden.

3.5.3 Ergebnis

Zusammenfassend wurde die NAT-Firewall-Funktion an allen Lokationen konfiguriert. Den Hosts ist es nun möglich, Anfragen ins Internet zu versenden. Der Router führt die Anfragen mit der von VirtualBox zur Verfügung gestellten Schnittstelle aus und gibt dem Host die Antwort zurück (siehe Abbildung 3.17).

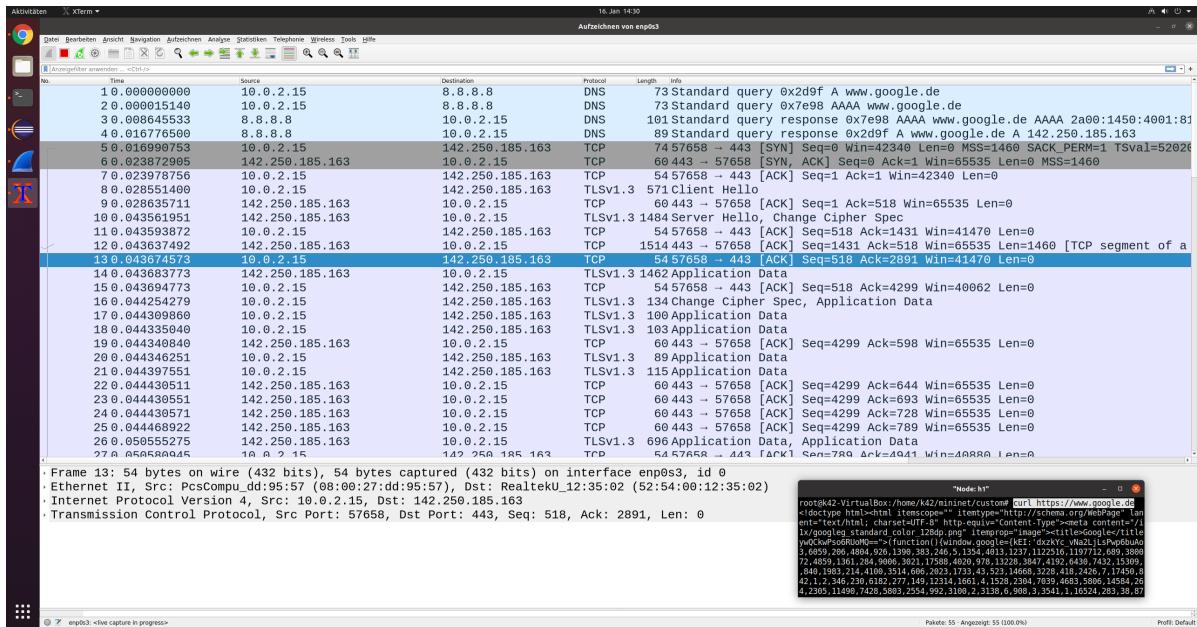


Abbildung 3.17: Anfrage von H1 wird über Public IP des Routers durchgeführt

Ein Problem bei der Durchführung war, dass die Netzwerkschnittstellen beim Beenden von Mininet auch für Ubuntu nicht mehr verfügbar waren. Deshalb musste immer wieder die virtuelle Maschine beziehungsweise Ubuntu neu gestartet werden, wenn Mininet

beendet wurde. Um dem entgegenzuwirken, wurde der Befehl `ip link set Schnittstellenbezeichnung netns 1` im Mininet-Script kurz vor dem Beenden eingefügt, damit die Schnittstelle erneut für Ubuntu verfügbar war.

3.6 Implementierung der Webproxy-Funktion

Der Proxy übernimmt jede Hypertext Transfer Protocol (HTTP) und Hypertext Transfer Protocol Secure (HTTPS) Anfrage der Hosts, führt sie selber durch und leitet die Antwort dem ursprünglichen Host wieder zurück. Der Vorteil hierbei ist, dass der WebProxy-Server für eine Sicherheit in allen Schichten des OSI-Modells sorgen kann. Es muss lediglich nur an dem Web-Proxy-Server Einstellungen bezüglich gewünschter Inhalte, IP-Adressen oder MAC-Adressen vorgenommen werden. Damit ist die Sicherheit für alle Geräte gewährleistet, die über den Web-Proxy-Server Anfragen verschicken. Des Weiteren wäre die Bandbreite weniger ausgelastet, da der Web-Proxy-Server jede neue Antwort in seinem Cache speichert. Bei gleicher Anfrage in einem benutzerdefinierten Zeitintervall wird die Antwort aus dem Cache statt durch erneute Anfrage ins Internet ausgegeben.

3.6.1 Vorüberlegung

Für eine Webproxy-Funktion muss in allen Lokationen ein Webproxy-Server eingerichtet werden. Hosts müssen ohne vorher konfiguriert zu werden, Anfragen über den Proxy versenden können, wenn sie neu an das Netzwerk hinzukommen. Dem Host ist nicht bewusst, dass seine Anfragen über einen Proxy laufen. Anfragen würden über den Switch an den Webproxy weitergeleitet werden. Dieser führt die Anfrage durch und gibt dem Host die Antwort zurück. Die Umleitung sollte durch den Controller konfiguriert werden.

3.6.2 Durchführung

Um die Webproxy-Funktion zu realisieren, wurde in allen Lokationen jeweils ein weiterer Host in Mininet konfiguriert. Es wurde allen eine IPv4-Adresse im privaten Netzwerkkadressenbereich vergeben und mit dem dazugehörigen Switch verbunden. Diese Hosts sollen nun als Webproxy-Server dienen. Es wurde zunächst per **Xterm Proxybezeichnung** Befehl ein externes Terminal gestartet und die Internetverbindung durch das Aufrufen einer beliebigen Website als funktionsfähig getestet. Dazu wurde das Kommandozeilenprogramm `cURL` verwendet, welcher vorher per `sudo apt install curl` installiert werden muss. Als Nächstes sollte eine weitere Schnittstelle des Webproxy-Servers mit einer Schnittstelle von VirtualBox verbunden werden. Dieser sollte dann als Bridge zwischen zwei VirtualBox-Maschinen genutzt werden. Auf der zweiten VirtualBox-Maschine

sollte das Programm Squid installiert werden. Im Gesamtbild sollte eine Anfrage eines Hosts über den Proxy-Server auf eine andere VirtualBox-Maschine weitergeleitet, dort durchgeführt und die Antwort anschließend zurückgeleitet werden. Leider scheiterte dieser Versuch, da bei der Haupt-VirtualBox-Maschine der Promiscuous-Modus des Netzwerkbrückenadapters auf deny eingestellt war. Dadurch konnte keine gescheite Verbindung zwischen den zwei VirtualBox-Maschinen hergestellt werden. Aus diesem Grund haben wurde ein alternativer Proxy-Skript aus Github benutzt, um die Webproxy-Funktionalität direkt auf dem Host beziehungsweise Server einzurichten. Dazu wurde per Xterm ein Terminal gestartet und das Skript ausgeführt. Dieser nimmt nur HTTP Anfragen entgegen, führt sie selber durch und gibt für einen bestimmtes Zeitintervall die Antworten aus dem Cache zurück. Hier ist noch mal zu verdeutlichen, dass das Python-Skript für keine umfangreiche Web-Proxy-Funktionalität ausgelegt ist, jedoch dieser aus Testzwecken benutzt wurde. Als Nächstes wurden alle Pakete, die beim Switch eingehen und einen Ziel-Port als 80 besitzen, an den Proxy weitergeleitet. Hierfür wurde über den Controller die entsprechenden Flows als Match eingetragen, die dazugehörigen Actions implementiert und dem Switch die Anweisungen geschrieben. Dadurch bekommt der Webproxy-Server die Anfragen der Hosts, ohne dass die Geräte umkonfiguriert werden müssen. Der Proxy nimmt die Anfrage entgegen, führt sie selber durch und gibt die Antwort an das jeweilige Gerät weiter. Hier tritt das Problem auf, dass das Gerät eine Antwort von der Website erwartet, jedoch die Antwort vom Proxy geschickt bekommt. Die Antwort des Proxys könnte ebenfalls per Controller modifiziert und an das jeweilige Gerät weitergeleitet werden. Jedoch fehlte die Information des Absenders aus dem Internet zu dem Zeitpunkt, an dem die Antwort vom Proxy an den Host geschickt wurde.

3.6.3 Ergebnis

Resultierend war es nicht möglich, die Webproxy-Funktion zu implementieren. Alternativ könnte eine transparente Web-Proxy-Funktion an dem Router der jeweiligen Lokationen eingerichtet werden. Dieser führt die Anfragen selber durch, speichert und sendet die Antwort an den jeweiligen Host zurück. Auch das Eintragen des Proxyservers an den jeweiligen Hosts, die mit dem Switch verbunden sind, würde sich als funktionsfähig erweisen. Diese Art wird direkter Proxy genannt und trägt viel Aufwand mit sich.

3.7 Aufbau eines zentralen Topologie-Viewers und einer Monitoring-Lösung

3.7.1 Topologie

Die Topologie unseres Netzwerkes kann auf dem Web User Interface des Floodlight-Controllers eingesehen werden. Ein mit der Standardinstallation geliefertes Floodlight-Modul stellt die Information der Topologie in einer Weboberfläche zur Verfügung. Zum Einsehen der Topologie muss auf einem Webbrower die Benutzeroberfläche des Controllers über den Link <http://<controller-ip>:8080/ui/index.html> aufgerufen werden. Anschließend wird auf dem linken Reiter die Option Topology angeklickt, wodurch die Topologie angezeigt wird (siehe Abbildung 3.18). Durch die Benutzung einer Virtual-Box werden vier getrennte Topologien angezeigt. Die Verbindungen zwischen den Routern werden bei der Visualisierung vom Controller ignoriert. Dies schränkt jedoch nicht die Funktionsweise des Netzwerkes ein. Bei der Nutzung vier getrennter VirtualBox-Maschinen, würde die Topologie korrekt dargestellt werden.

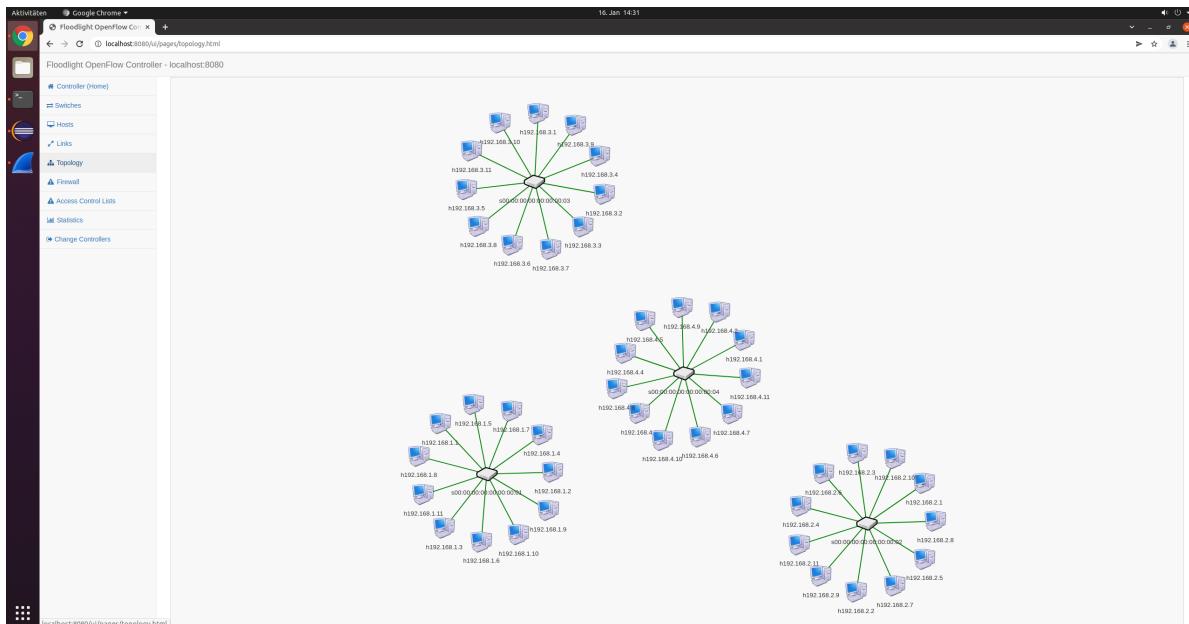


Abbildung 3.18: Topologie des Netzwerkes auf der Webbenutzeroberfläche von Floodlight

3.7.2 Monitoring-Lösung

Für das zentrale Monitoring wird Nagios Core zum Einsatz kommen. Es kann auf einer großen Anzahl an Umgebungen installiert werden. Die Software bietet eine Weboberfläche, die von allen Netzwerkelementen erreicht werden kann. Dabei kann die Verfüg-

barkeit, Geschwindigkeit und Dienste der Netzwerkkomponenten überprüft werden. Bei entstehenden Fehlern können Administratoren benachrichtigt werden, um die Fehler zu beheben. Bei großen Netzwerken spielt dies eine wichtige Rolle, da die Verfügbarkeit bei kritischen Anwendungen von großer Bedeutung ist.

Zunächst wird Nagios Core auf der VirtualBox-Maschine installiert, wo auch Mininet und Floodlight laufen. Bis jetzt lief Floodlight auf dem localhost, also der **127.0.0.1** IP, welches nun geändert werden muss damit die Netzwerkteilnehmer auf die Floodlight REST-API und im Moment wichtiger auf den Nagios Core Dienst zugreifen können. Dafür werden im Mininet-Skript Befehle eingefügt, um den Switches 1-4 eine IP-Adresse zuzuweisen (siehe Abbildung 3.19). Ebenfalls wird die Default-Route als **192.168.1.1** hinzugefügt, da Nagios Core auf der IP **192.168.1.20** laufen wird. Der Controller wurde auch konfiguriert auf dieser IP zu laufen. Nagios Core und der Controller sind nun von allen Netzwerkteilnehmern erreichbar.

```
os.system("sudo ip addr add 192.168.1.20/24 dev s1")
os.system("sudo ip link set s1 up")
```

Abbildung 3.19: Hinzufügen der Switch IP für s1

Die Installation von Nagios Core findet auf der Haupt-VirtualBox-Maschine statt und wird Schritt für Schritt erklärt. Die Befehle sollten nacheinander im Linux-Terminal ausgeführt werden.

1. Das Betriebssystem auf den neuesten Stand bringen

```
$ sudo apt update & & upgrade -y
```

2. Alle benötigten Abhängigkeiten installieren

```
$ sudo apt install -y build-essential apache2 php openssl perl makephp-gd libgd-dev libapache2-mod-php libperl-dev libssl-dev daemonwget apache2-utils unzip
```

3. Nagios Benutzer und Gruppe hinzufügen

```
$ sudo useradd nagios
$ sudo groupadd nagcmd
$ sudo usermod -a -G nagcmd nagios
$ sudo usermod -a -G nagcmd www-data
```

4. Nagios in einen beliebigen Ordner herunterladen

```
$ wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.4.5.tar.gz
```

5. Die Datei extrahieren und in den Ordner wechseln

```
$ tar -zxvf /tmp/nagios-4.4.5.tar.gz  
$ cd /nagios-4.4.5/
```

6. Nagios einstellen (hierbei wird der Benutzer und die Gruppe angegeben)

```
$ sudo ./configure --with-nagios-group=nagios --with-command-group=nagcmd  
--with-httpd_conf=/etc/apache2/sites-enabled/
```

7. Die Dateien vorbereiten und installieren

```
$ sudo make all  
$ sudo make install  
$ sudo make install-init  
$ sudo make install-config  
$ sudo make install-commandmode
```

8. Optional: Kontaktdaten in die contacts.cfg einfügen, um Notifications zu erhalten

```
$ sudo gedit /usr/local/nagios/etc/objects/contacts.cfg
```

9. Webinterface installieren und Module aktivieren

```
$ sudo make install-webconf  
$ sudo a2enmod cgi
```

10. Nagios-Benutzer erstellen

```
$ sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
```

11. Webserver neustarten

```
$ sudo systemctl restart apache2
```

12. Optional: Nagios Plugins herunterladen und installieren

```
$ wget https://nagios-plugins.org/download/nagios-plugins-2.3.3.tar.gz  
$ tar -zxvf /tmp/nagios-plugins-2.3.3.tar.gz  
$ cd /nagios-plugins-2.3.3/  
$ sudo ./configure --with-nagios-user=nagios --with-nagios-group=nagios  
$ sudo make  
$ sudo make install
```

13. Nagios bei Systemstart starten und Nagios ausführen

```
$ sudo systemctl enable nagios
```

```
$ sudo systemctl start nagios
```

Mit all den Schritten wurde Nagios Core inklusive der Plugins installiert. Der Webserver von Nagios läuft nach Neustarten des Dienstes auf den IP-Adressen der Switches. Dazu muss die IP inklusive des Pfades **/nagios** aufgerufen werden. Dort sind im Moment nur Informationen über das System, welches Nagios ausführt, enthalten. Die Netzwerkkomponenten müssen manuell in Nagios eingefügt werden. Bevor das funktioniert, muss für die Switches eine Einstellung in der **nagios.cfg** vorgenommen werden. Dort muss die Raute vor dem **cfg_file=/usr/local/nagios/etc/objects/switch.cfg** Befehl entfernt werden, damit die **switch.cfg** in Nagios mitübernommen wird. Nun werden in der **switch.cfg** alle Switches mit der Bezeichnung, deren IP und der zugehörigen Gruppe aufgelistet (siehe Abbildung 3.20). Danach können sogenannte Services für die Überwachung der Verfügbarkeit, der Geschwindigkeit und weitere Informationen hinzugefügt werden (siehe Abbildung 3.21). Anschließend muss der Nagios Dienst neu gestartet werden. Um Fehler zu vermeiden, sollte per **sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg** Befehl geschaut werden, ob Unstimmigkeiten bei den geänderten Konfigurationen existieren.

```
# Define the switch that we'll be monitoring

define host {
    use generic-switch ; Inherit default values from a template
    host_name S1-FRA ; The name we're giving to this switch
    alias S1-Frankfurt ; A longer name associated with the switch
    address 192.168.1.20 ; IP address of the switch
    hostgroups switches ; Host groups this switch is associated with
}
```

Abbildung 3.20: Definition der Switch für Nagios

```
# Create a service to PING to switch

define service {
    use generic-service ; Inherit values from a template
    host_name S1-FRA,S2-BER,S3-HAM,S4-MUN ; The name of the host the service is associated with
    service_description PING ; The service description
    check_command check_ping!200.0,20%!600.0,60% ; The command used to monitor the service
    check_interval 5 ; Check the service every 5 minutes under normal conditions
    retry_interval 1 ; Re-check the service every minute until its state is determined
}
```

Abbildung 3.21: Definition der Services für die Switch

Nachdem alle Netzwerkteilnehmer und -komponenten hinzugefügt und die gewünschten Services eingestellt worden sind, werden alle Informationen auf der Webbenutzeroberfläche angezeigt (siehe Abbildung 3.22). Unter dem Reiter Hosts kann die Verfügbarkeit aller Teilnehmer angezeigt werden. Die Verfügbarkeit kann unter mehreren Zuständen unterscheiden. Ein Host kann daher den Zustand **Up**, **Down**, **Unreachable** und **Pending** haben, welche auch mit Farben visualisiert werden. Unter Services werden alle

3 Durchführung des Projektes

Teilnehmer mit den konfigurierten Services angezeigt. Die Services haben ebenfalls Zustände. Dazu gehört unter anderem **Ok**, **Warning**, **Unknown**, **Critical** und **Pending**, die ebenfalls mit Farben visualisiert werden. Bei Host Groups werden die Teilnehmer gruppiert und die Informationen zusammengefasst angezeigt. Des Weiteren können viele weitere Einstellungen bezüglich der Benachrichtigungen und der Services über die Webbenutzeroberfläche vorgenommen werden. Dazu klickt man auf einen Teilnehmer und kann auf dem rechten Abschnitt sogenannte **Host Commands** benutzen. Beispielsweise können Kommentare zu Hosts oder Services verfasst werden, damit andere Nutzer bestimmte Vorgänge nachvollziehen können. Zusammenfassend kann über Nagios das gesamte Netzwerk zentral überwacht werden.

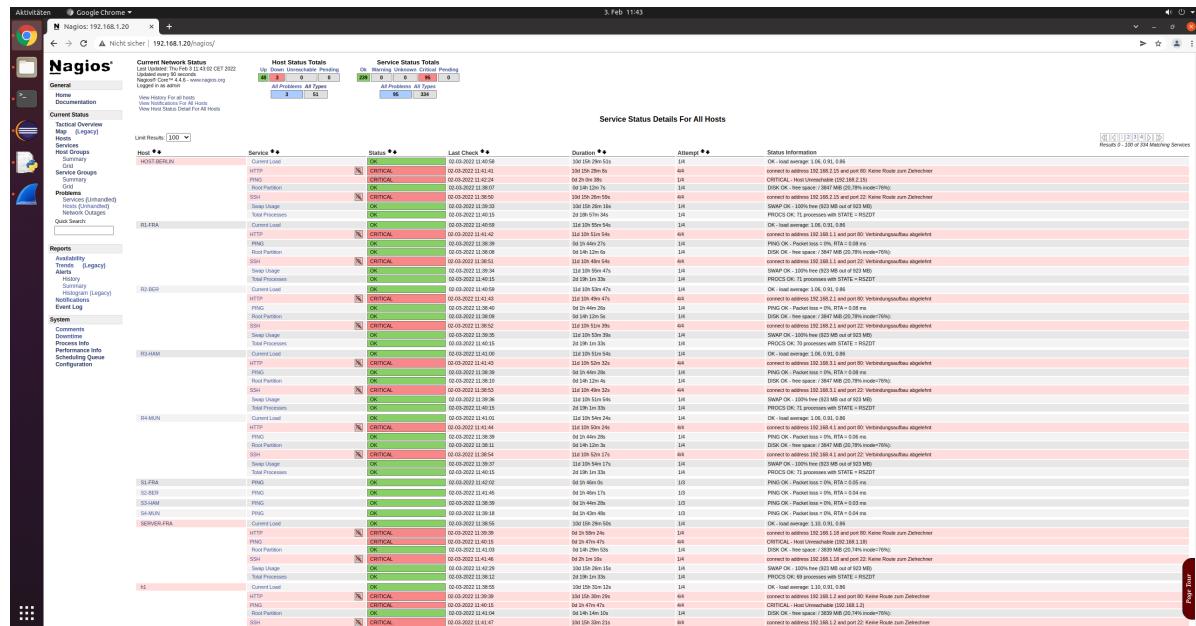


Abbildung 3.22: Webbenutzeroberfläche von Nagios Core

3.8 Realisierung einer Quality of Service Funktion

Die Quality of Service Funktion gewährleistet die Fähigkeit des Netzwerkes, Anwendungen und Datenverkehr selbst bei begrenzter Netzwerkkapazität zuverlässig mit hoher Priorität auszuführen. Das Netzwerkes muss für alle Audio- und Video-Konferenzen innerhalb der Lokationen genügend Bandbreite zur Verfügung stellen. Die Verbindung zwischen den Lokationen besteht aus einer Standleitung mit einer Geschwindigkeit von 20 Megabit.

3.8.1 Vorüberlegung

Pakete, die für die Audio- und Videokonferenzen verantwortlich sind, müssen an dem Switch über den Controller priorisiert werden. Durch die Priorisierung sollen bei einer Auslastung des Netzwerkes die Audio- und Videokonferenzen stabil und flüssig laufen. Nach einer ausgiebigen Recherche kam der Entschluss, UDP Pakete zu priorisieren, da die meisten Videokonferenzprogramme wie beispielsweise Skype und Zoom diese verwenden. Zusätzlich muss getestet werden, wie viel Megabit pro Host eine ausreichende Bandbreite darstellen.

An dem Port der Switch, der zum Router der Lokationen führt, werden zwei sogenannte Queues erstellt. Eine Queue wird für die Audio- und Videokonferenzen und die andere Queue für den restlichen Datenverkehr verwendet. Dabei erhalten die Queues Eigenschaften wie der minimalen und/oder maximalen Bandbreite zur Gewährleistung der Quality of Service. Nach der Konfiguration sollte gezeigt werden, ob Pakete tatsächlich über die Queue verlaufen.

3.8.2 Durchführung

Auf allen vier Switches wird zuerst der Befehl `sudo ovs-vsctl set port Router-Port qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=20000000 queues:0=@newqueue0 queues:1=@newqueue1 -- --id=@newqueue0 create queue other-config:max-rate=20000000 -- --id=@newqueue1 create queue other-config:min-rate=7000000 other-config:max-rate=20000000` mit der Angabe der Bezeichnung für den Port zum Router ausgeführt. Das Ganze geschieht im Ubuntu-Terminal während Mininet ausgeführt wird. Dabei wird auf dem Switch die Quality of Service und genau zwei Queues konfiguriert, die sich anhand der ID unterscheiden. Die allgemeine Bandbreite des Ports wird auf **20 Megabit**, die **Queue 0** auf **maximal 20 Megabit** und die **Queue 1** auf **minimal sieben und maximal 20 Megabit** gesetzt. Die minimale Bandbreite für die Priorisierung wurde anhand eines Selbst-Tests mit Zoom und der Information des Ressourcenmonitors von Windows festgelegt. Dabei wurde mit Video- und Audioübertragungen eine Uploadrate zwischen **500-900 Kilobit** pro Sekunde gemessen. Daraus resultierend wurden für zehn Arbeitsplätzen die Geschwindigkeit auf Minimum sieben Megabit eingestellt.

Der Datenverkehr wird über den Open vSwitch konfiguriert. Der Verkehr soll entweder auf die **Queue 0** oder **Queue 1** weitergeleitet werden. So erhältet der Verkehr die Eigenschaften dieser Queue bezüglicher der festgelegten Raten. Dazu wird über das Mininet-Skript im Ubuntu-Terminal für alle Switches der Befehl `sudo ovs-ofctl add-flow Switch-Bezeichnung priority=1000, actions=set_queue:0, normal` ausgeführt. Der Befehl leitet jedes Paket an die Queue 0. Somit wäre jeglicher Datenverkehr auf maximal 20 Megabit begrenzt. Als Nächstes werden nur die UDP-Pakete mit dem angegebenen Adressenbereich mit dem Befehl `sudo ovs-ofctl add-flow Switch-Bezeichnung`

priority=65535, udp, nw_src=192.168.0.0/16, nw_dst=192.168.0.0/16, actions=set_queue:1, normal bei allen Switches auf die Queue 1 weitergeleitet. Damit hätten die UDP-Pakete eine minimale Bandbreite von sieben und eine maximale Bandbreite von 20 Megabit. Bei den Bedingungen wurde zusätzlich die Priorität eingegeben. Desto höher die Priorität gesetzt wird, desto höher ist die Priorität des Flows und auch somit die Wichtigkeit. Bei der Bedingung für die Quality of Service wird die Priorität auf das Maximum, nämlich **65535**, gestellt. Beim restlichen Datenverkehr kann dieser beliebig größer null gewählt werden. Zusätzlich wird am Ende jeder Bedingung der Flow auf normal gesetzt. Dieser bedeutet, dass das Paket wie gewöhnlich im Layer zwei weiterverarbeitet wird.

3.8.3 Ergebnis

Die Quality of Service und Queue Einstellungen können über die Befehle **sudo ovs-vsctl list qos** und **sudo ovs-vsctl list queue** angezeigt werden (siehe Abbildung 3.23). Mit dem Befehl **sudo ovs-ofctl dump-flows switchbezeichnung** können die Flows des jeweiligen Switches ausgegeben werden. Dort werden unteranderem die Flows für die Quality of Service und des restlichen Datenverkehrs angezeigt.

3 Durchführung des Projektes

```
k42@k42-VirtualBox:~/mininet/custom$ sudo ovs-vsctl list qos
_uuid          : abca236b8-768e-4301-8473-0a4518334434
external_ids   : []
other_config   : [{"max-rate": "20000000"}]
queues         : [{"b6=a7fb620-b568-45ad-a032-bfff115c29ad, 1=8260945e-4c06-4a9d-aa34-178e0527755d}]
type           : linux-htb

_uuid          : 5d0cdccb-6157-4791-bdbc-61ce782b289c
external_ids   : []
other_config   : [{"max-rate": "20000000"}]
queues         : [{"b0=f4237336-db02-4cf8-999c-74c18b4e1017, 1=99a1196e-65f3-4146-b3c4-f06eb27e5035}]
type           : linux-htb

_uuid          : d76c2cf3-b364-4e4e-9c8c-e0722a1ae01d
external_ids   : []
other_config   : [{"max-rate": "20000000"}]
queues         : [{"b0=ad99662-1a2a-4467-8100-0ffa00b6ccb0, 1=c6f400c1-108f-4c34-9de2-614dfffc802}]
type           : linux-htb

_uuid          : 3c9d35ef-893e-41ec-a478-9b33dab083e1
external_ids   : []
other_config   : [{"max-rate": "20000000"}]
queues         : [{"b0=f59b514f-cd1e-4eed-b279-63207933ba83, 1=b713fe13-58cf-487b-9a9d-db7d953e9b87}]
type           : linux-htb

k42@k42-VirtualBox:~/mininet/custom$ sudo ovs-vsctl list queue
_uuid          : 6a7fb620-b568-45ad-a032-bfff115c29ad
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000"}]

_uuid          : c6f400c1-108f-4c34-9de2-614dfffc802
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000", "min-rate": "7000000"}]

_uuid          : ad99662-1a2a-4467-8100-0ffa00b6ccb0
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000"}]

_uuid          : b713fe13-58cf-487b-9a9d-db7d953e9b87
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000", "min-rate": "7000000"}]

_uuid          : f4237336-db02-4cf8-999c-74c18b4e1017
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000"}]

_uuid          : 8260945e-4c06-4a9d-aa34-178e0527755d
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000", "min-rate": "7000000"}]

_uuid          : 99a1196e-65f3-4146-b3c4-f06eb27e5035
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000", "min-rate": "7000000"}]

_uuid          : f59b514f-cd1e-4eed-b279-63207933ba83
_dscp          : []
_external_ids  : []
_other_config  : [{"max-rate": "20000000"}]

k42@k42-VirtualBox:~/mininet/custom$
```

Abbildung 3.23: Ausgabe der QoS und Queue der Switches aller Lokationen

Für den Test wurden zwei weitere VirtualBox-Maschinen mit Ubuntu aufgestellt. Eine Maschine stellt den Server in der Lokation Frankfurt dar. Die andere Maschine stellt einen Host in der Berliner Lokation dar. Beide Maschinen wurden mit der Haupt-VirtualBox-Maschine über eine interne Schnittstelle verbunden. Der Frankfurter Server besitzt die IP **192.168.1.18**, während der Berliner Host die IP **192.168.2.15** besitzt. Dies wurde per Netplan eingerichtet, welcher per **sudo apt install netplan.io** Befehl auf beiden Systemen installiert worden ist. Danach wurden in den Konfigurationsdateien im Dateipfad **/etc/netplan/** die dementsprechende IP-Adresse, das Gateway der Lokation eingegeben und dhcp auf false gestellt. Anschließend wurden die Einstellungen mit dem Befehl **sudo netplan apply** übernommen. Nach den Schritten besteht zwischen dem Host in Berlin und dem Server in Frankfurt über die Haupt-VirtualBox-Maschine eine Verbindung.

3 Durchführung des Projektes

Um nun die Quality of Service und Queue Konfiguration zu testen, wurde auf dem Server in Frankfurt der Prosody *Extensible Messaging and Presence Protocol* (XMPP) Server installiert und konfiguriert. Dazu wurde der Befehl **sudo apt-get install prosody** im Terminal eingegeben. Anschließend wurde der Server anhand der Datei im Pfad **/etc/prosody/prosody.cfg.lua** konfiguriert und mit dem Befehl **sudo systemctl restart prosody-service** neu gestartet. Nun läuft in der Frankfurter Lokation ein XMPP-Server. Des Weiteren wurde auf dem Server in Frankfurt und auf dem Host in Berlin der XMPP-Client Pidgin mit dem Befehl **sudo apt install pidgin** installiert. Es wurde danach über Prosody zwei XMPP-Benutzer mit dem Befehl **sudo prosodyctl adduser user1@192.168.1.18** und **sudo prosodyctl adduser user2@192.168.1.18** mit der Eingabe eines Passwortes erstellt. Mit dem Pidgin-Client wurden in beiden Lokationen die Nutzer eingeloggt, die sich zusätzlich gegenseitig zu den Kontakten hinzugefügt haben. Nun waren die Nutzer gegenseitig für einander sichtbar. Jetzt kann per Rechtsklick ein Sprachanruf gestartet und getätigt werden (siehe Abbildung 3.24). Während dem Anruf kann per **sudo ovs-ofctl dump-flows s1** Befehl erkannt werden, dass die Anzahl der Pakete für die vorher eingefügte UDP-Bedingung stetig steigen (siehe Abbildung 3.25). Per Wireshark können die zwischen den Standorten Übermittelten UDP-Pakete betrachtet werden (siehe Abbildung 3.26). Damit wurde gezeigt, dass die UDP-Pakete des Sprachanrufes über die gesetzte Bedingung auf die Queue 1 weitergeleitet wird. Somit erhalten alle Pakete eine minimale Geschwindigkeit von sieben Megabit.

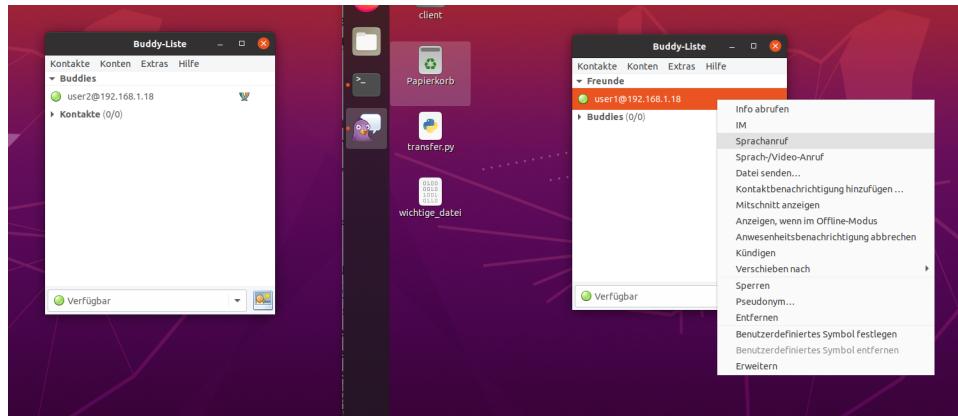


Abbildung 3.24: Sprachanruf zwischen Berlin und Frankfurt

A terminal window titled 'k42@k42-VirtualBox: ~/mininet/custom' displays the output of the command 'sudo ovs-ofctl dump-flows s1'. The output shows two flow entries for queue 1: one with duration 278.397s and another with duration 278.408s. Both flows have priority 0 and are associated with controller 65535. The actions for both flows are 'set_queue:1,NORMAL'.

```
k42@k42-VirtualBox:~/mininet/custom$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=278.397s, table=0, n_packets=3109, n_bytes=385308, priority=0, nw_src=192.168.0.0/16, nw_dst=192.168.0.0/16 actions=set_queue:1,NORMAL
cookie=0x0, duration=278.408s, table=0, n_packets=83, n_bytes=8083, priority=0, actions=CONTROLLER:65535
cookie=0x0, duration=278.424s, table=0, n_packets=8766, n_bytes=1033874, priority=1000 actions=set_queue:0,NORMAL
k42@k42-VirtualBox:~/mininet/custom$
```

Abbildung 3.25: Ausgabe der Flows von s1

3 Durchführung des Projektes

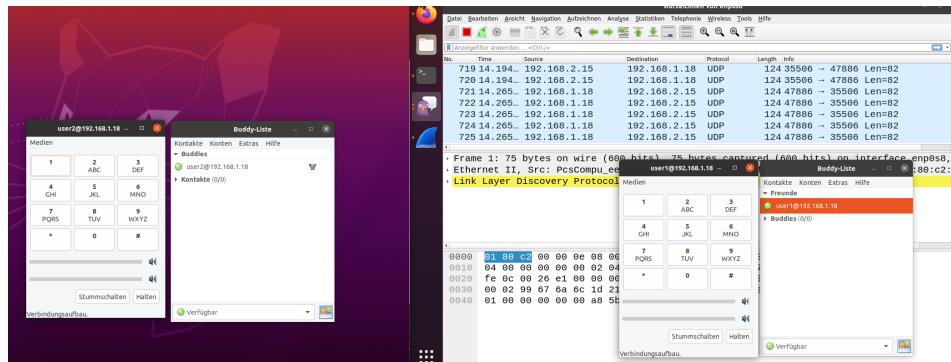


Abbildung 3.26: UDP-Pakete während dem Sprachanruf

3.9 Priorisierung der Datenübertragung über API

3.9.1 Vorüberlegung

Anhand des Schlüsselwortes ***priority*** soll der Flow einer Datenübertragung zwischen der Lokation Berlin und Frankfurt priorisiert werden. Allen anderen Flows werden auf der Strecke eine geringere Priorität zugewiesen, sodass die Datenübertragung bevorzugt wird. Falls die gleiche Priorität bei zwei Flows existiert, wird der Flow mit der genaueren Bedingung verwendet. Die Datenübertragung soll über einem in der Zentrale eingerichteten ***Secure-Shell*** (SSH) Server laufen. Dazu wird auf dem Host in Berlin ein ***SSH-Client*** benötigt. Bei der Übertragung nutzt SSH üblicherweise den ***Port 22*** über dem ***Transport Control Protocol*** (TCP). Mit dieser Information wird die Bedingung für die Priorisierung des Flows festgelegt. Die Übertragung und die Priorisierung werden versucht, in einem ausführbaren Python-Skript zu realisieren, bei der die Angabe der IPv4-Adresse des Hosts in Berlin und des Hosts in der Zentrale als Parameter nötig werden. Es könnte die IPv4-Adresse der Quelle und des Ziels auch automatisch vom Skript durch das System ermittelt werden. Das Skript sollte folgende Schritte beinhalten:

1. IP-Quelle und Ziel eingeben und einlesen (später -> automatisieren)
2. Voraussichtlichen Flow über API priorisieren. Restlichen Datenverkehr drosseln
3. Übertragung per SSH starten
4. Bei Erfolg Priorisierung und Drosslung wieder aufheben
5. Bei Fehler Fehlermeldung ausgeben

Der Controller, genauer der OpenFlow-Server und der Restserver laufen unter der IP-Adresse ***192.168.1.20*** und kann von jedem Host erreicht werden. Dies ist wichtig, da die Priorisierung von außen durchgeführt wird. Ansonsten kann der Controller, wenn dieser auf ***localhost*** läuft, nicht erreicht werden.

3.9.2 Durchführung

Die Durchführung erfolgt mit den im vorherigen Kapitel aufgestellten VirtualBox Maschinen. Auf dem Server in Frankfurt wurde per ***sudo apt install openssh-server*** Befehl der SSH-Server installiert. Dieser wurde eingestellt, auf der IP des Servers zu laufen. Dafür wurde per ***sudo nano /etc/ssh/sshd_config*** Befehl in der Konfigurationsdatei des SSH-Servers die IP-Adresse geändert und die Publickeyauthentication ausgeschaltet. Anschließend wurde der SSH-Server-Dienst mit dem Befehl ***sudo systemctl restart ssh*** neu gestartet, um die Einstellungen zu übernehmen. Auf dem Berliner Host wurde per ***dd if=/dev/zero of=wichtige_datei bs=50MB count=1*** Befehl eine Dummy-Datei für die Übertragung auf dem Schreibtisch erstellt.

Der Server betreibt einen SSH-Server und der Host einen SSH-Client, womit wichtige Dateien übertragen werden. Dafür wurde auf dem Host ein Python-Skript erstellt, welcher näher erläutert wird. Vorher muss per ***sudo pip install paramiko scp*** Befehl die benötigten Python-Module für die SSH-Verbindung installiert werden. Die Installation geschieht mit pip dem Paketverwaltungsprogramm von Python. Das Skript holt sich automatisch die IP-Adresse des Senders anhand der ***get_IP-Methode*** und speichert diese in eine Variable. Anschließend wird die Angabe der Informationen zur Übermittlung verlangt. Dazu gehört der Dateipfad, die IP-Adresse des Empfängers, den Servernamen, das Serverpasswort und den Pfad, wohin die Datei im Ziel gespeichert werden soll (siehe Abbildung 3.28). Danach wird ein Objekt als Rest-Client zur Übermittlung der Prioritätensetzung erstellt. Dieser bekommt die ***192.168.1.20*** als IP-Adresse des Controllers gesetzt. Jetzt werden die Flows zur Priorisierung erstellt. Dabei werden jeweils drei gleiche Flows auf den Switch in der Berliner Lokation mit der ID 2 und auf den Switch in der Frankfurter Lokation mit der ID 1 zugewiesen. Die zwei Flows priorisieren TCP-Pakete mit dem Port 22 und der angegebenen Quell- und Zieladresse. Beide Flows decken eingehende und ausgehende Pakete ab und bekommen ***32768*** als die maximale Priorität gesetzt. Der dritte Flow drosselt den restlichen Verkehr. Hierbei wird nur der Etherettyp auf IPv4 und die Priorität auf ***100*** gesetzt (siehe Abbildung 3.27). Damit ist die Priorität im Vergleich

```
# IP des Floodlight Controllers
pusher=StaticEntryPusher('192.168.1.20')

# Priorisierung
prio1_dst={
    'switch': "00:00:00:00:00:00:00:01",
    "name": "prio1_dst",
    "priority": "32768",
    "eth_type": "0x0800",
    "ip_proto": "6",
    "tcp_dst": "22",
    "ipv4_dst": destination_ip,
    "ipv4_src": source_ip,
    "active": "true",
    "actions": "output=normal"
}

# Priorisierung
prio1_src={
    'switch': "00:00:00:00:00:00:00:01",
    "name": "prio1_src",
    "priority": "32768",
    "eth_type": "0x0800",
    "ip_proto": "6",
    "tcp_src": "22",
    "ipv4_dst": source_ip,
    "ipv4_src": destination_ip,
    "active": "true",
    "actions": "output=normal"
}

# Drosselung
dros1={
    'switch': "00:00:00:00:00:00:00:01",
    "name": "dros1",
    "priority": "100",
    "eth_type": "0x0800",
    "active": "true",
    "actions": "output=normal"
}
```

Abbildung 3.27: Flows für die Priorisierung und die Drosselung

zu der Datenübertragung deutlich gering und wird an nächster Stelle bearbeitet. Die Flows werden nun mit der ***Set-Methode*** dem Controller über die REST-API übertragen, der diese dann an die Switches zuweist. Nachdem die Priorisierung abgeschlossen ist, wird im nächsten Schritt die Datei per SSH übertragen. Dazu wird im Skript die vom Benutzer vorher gefragten Informationen zur Datenübertragung benutzt, um eine SSH-Verbindung aufzubauen, um die Datei zu senden (siehe Abbildung 3.29). Nachdem die Übertragung beendet wurde, wird mit der ***Remove-Methode*** alle Priorisierungen von den Switches über die REST-API entfernt.

```
# Abfrage welche Datei übermittel werden soll
valid = False
while not valid:
    file_path = str(input(
        "Pfad der Datei zum Transfer eingeben (Beispiel:/home/username/Schreibtisch/file)\n"))
    valid = check_file(file_path)

# IP des Empfängers ermitteln
valid = False
while not valid:
    destination_ip = str(
        input("IPv4-Adresse des Ziels eingeben(Beispiel:192.168.1.1)\n"))
    valid = check_ip(destination_ip)

# Name des Empfängers ermitteln
destination_host_name = str(input('Hostname für die IP %s eingeben\n' % destination_ip))

# Password des Empfängers ermitteln
destination_pass = str(input('Passwort für Host %s eingeben\n' % destination_host_name))

# Wohin soll gesendet werden
file_path_remote = str(input(
    "Pfad, wohin Datei bei Remote gespeichert (Beispiel:/home/username/Schreibtisch/file)\n"))
```

Abbildung 3.28: Abfrage der Daten für den Dateitransfer

```
# SSHClient zur Übermittlung
def createSSHClient(server, port, user, password):
    client=paramiko.SSHClient()
    client.load_system_host_keys()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(server, port, user, password)
    return client

ssh=createSSHClient(server=destination_ip, port=22,
                    user=destination_host_name, password=destination_pass)
scp=SCPClient(ssh.get_transport())

scp.put(file_path, remote_path=file_path_remote,
        recursive=False, preserve_times=False)
```

Abbildung 3.29: Übermittlung der Datei über SSH

3.9.3 Ergebnis

Über das Python-Skript ist es nun möglich eine Datei priorisiert auf einen Server an der Zentrale zu senden. Das Skript enthält einige Überprüfungen, um beispielsweise zu schauen, ob die eingegebene IP-Adresse und der Dateipfad gültig sind.

3.10 Analyse weiterer Netzwerkfunktionen

3.10.1 Hub

Repeater arbeiten auf der Ebene 1 des OSI-Modells. Seine Aufgabe ist es, das Signal über dasselbe Netz zu regenerieren, bevor es zu schwach oder beschädigt wird, um die Länge von Signalübertragung im selben Netzwerk zu verlängern. Ein Hub ist im Grunde ein Multiport-Repeater. Ein Hub verbindet mehrere Geräte in einem Netzwerk. Hubs können keine Daten filtern, daher werden Pakete an alle verbundenen Geräte gesendet. Darüber hinaus verfügen sie nicht über die Intelligenz, um den besten Pfad für Pakete zu finden, was zu Ineffizienz und Verschwendungen führt.

Beim Layer-1-Switching geht es im Wesentlichen um unintelligente Geräte wie Hubs und Repeater. SDN-Switches können so angepasst werden, dass sie sich wie Hubs oder Repeater verhalten. Da Hubs und Repeater keine Informationen speichern können, können Controller bei der Layer-1-Switching keine Datenflüsse weiterleiten. In diesem Fall, jedes Mal, wenn ein Paket ankommt, wird es an den Controller weitergeleitet, der dann entscheidet, alle mit ihm verbundenen Hosts zu überfluten. Dies braucht Zeit, um herauszufinden, wohin das Paket gehen soll, was die Gesamtleistung des Netzes beeinträchtigt.

3.10.2 Bridge

3.10.3 Layer-2-Switch

3.10.4 Layer-3-Switch

Da Switching schneller als Routing ist, werden in LANS fast nur mit Switches gearbeitet und an zentralen Stellen Router genutzt, um das Internet zu erreichen. Im Netzwerkplan wurde an jeder Lokation nur ein Switch benötigt, dieser musste nicht VLAN fähig sein. Ein Layer-3-Switch ist beispielsweise ein VLAN-fähiger Switch, der grundlegende Routingfunktionalität bietet und auf Ebene 3 des OSI-Modells arbeitet, die Vermittlungsschicht. Das heißt, ein Layer-3-Switch arbeitet mit IP-Adressen und hat mit der Vermittlung von Paketen zu tun. Innerhalb unseres Projekts entschied man sich keine Layer-3-Switches zu nutzen, da die Funktionalitäten des Layer-2-Switches ausgereicht haben.

3.10.5 Dynamic Host Configuration Protocol

DHCP ist ein Dienst, der an das Netzwerk angeschlossene Geräte mit einer Netzwerkkonfiguration versorgt. Falls ein Gerät eine Konfiguration benötigt, wird dies mithilfe vom DHCP Server konfiguriert, dieser kann beispielsweise im Router des Netzwerks sein.

3 Durchführung des Projektes

Mithilfe von Mininet konnte eine DHCP Konfiguration realisiert werden. Bis zu diesem Punkt wurde die IP-Adressverwaltung ohne DHCP realisiert. Das heißtt, jeder Host musste manuell eine IP-Adresse zugewiesen bekommen. Diese manuelle Konfiguration wurde durch eine for-Schleife im Skript (siehe Abbildung 3.30) durchgeführt.

```
67      # Erstellen der 40 Host's (10 pro Site) mit anschließender Verlinkung
68      for h in range(10):
69          name = ((r)*10)+(h+1)
70          host = self.addHost(name='h%s' % (name), ip='192.168.%s.%s/24' % (r+1, h+2),
71                             defaultRoute='via 192.168.%s.1' % (r+1), mac='00:00:00:00:00:%s' % (r+1,
72                             self.addLink(host, switch)
```

Abbildung 3.30: Statische IP-Adressenverwaltung

Bei aktivierten DHCP in Mininet verwaltet der DHCP-Server IP-Adressen ohne manuellen Eingriff des Administrators und weist jedem Host eine IP-Adresse zu. Somit wurde die Funktion der for-Schleife umgeändert, dass jeder Host von Beginn an die IP-Adresse **0.0.0.0** zugewiesen bekommt und die IP-Adressverwaltung vom DHCP-Server übernommen wird (siehe Abbildung 3.31).

```
69      # Erstellen der 40 Host's (10 pro Site) mit anschließender Verlinkung
70      for h in range(10):
71          name = ((r)*10)+(h+1)
72          host = self.addHost(name='h%s' % (name), ip='0.0.0.0', #% (r+1, h+2),
73                             defaultRoute='via 192.168.%s.1' % (r+1), mac='00:00:00:00:00:%s' % (r+1,
74                             self.addLink(host, switch)
```

Abbildung 3.31: Dynamische IP-Adressenverwaltung

3.10.5.1 Durchführung

Um ein DHCP Server im Netzwerk zu realisieren, muss als erstes der DHCP server auf die VM installiert werden. Dies wird mit dem Befehl **sudo apt-get install isc.dhcp-server** durchgeführt. Als zweites muss im Mininet-Skript die Zeile **os.system("service isc-dhcp-server restart")** ergänzt werden, um den DHCP-Server neu zu starten. Anschließend wird der DHCP server konfiguriert. Dies erfolgt in der **dhcpd.conf** Datei, die sich im Pfad **/etc/dhcp** befindet (siehe Abbildung 3.32). In dieser Datei werden die von den DHCP-Clients benötigten Netzwerkkonfigurationsinformationen gespeichert. Da das Netzwerk aus vier Subnetzen besteht, die mit dem DHCP-Server verbunden sein sollen, müssen vier Subnetze

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.2 192.168.1.254;
    option routers 192.168.1.1;
    INTERFACES="r1-eth0";
}
subnet 192.168.2.0 netmask 255.255.255.0 {
    range 192.168.2.2 192.168.2.254;
    option routers 192.168.2.1;
    INTERFACES="r2-eth0";
}
subnet 192.168.3.0 netmask 255.255.255.0 {
    range 192.168.3.2 192.168.3.254;
    option routers 192.168.3.1;
    INTERFACES="r3-eth0";
}
subnet 192.168.4.0 netmask 255.255.255.0 {
    range 192.168.4.2 192.168.4.254;
    option routers 192.168.4.1;
    INTERFACES="r4-eth0";
}
```

Abbildung 3.32: DHCP-Konfiguration für alle Standorte

3 Durchführung des Projektes

in der ***dhcpd.conf*** Datei definiert werden. Diese Subnetze benötigen folgende Parameter: ***subnet***, ***netmask***, ***range***, ***option routers***, ***default-lease-time***, ***max-lease-time*** und ***INTERFACES***. Wie in der Tabelle des Netzwerkplans vorgezeigt (siehe Abbildung 3.1), hat jede Lokation eine festgelegte Subnetz IP (***subnet***), Netzwerkmaske (***netmask***) und Router IP (***option routers***) und wird hier definiert. Um allen Hosts in den jeweiligen Subnetzen dynamisch Adressen zuzuweisen, muss innerhalb der Subnetz-Deklaration ein Bereich definiert werden (***range***). Anschließend wird definiert, an welches Netzwerkgerät der DHCP-Server verbunden werden soll (***INTERFACES***). Es kann auch die Gültigkeitsdauer definiert werden, die angibt wie lange einem Host die IP-Adresse zu Verfügung steht. Normalerweise reichen die Voreinstellungen aus, können aber mit ***default-lease-time*** und ***max-lease-time*** erhöhen oder herabsetzen, je nach Notwendigkeit. Anhand dieser Informationen, kann für jeden Standort ein Subnetz definiert werden.

3.10.5.2 Ergebnis

Nachdem das Mininet Skript ausgeführt wird, kann mit Wireshark geprüft werden, ob ein DHCP-Server richtig implementiert wurde. Aus der folgenden Abbildung 3.33 ist ersichtlich, dass die Pakete zwischen h1 (192.168.1.2) und h11 (192.168.2.2) übertragen werden.

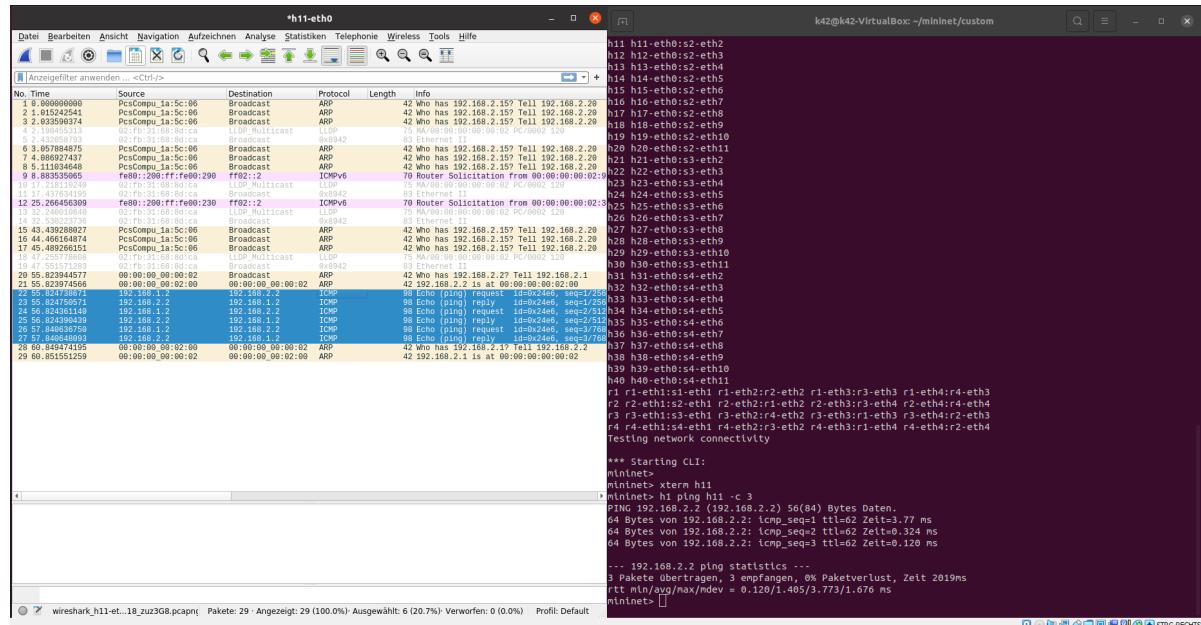


Abbildung 3.33: Datenverkehr zwischen h1 und h11

Um die Netzwerkkonfiguration vom DHCP-Server zu betrachten, muss Wireshark vor Mininet gestartet und beim Ausführen des Mininet Skripts die eine Switch betrachtet

3 Durchführung des Projektes

werden. In diesem Beispiel wird die Lokation Frankfurt betrachtet (siehe Abbildung 3.34).

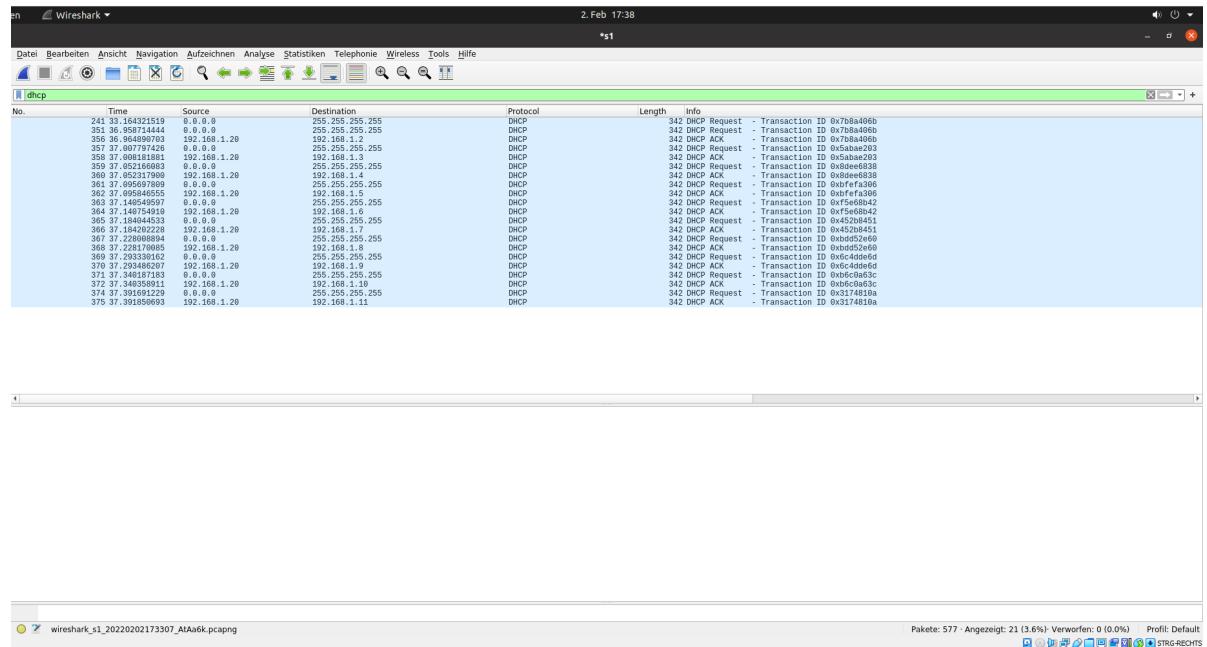


Abbildung 3.34: Wireshark-Aufnahme der Switch s1 (Frankfurt)

3.10.6 Domain Name System

4 Zusammenfassung

Analyse der Ergebnisse
Kritische Betrachtung
Hier erwähnen welche Teile der Aufgaben nicht geklappt haben, wie beispielsweise Aufgabe 5 Webproxy

4.1 Analyse der Ergebnisse

4.2 Kritische Betrachtung

5 Fazit

Eventuell als Überpunkt zu Gesamtergebnis? Was ist der finale Stand des Projekts?
Inwiefern wurden die Ziele erreicht?

5.1 Zukunftsaussichten

Inwiefern können die Ergebnisse des Projekts weiter genutzt werden?

Literaturverzeichnis

- [1] Bob Emmerson. *The Case for SDN*. 2012. URL: <https://www.nojitter.com/case-sdn> (besucht am 16.12.2021).
- [2] Nick Feamster Hyojoon Kim. „Improving Network Management with Software Defined Networking“. In: *IEEE Communications Magazine* (2013).