

Projektarbeit

Software-defined Networking mit Openflow

Mücahit Sagiroglu

Matrikelnummer: 1228852

James Belmonte

Matrikelnummer: 1340604

Naghmeh Ghavidel Rostami

Matrikelnummer: 1249307

Tung Trinh

Matrikelnummer: 1320718

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.

J. Belmonte

James Belmonte

Mücahit Sagiroglu

Mücahit Sagiroglu

Naghmeh Ghavidel

Naghmeh Ghavidel

Tung Trinh

Tung Trinh

Inhaltsverzeichnis

Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Listings	viii
Abkürzungsverzeichnis	ix
Abstract	x
1. Software-defined Networking	1
1.1. Motivation	9
1.2. Problemstellung	9
1.3. Aufbau der Arbeit	10
2. Projekt	11
2.1. Projektziel	11
2.2. Vorgehen	11
2.3. Festlegen von Meilensteinen	12
2.4. Verwendete Werkzeuge	13
3. Durchführung des Projektes	22
3.1. Netzwerkplan	22
3.2. Aufbau des Netzwerkgerüstes in Mininet	24
3.3. Verschlüsselung der Netzwerkverbindung zwischen den Lokationen	28
3.4. Auswahl des Service-Providers	31
3.5. Einrichtung des NAT-Firewalls	34
3.6. Implementierung der Webproxy-Funktion	37
3.7. Aufbau eines zentralen Topologie-Viewers und einer Monitoring-Lösung	39
3.8. Realisierung einer Quality of Service Funktion	44
3.9. Priorisierung der Datenübertragung über API	49
3.10. Analyse weiterer Netzwerkfunktionen	52
4. Diskussion der Ergebnisse	61
4.1. Analyse der Ergebnisse	61
4.2. Kritische Betrachtung	62

Inhaltsverzeichnis

5. Fazit	64
5.1. Zukunftsaussichten	66
Literaturverzeichnis	67
A. Anhang	73

Abbildungsverzeichnis

1.1.	SDN Architektur	1
1.2.	SDN Architektur	3
1.3.	SDN Architektur	5
2.1.	Zeitplan des Projektes	12
2.2.	Erstellung eines Standardnetzwerkes mit Mininet	16
2.3.	Webbenutzeroberfläche vom Floodlight-Controller	18
2.4.	Ausführung von Floodlight über den Linux-Terminal	21
2.5.	Mininet Controller Verbindung und Ping-Test	21
3.1.	Standortvernetzung mit der Any-to-Any Topologie	22
3.2.	Netzwerkplan aller Lokationen	23
3.3.	Netzwerk-Klasse zur Topologie Erstellung	24
3.4.	Erstellung und Verbindung von Router und Switch	25
3.5.	Erstellung und Verbindung von Hosts und Switch	25
3.6.	Verbindung und Konfigurierung der Router	26
3.7.	Erstellung eines RemoteController's	26
3.8.	Erstellung des Mininet-Objektes	26
3.9.	Ausführung und Testen vom Mininet-Skript	27
3.10.	Aufstellen der GRE-Tunnel	29
3.11.	Wandlung der Pakete bei IPSEC over GRE	29
3.12.	Konfiguration der Routen	30
3.13.	Erstellung der States für die Ver- und Entschlüsselung	30
3.14.	Erstellung der Policies für die Ver- und Entschlüsselung	31
3.15.	Verschlüsselter Verkehr zwischen Standort Frankfurt und Berlin	31
3.16.	VirtualBox NAT-Adapter	34
3.17.	Einbindung und Konfigurierung des NAT-Adapters	35
3.18.	Anfrage von H1 wird über Public IP des Routers durchgeführt	36
3.19.	Topologie des Netzwerkes auf der Webbenutzeroberfläche von Floodlight	39
3.20.	Hinzufügen der Switch IP für s1	40
3.21.	Definition der Switch für Nagios	42
3.22.	Definition der Services für die Switch	42
3.23.	Webbenutzeroberfläche von Nagios Core	43
3.24.	Ausgabe der QoS und Queue der Switches aller Lokationen	46
3.25.	Sprachanruf zwischen Berlin und Frankfurt	47
3.26.	Ausgabe der Flows von s1	47

Abbildungsverzeichnis

3.27. UDP-Pakete während dem Sprachanruf	48
3.28. Flows für die Priorisierung und die Drosselung	50
3.29. Abfrage der Daten für den Dateittransfer	51
3.30. Übermittlung der Datei über SSH	51
3.31. Statische IP-Adressenverwaltung	54
3.32. Dynamische IP-Adressenverwaltung	54
3.33. DHCP-Konfiguration für alle Standorte	55
3.34. Datenverkehr zwischen h1 und h11	56
3.35. Wireshark-Aufnahme von Switch s1	56
3.36. Kommunikation zwischen DHCP-Client und DHCP-Server	57
3.37. Wireshark-Aufnahme vom gekürzten DHCP Prozess	58
3.38. Kommunikation zwischen DHCP-Client und DHCP-Server	58
3.39. DHCP-Konfiguration für alle Standorte	59
3.40. Im DHCP ACK Protokoll werden die DNS Informationen vermittelt	60
3.41. Terminalausgabe von h1	60
5.1. Die Topologie zeigt, dass h0 in Frankfurt existiert	66

Tabellenverzeichnis

3.1. Vergabe von IPv4-Adressen im Netzwerk	23
3.2. DSL-Angebote verschiedener Internet-Service-Provider	32

Listings

A.1. Das in Python geschrieben Mininet-Skript	73
A.2. Das in Python geschrieben SSH-Transfer-Skript	87

Abkürzungsverzeichnis

- ADSL** Asymmetric Digital Subscriber Line
- API** application programming interface
- 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
- OF** OpenFlow
- 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

Abstract

Verfasst von: Tung

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 [14].

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 [22, S. 114–115].

1. Software-defined Networking

Im folgenden Abschnitt werden die jeweiligen Einleitungen jedes Mitgliedes vorgestellt. Sie dienen als separate Einleitungen zum Projektbericht.

Einleitung von James

Gegenwärtig gibt es im Bereich der Verwaltung von Computernetzwerken viele neue komplexe Anforderungen und notwendige Konfigurationen, die berücksichtigt werden müssen. Der Datenaustausch innerhalb eines Netzwerkes ist um ein Vielfaches gestiegen, somit steigt die Auslastung, welches ein Netzwerksystem aushalten muss. Bei einem traditionellen Netzwerk würde das heißen, dass ein Netzwerkadministrator alle Netzwerkkomponenten wie Router, Switches und Firewalls etc. manuell konfigurieren müsste, um allen entsprechenden Anforderungen zu verwirklichen. Ferner müssen Netzwerkadministratoren viel Zeit in die Konfiguration legen, um alles korrekt zu implementieren.

Software Defined Networking bietet hierzu eine Alternativmöglichkeit zur Verwaltung einer Netzwerkumgebung. Im Gegensatz zu traditionellen Netzwerken trennt SDN die Kontrollsicht und die Datenschicht, und ermöglicht dadurch die Kontrolle des Netwerkes über das Netz. Im Mittelpunkt des Netzwerkes befindet sich ein SDN-Controller, der zur Konfiguration von allen Netzwerkkomponenten genutzt wird [2]. Die SDN-Architektur ist auf drei Ebenen aufgeteilt und wird anhand von Abbildung 1.1 nochmals visuell dargestellt. Die Anwendungsschicht enthält Applikationen für SDN wie Firewall und Loadbalancer und wird mithilfe der REST API bereitgestellt [6]. Die Kontrollsicht enthält den SDN-Controller, worüber das gesamte Netzwerk gesteuert wird. Die Infrastruktursicht enthält die Netzwerkkomponenten wie Switches und Router und sind mit dem SDN-Controller verbunden, um die eingestellten Anforderungen vom Controller durchzusetzen. Die Architektur enthält mehrere Schnittstellen, sowohl eine Northbound API zwischen Anwendungsschicht und Kontrollsicht als auch eine Soutbound API zwischen

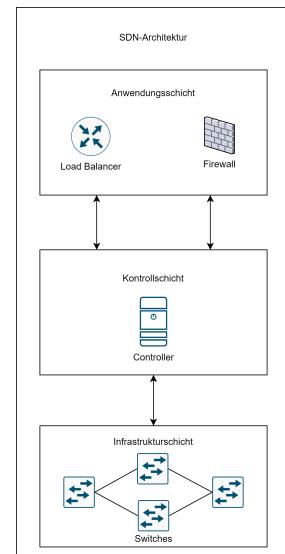


Abbildung 1.1.: SDN Architektur

1. Software-defined Networking

Kontrollsicht und Infrastruktursicht. Die Datenübertragung beim Southbound API erfolgt mit dem OpenFlow (OF) Protokoll. Bei der Weiterleitung eines Datenpakets wird die Weiterleitung auf der Datenschicht von einem Switch oder einem Router aufgenommen. Der SDN-Controller, der sich auf der Kontrollsicht befindet, entscheidet, wie das Datenpaket weitergeleitet wird [6]. Dadurch können verschiedene Netzwerkfunktionen implementiert werden, die von den Switches und Routern umgesetzt werden müssen. Die Kommunikation zwischen dem SDN-Controller und den Switches in der Infrastruktursicht wird durch Openflow realisiert. Durch die Änderung des flow tables im Switch, kann der Controller das Verhalten des Switches beeinflussen und so einstellen, dass die Instruktionen des Controllers umgesetzt werden [6]. Im Gegensatz zu der traditionellen Weise die Netzwerkkomponenten manuell zu konfigurieren, kann durch die Trennung von Kontrollsicht und Datenschicht der Controller genutzt werden, um alle Konfigurationen von Komponenten im Netzwerk umzusetzen [40]. Zudem kann durch das zentrale Management des Netzwerkes und das der SDN-Controller programmierbar ist, ein Administrator effizienter, flexibler und agiler Handeln [6]. Besonders im Bereich Quality of Service muss ein Netzwerk agiler und flexibler als in der Vergangenheit sein. Somit kann Quality of Service mit SDN gewährleistet und einfacher umgesetzt werden [60]. Logischerweise kann auch argumentiert werden, dass durch das zentrale Management über das Netz geringere Betriebskosten auftreten, da die Konfigurationsänderungen effizienter umgesetzt werden können. Jedoch ist ein erwähnenswerter Nachteil, dass beim Ausfall eines SDN-Controllers in einem Netzwerk das gesamte Netzwerk ausfällt. Dies könnte durch eine Denial of Service attack ausgelöst werden. Somit könnte die Möglichkeit bestehen, mehrere Ausweichcontroller für solche Ereignisse vorzubereiten.

Einleitung von Mücahit

Software-defined Networking beschreibt einen agilen Ansatz zur Verwaltung und Administrierung von Computernetzwerken aus einer zentralen Stelle. Dabei muss nur an der zentralen Stelle eine Änderung vorgenommen werden, um Änderungen an mehrere Netzwerkkomponenten beziehungsweise -bereichen vorzunehmen. Ein sogenannter Controller stellt eine zentrale Stelle dar und steuert den ihm zugehörigen Netzwerkabschnitt. Switches und Routern leiten Pakete nach Regeln beziehungsweise Tabellen weiter, wo üblicherweise für eine Änderung der Regeln jedes Gerät einzeln angesprochen werden muss. Durch die Software im Controller werden Änderungen angenommen und auf die Netzwerkkomponenten übertragen. Diese Vorgehensweise durch Software-defined Networking erlaubt eine dynamische Anpassung des Computernetzwerkes [2, S. 1–2].

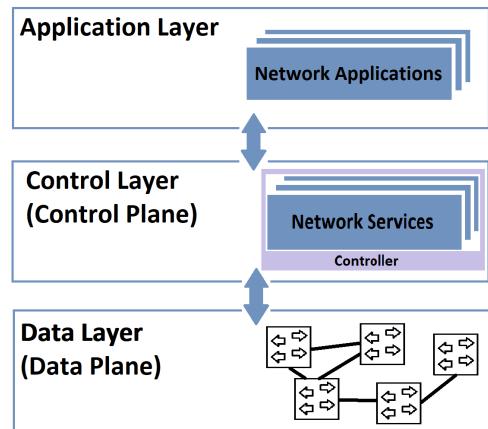


Abbildung 1.2.: SDN Architektur

Durch das Hinzufügen eines Controllers wird die sogenannte Control Plane vom Switch und Router entbunden. Dadurch verbleiben der Switch und der Router in der Data Plane, wo sie weiterhin für die Paketweiterleitung zuständig bleiben. Der Controller hingegen ist nun für die Entscheidungen verantwortlich, wie mit einem Paket verfahren werden soll. Aus der Entkopplung der Control Plane von der Data Plane ergibt sich im Ganzen die sogenannte Application Layer, die Data Plane und dazwischen die Control Plane, was nun Änderungen leichter zulässt (siehe Abbildung 1.2). Programme können über die application programming interface (API) mit dem Controller kommunizieren und ihm Anweisungen über Pakete oder Netzwerkbereiche geben. Der Controller wiederum wendet die getroffenen Entscheidungen an den zugehörigen Netzwerkkomponenten an [5, S. 2]. Des Weiteren kann über die API verschiedene Netzwerkleistungen implementiert werden. Darunter zählen neben der Kontrolle des Paketflusses beispielsweise Implementierungen bezüglich der Sicherheit, der Quality of Service (QoS), der Zugriffskontrolle und weitere Leistungen, die für ein Netzwerk wichtig sind [5, S. 8–9].

Bei klassischen Switches werden anhand eingehender Pakete MAC-Adressen mit dem dazugehörigen Ausgangsport in einer Tabelle hinterlegt. Fehlt diese Information, wird ein Paket auf allen verfügbaren Ports ausgegeben. Hierbei würde ein Switch keine Auslastungen außerhalb seines Bereiches kennen. Aus diesem Grund würde der Switch das Paket auf die für das Ziel vorgesehene Strecke einordnen, die er vorher per Address Resolution Protocol Anfrage (ARP) in seinem Cache abgelegt hat. Jedoch hat der Controller einen Überblick auf das ganze Netzwerk, das von ihm verwaltet wird, weshalb er eine performantere Entscheidung treffen kann, wohin das Paket am besten weitergeleitet werden sollte, um beispielsweise Überlastungen zu vermeiden [2, S. 1–2].

1. Software-defined Networking

Über ein im Vergleich zu Hypertext Transfer Protocol (HTTP) junges Protokoll namens OpenFlow kommuniziert der Controller mit bestimmten Netzwerkkomponenten wie beispielsweise ein OpenFlow Switch, um diese zu steuern. Es gibt zwei Arten von OpenFlow Switches: OpenFlow-only und OpenFlow-enabled Switches. OpenFlow-enabled Switches sind klassische Switches, die zusätzlich OpenFlow fähig sind. OpenFlow-only Switches haben keine zusätzlichen Module eingebaut, auf die Sie zurückgreifen können und müssen deswegen an einen Controller gebunden sein, der dann beispielsweise entscheidet, wohin eingehenden Pakete weiterleitet werden sollen. Dafür sendet der Switch ein Paket oder ein Teil eines Paketes an den Controller und wartet auf einen Befehl, wie er sich zu verhalten hat. Falls der Switch vorher einen Eintrag über das Paket besitzt, greift er auf diese zurück. Wenn überhaupt kein Eintrag vorhanden ist, wird das Paket nach den Einstellungen der Switch entweder verworfen, an allen Ports ausgegeben oder an den Controller gesendet. Die Fähigkeit, bestimmte Befehle auf Pakete anzuwenden, die auf einer Übereinstimmung basieren, zeichnet Software-defined Networking aus [5, S. 9–10].

Der Controller stellt einen sogenannten Single Point of Failure dar. Das bedeutet, wenn der Controller ausfällt, fallen alle vom Controller abhängigen Netzwerkkomponenten ebenfalls aus. So würde beispielsweise bei einer Denial-of-Service-Attacke durch das Überfluten des Controllers ein Zustand erfolgen, bei dem der Controller an Bandbreite, Speicher und Rechenleistung verlieren und im schlimmsten Fall auch lahmgelegt werden kann. Das ist eines der schlimmsten Szenarien und könnte durch das Hinzufügen eines zweiten Controllers kompensiert werden [30, S. 3].

Einleitung von Naghmeh

Kommunikationsnetzwerke werden immer größer und komplexer. Traditionelle Infrastrukturen und Netzwerksysteme können kaum eine Lösung bieten, um die heutigen Netzwerkanforderungen angemessen zu erfüllen. Dies hat zu einem anderen Ansatz für die Netzwerksystemarchitektur geführt, der als Software defined Networking bekannt ist. SDN-Ansätze wurden bereits Mitte der 1990er Jahre eingeführt, sind aber erst seit einigen Jahren ein etablierter Industriestandard geworden. Viele Netzwerkarchitekturen und -systeme haben SDN eingeführt, und Anbieter entscheiden sich für SDN als eine bessere Alternative zu festen und unflexiblen Protokollstapeln. SDN ist eine moderne Netzwerkarchitektur, die die Netzwerksteuerung von Datenübertragungsgeräten entkoppelt und die Netzwerkprogrammierung unterstützt [18].

SDN trennt die Netzarchitekturen in drei verschiedene Ebenen: Daten-, Kontroll- und Anwendungsebene. Die Datenebene umfasst alle Routing-Geräte, die Weiterleitungsregeln speichern, um den Datenverkehr zu verarbeiten. Die Kontrollebene besteht aus einem oder mehreren Controllern. Der Controller ist dafür verantwortlich, den Datenverkehr zu überprüfen, Entscheidungen zu treffen und die Netzwerkkonfiguration zu ändern. Der Controller kann auch die Ressourcen optimieren und die Fehlerbehebungsverfahren implementieren. Die Anwendungsebene gruppiert Anwendungen und verarbeitet die Daten für den Controller. SDN definiert auch Schnittstellen für die Kommunikation zwischen Ebenen: Anwendungs- und Kontrollebene kommunizieren über die Northbound API, und Daten- und Kontrollebenen verwenden die Southbound-API [56].

Die SDN-Architektur hängt stark von der Effizienz der Kontrollebene ab, auf der sich der SDN-Controller befindet. Um die Kontrollebene möglichst effizient zu gestalten, hat die Auswahl des Controllers eine große Bedeutung. Der Controller soll sowohl weniger Zeit benötigen, um Entscheidungen zu treffen und Entscheidungen an die Datenebene weitergeben, als auch die Herausforderungen des modernen Netzwerkverkehrs bewältigen[56].

Das von SDN eingeführte modulare Konzept und die moderne Schichtenstruktur ermöglichen jeder Schicht, unabhängig voneinander zu innovieren, was den Herstellern von Netzwerkgeräten neue Möglichkeiten eröffnet, moderne Netzwerkdienste und -anwendungen zu entwickeln. Netzwerkadministratoren können die Netzwerkleistung und das Netzwerkverhalten in Echtzeit überwachen. Sie können auch Entscheidungen über die Bereitstellung von Netzwerkdiensten oder Änderungen und Konfiguration der Topologie von ihrem Arbeitsplatz aus treffen und diese durchführen [21].

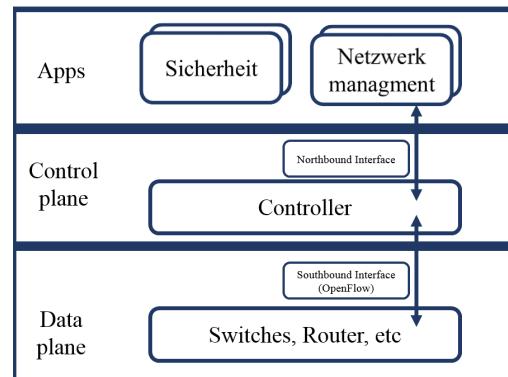


Abbildung 1.3.: SDN Architektur

1. Software-defined Networking

SDN eignet sich auch ideal für die Entwicklung von Mechanismen zur Bewältigung von Sicherheitsproblemen, aber dazu müssen die Administratoren die Controller für die Ausführung bestimmter Aufgaben erweitern. Diese Art von Mechanismus kann auf der Kontrollebene oder als eine Reihe von Teilen auf der Anwendungs- und Kontrollebene ausgeführt werden [44].

Die wichtigsten Funktionen, die SDN für Computernetzwerke bietet, sind die Möglichkeiten, das Netzwerk zu programmieren, es flexibler und einfacher zu verwalten. Diese Fähigkeit spielt eine wichtige Rolle bei der Reduzierung des Gesamtsystemkapitals und der Verwaltungskosten sowie bei der Verbesserung von Leistung, Zuverlässigkeit und Servicequalität. Die ständige Nachfrage nach Neukonfiguration bestehender Netzwerke aufgrund von Bandbreite und neuen Technologien (Hard- und Software) ist sehr anspruchsvoll und komplex. Mit der SDN-Technologie wird dieser Prozess im Vergleich zu herkömmlichen Netzwerken vereinfacht[21].

Einleitung von Tung

Derzeit gibt es viele Definitionen von SDN, aber laut Open Networking Foundation (ONF), ist SDN definiert als: „Software-Defined Networking ist eine aufstrebende Architektur, die dynamisch, verwaltbar, kostengünstig und anpassungsfähig ist, wodurch sie sich ideal für die hohe Bandbreite und Dynamik heutiger Anwendungen eignet. Diese Architektur entkoppelt die Netzwerksteuerungs- und Weiterleitungsfunktionen, wodurch die Netzwerksteuerung direkt programmierbar und die zugrunde liegende Infrastruktur für Anwendungen und Netzwerkdienste abstrahiert werden kann.“[51] Diese Definition kann sehr allgemein und verwirrend sein. Um zu verstehen, was SDN ist, erinnern wir uns zunächst ein wenig an die traditionelle Netzwerkarchitektur.

Andreas Donner beschreibt das herkömmliche Netzwerk: Konventionelle Netzwerke bestehen aus Routern und Switches, die mit proprietärer, herstellerspezifischer Firmware betrieben werden. Die Datenpakete werden von den Komponenten gemäß den Vorgaben der Software weitergeleitet. Die Netzwerkgeräte besitzen intern spezielle Hardware, die für die Weiterleitung der Pakete verantwortlich ist. Die Steuerung der Hardware übernimmt die Software und Kontrollebene des Betriebssystems, das ebenfalls auf dem Gerät integriert ist. [13] Es bedeutet, jedes Gerät im Netzwerk verfügt sowohl eine logische Rolle in der Steuerleitung als auch die Weiterleitung von Datenpaketen. Im traditionellen Netzwerk muss der Administrator bei der Konfiguration eines Mehrkomponentennetzwerks jedes Gerät manuell konfigurieren, und die Geräte sind in Bezug auf die Verwaltung nicht miteinander verbunden. Diese sind besonders schwierig zu konfigurieren und zu bedienen. Die Geräte sind völlig unabhängig, sodass die Konfiguration an jedem Gerät nur manuell nacheinander geändert werden kann.

Die SDN-Architektur hat drei Ebenen, die über Northbound- und Southbound-Anwendungsprogrammierschnittstellen kommunizieren. Die Ebenen umfassen:

- Application Plane – SDN-Anwendungen kommunizieren Verhalten und benötigte Ressourcen mit dem SDN-Controller.
- Control Plane – Verwaltet Richtlinien und Verkehrsfluss. Der zentralisierte Controller verwaltet das Verhalten der Datenebene.
- Data Plane – Besteht aus den physischen Switches im Netzwerk.[4]

Mit der obigen Architektur bietet SDN verschiedene Möglichkeiten. Die Control Plane kann direkt programmiert werden. Das Netzwerk wird durch Änderungen auf der Control Plane schnell angepasst und geändert. Das Netzwerk wird zentral verwaltet, da die Steuerung auf der Control Plane zentralisiert ist. Die Konfigurationen der Data Plane können auf der Application Plane programmiert und an die unteren Schichten kommuniziert werden.

Netzwerkgeräte haben nur die Funktion, Daten weiterzuleiten, und der Routing-Teil wurde auf ein anderes Gerät übertragen, das als Controller für das gesamte Netzwerk fungiert und als SDN-Controller bezeichnet wird. Es enthält alle Routing-Algorithmen,

1. Software-defined Networking

Safety-, Security- und Load-Balancing-Lösungen. Bei der Konfiguration des gesamten Netzwerks muss der Administrator nur die Algorithmen auf diesem Gerät mit Software schreiben und ändern, alle Netzwerkgeräte werden synchron verwaltet. Wenn Benutzer Parameter oder Anforderungen für das Netzwerk ändern möchten, ändern sie diese einfach in der Verwaltungssoftware des SDN-Controllers. Dies ist völlig anders als herkömmliche Netzwerktechnologien [63, S. 37–38].

Mit SDN kann der Administrator des Netzwerks alle Switching-Regeln des Netzwerks nach Bedarf ändern. Administratoren können bestimmte Pakettypen mit einem bestimmten Maß an Kontrolle und Sicherheit priorisieren, depriorisieren oder sogar blockieren. Dies ist besonders nützlich in Cloud-Computing-Architekturen, da es Administratoren ermöglicht, den Datenverkehr flexibler und effizienter zu verwalten, wenn es mehrere Dienstmandanten gibt. Im Wesentlichen ermöglicht dies Administratoren, Switches effizienter zu nutzen und mehr Kontrolle über den Netzwerkverkehr zu haben als je zuvor. [20]

Weitere Vorteile von SDN sind, dass es Verwaltung und Einblick in die Komponenten und Konfiguration des Netzwerks bietet. Administratoren müssen nur mit einem einzigen zentralen Controller interagieren, um Richtlinien an verbundene Switches im Netzwerk zu verteilen, anstatt mehrere Geräte einzeln konfigurieren zu müssen. Diese Fähigkeit ist auch in Bezug auf die Sicherheit ein großer Vorteil, da der Controller den Datenverkehr überwachen und Sicherheitsrichtlinien zentral, synchron und einfach implementieren kann. Wenn der Controller beispielsweise verdächtigen Datenverkehr sieht, kann er das Paket sofort umleiten oder verwerfen, ohne es an einen anderen Handler weiterzuleiten.[23]

1.1. Motivation

Verfasst von: James

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.

1.2. Problemstellung

Verfasst von: James

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

1. Software-defined Networking

sowohl eine graphische Darstellung der Netzwerkstruktur durch einen Topologieweiter 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.

1.3. Aufbau der Arbeit

Verfasst von: James

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

Verfasst von: Tung

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

Verfasst von: Tung

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

2. Projekt

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

Verfasst von: Tung, Naghmeh, James, Mücahit

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:

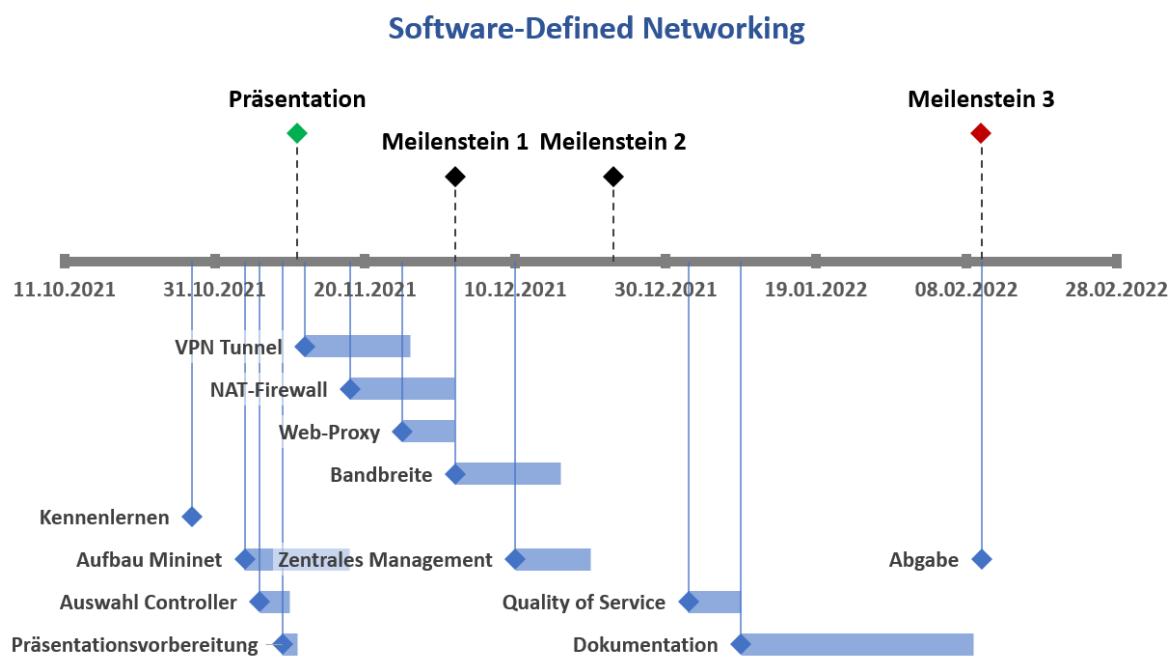


Abbildung 2.1.: Zeitplan des Projektes

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

Verfasst von: Naghmeh

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

Verfasst von: Tung

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 [43].

2.4.1.1. Einführung

Verfasst von: Tung

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 [28, S. 139]. 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[28, S. 141].

2.4.1.2. Funktionalität

Verfasst von: Tung

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 [43].

2.4.1.3. Nachteile

Verfasst von: Tung

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 [69].

2.4.1.4. Komponenten

Verfasst von: Tung, Naghmeh, James, Mücahit

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.

2. Projekt

- **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

Verfasst von: Tung

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 [42]:

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
```

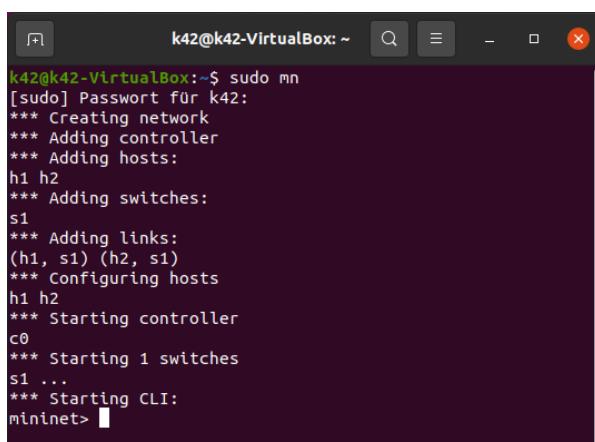
3. Mit install.sh die Installation starten

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

2.4.1.6. Aufbau

Verfasst von: Tung

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



```
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. Projekt

2.4.2. Floodlight

Verfasst von: Mücahit, Tung

Floodlight ist ein sogenannter SDN-Controller in der Control Plane. Dieser kommuniziert mit der Data Plane über ein Kommunikationsprotokoll namens OpenFlow und verwaltet diesen [54, S. 161].

2.4.2.1. Einführung

Verfasst von: Mücahit, Tung

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 sollten alle Projektziele der Projektarbeit erreichbar sein [61].

Verfasst von: Mücahit

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

Verfasst von: Mücahit

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 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 [45].

2. Projekt

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 [25].

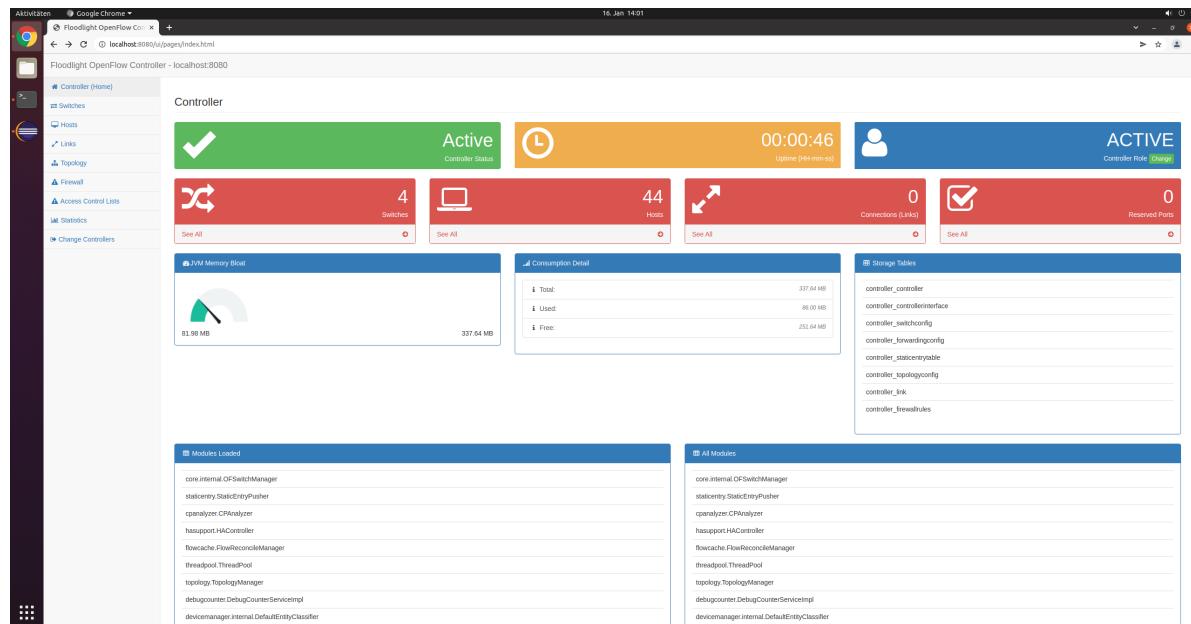


Abbildung 2.3.: Webbenutzeroberfläche vom Floodlight-Controller

2.4.2.3. Nachteil

Verfasst von: Mücahit

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 [9].

2.4.2.4. Komponenten

Verfasst von: Mücahit

Mit der Installation von Floodlight kommen sogenannte Module zum Einsatz. Die meisten der Module sind bereits aktiviert und bewältigen bestimmte Netzwerkaufgaben. Einer der Module ist der 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 das Paket als erstes an alle Netzwerkteilnehmern geschickt. Wenn der Zielhost antwortet, speichert der Switch die MAC-Adresse mit der jeweiligen Route in einer Tabelle ab. Wenn erneut ein Paket zum gleichen Empfänger will, wird über die Tabelle der Ausgangs-Port gelesen und das Paket darüber weitergeleitet [78]. Weitere Beispielmodule wären der Load Balancer, der für einen Ausgleich des Datenverkehrs im gesamten Netzwerk sorgt [24]. Über die REST-API stellt Floodlight die Netzwerktopologie über die Webbenutzeroberfläche grafisch dar. Es existieren noch weitere Module, wobei auch eigene programmiert werden können. Damit könnte das Netzwerk dynamischer gestaltet werden [26].

2.4.2.5. Installation

Verfasst von: Mücahit

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 [77].

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
```

Verfasst von: James

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, 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 ***Next*** klicken. Von ***Select root directory*** den ***Browser*** und das 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 [77].

2.4.2.6. Aufbau

Verfasst von: Mücahit

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

2. Projekt

```

k42@k42-VirtualBox:~/mininet/custom$ 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
ns' 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.l.LinkDiscoveryManager] Link latency history set to 10 LLDP data points
2022-01-31 20:43:21.706 INFO [n.f.l.l.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=tr
ue, 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.l.SyncManager] [1] Updating sync configuration ClusterConfig [allNodes={1=Node [hostname=192.168.56.1, port=6642, n
odeId=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, keyStorePassw
ord is set]

```

Abbildung 2.4.: Ausführung von Floodlight über den Linux-Terminal

2.4.3. Ergebnis

Verfasst von: Tung

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
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

Verfasst von: James

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

Verfasst von: Tung, Naghmeh, James, Mücahit

Die topologische Struktur des Netzwerks, die in Mininet erstellt wird, ist in Abbildung 3.2 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.

Darüber hinaus wurde für den Netzwerkplan die Standleitung in Abbildung 3.1 dargestellt. Dabei wurde eine Any-to-Any Topologie für die Vernetzung ausgewählt. Diese Topologie bietet ein hohes Maß an Zuverlässigkeit und Fehlertoleranz, was jedoch mit einem hohen Preis verbunden sein kann.

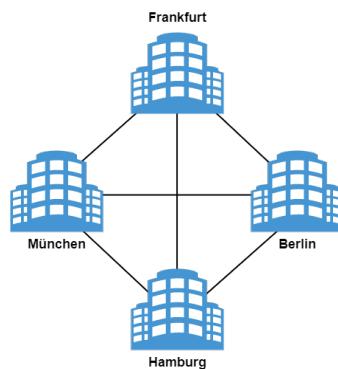


Abbildung 3.1.: Standortvernetzung mit der Any-to-Any Topologie

3. Durchführung des Projektes

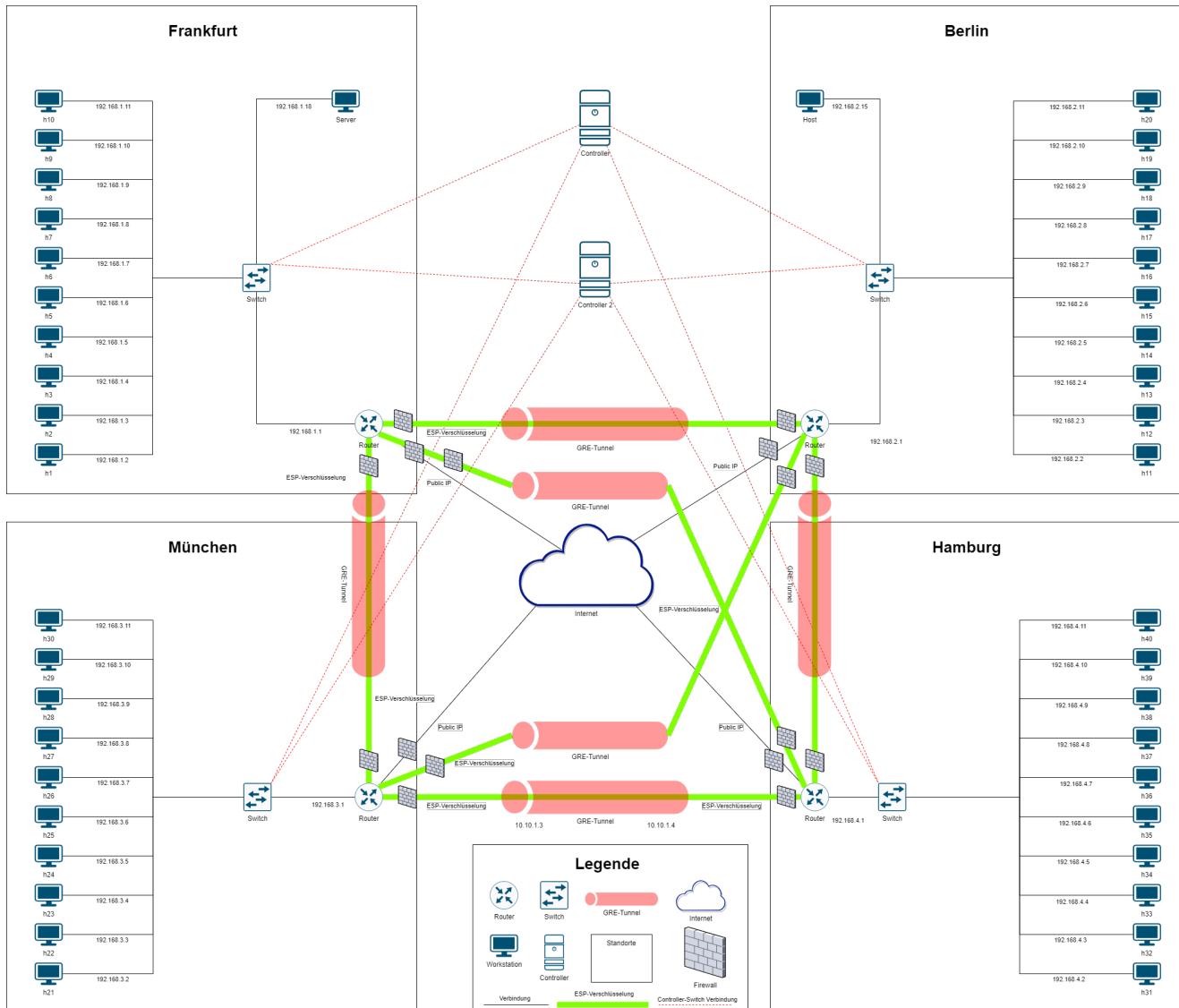


Abbildung 3.2.: Netzwerkplan aller Lokationen

Tabelle 3.1.: Vergabe von IPv4-Adressen im Netzwerk

Standort	Frankfurt	Berlin	München	Hamburg
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	192.168.1.254	192.168.2.254	192.168.3.254	192.168.4.254

3. Durchführung des Projektes

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

Verfasst von: Tung

In der Main-Funktion werden die Komponenten eines Netzwerks deklariert und aufgerufen. Das sind eine Topologie, ein Controller mit zugewiesenem 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

Verfasst von: Tung, Mücahit

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 3.3). 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 = []
```

Abbildung 3.3.: Netzwerk-Klasse zur Topologie Erstellung

3. Durchführung des Projektes

Im nächsten Teil der Klasse Netzwerk, werden die Komponenten des Netzwerks implementieren. Dies ist mit Hilfe einer Schleife mit 4 Durchläufen ausgeführt worden. Jeder 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 3.4).

```
# 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 3.4.: 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 3.5). 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.24' % (r+1, h+2),
                        defaultRoute='via 192.168.%s.1' % (r+1), mac='00:00:00:00:00:0%s' % (r+1, h))
    self.addLink(host, switch)
```

Abbildung 3.5.: 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

3. Durchführung des Projektes

Megabit gesetzt, wobei dies die geforderte SDSL-Leitung im Kapitel 3.8 darstellen soll (siehe Abbildung 3.6).

```
# 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 3.6.: Verbindung und Konfigurierung der Router

3.2.3. Controller Implementierung

Verfasst von: Tung, Mücahit

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 3.7). 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 [19].

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

Abbildung 3.7.: 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 3.8). 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 [74].

```
# 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 3.8.: Erstellung des Mininet-Objektes

3. Durchführung des Projektes

3.2.4. Ergebnis

Verfasst von: Tung

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 3.9).

Abbildung 3.9.: Ausführung und Testen vom Mininet-Skript

3. Durchführung des Projektes

3.3. Verschlüsselung der Netzwerkverbindung zwischen den Lokationen

Verfasst von: Mücahit

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

Verfasst von: Mücahit

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 [52]. 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 [16].

3.3.2. Durchführung

Verfasst von: Mücahit

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))*** ausgeben und ausgeführt wird (siehe Abbildung 3.10). 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 [65].

3. Durchführung des Projektes

```
# 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 3.10.: 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 [15] (siehe Abbildung 3.11). 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 [10].

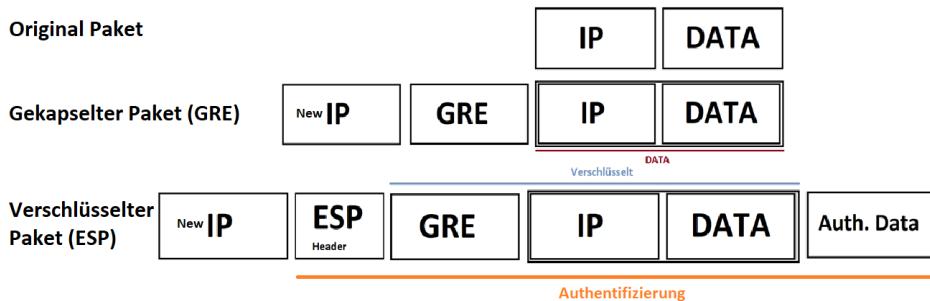


Abbildung 3.11.: Wandlung der Pakete bei IPSEC over GRE

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 [8] (siehe Abbildung 3.12). 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.

3. Durchführung des Projektes

```
# 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 3.12.: 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 [29]. 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 3.13).

```
# 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 3.13.: 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 3.11 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 3.14). Es wird nicht die Tunnel-IP, sondern die Router-IP angegeben [64].

3. Durchführung des Projektes

```
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 3.14.: Erstellung der Policies für die Ver- und Entschlüsselung

3.3.3. Ergebnis

Verfasst von: Mücahit

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 [32]. Zwischen den Lokationen war vorher der gesamte Datenverkehr ersichtlich, wohingegen jetzt alles verschlüsselt ist (siehe Abbildung 3.15).

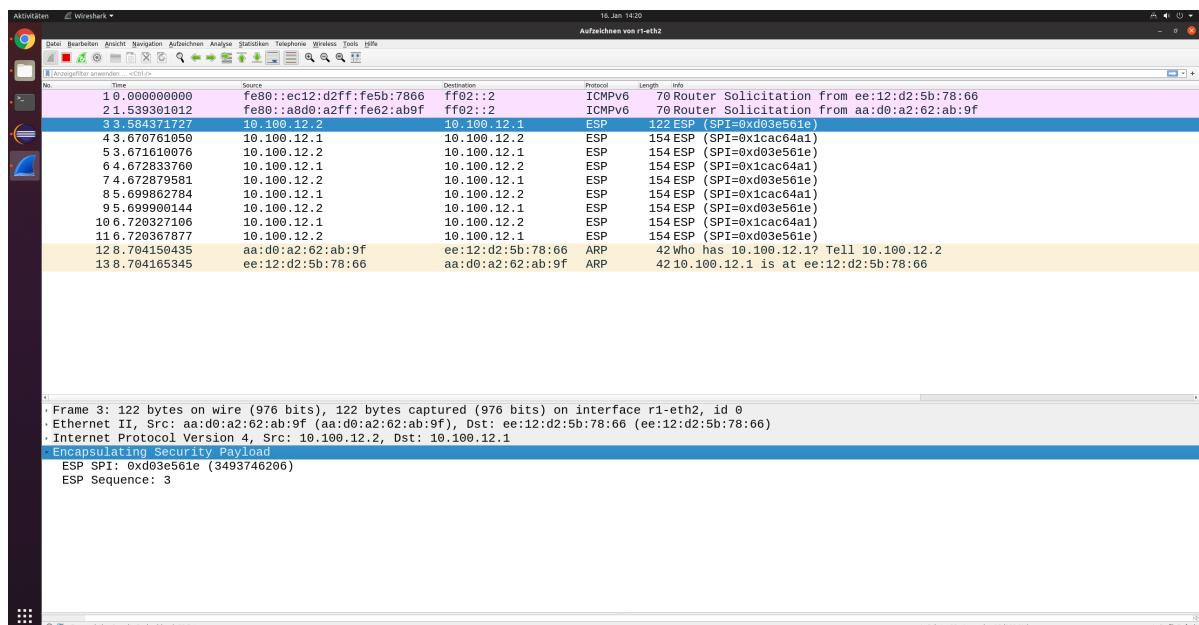


Abbildung 3.15.: Verschlüsselter Verkehr zwischen Standort Frankfurt und Berlin

3.4. Auswahl des Service-Providers

Verfasst von: Tung, Naghme, James, Mücahit

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.

3. Durchführung des Projektes

Im Folgenden sind in einer Tabelle die Internet-Service-Provider O2, Vodafone, 1&1 und Telekom mit verschiedenen Bandbreiten und den dazugehörigen Nettopreisen aufgelistet [50] [72] [1] [68] (siehe Tabelle 3.2). Die Angebote sind aktuell und aus den originalen Webseiten der Provider entnommen worden.

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
	DSL 100	54 / 88,6 / 100	20 / 36,9 / 40	25 Euro
1&1	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
	Plus 100 Regio DSL	54 / 87 / 100	20 / 37 / 40	25 Euro
O2	Plus 250 Regio DSL	175 / 210 / 250	20 / 37 / 40	30 Euro
	MyOffice S	0,3 / 8 / 10	0,3 / 1,5 / 2	25 Euro
	MyOffice M	3 / 38 / 50	0,7 / 8 / 10	27,5 Euro
	MyOffice L	50 / 83 / 100	10 / 33 / 40	30 Euro
	MyOffice XL	105 / 200 / 250	12 / 33 / 40	35 Euro

3. Durchführung des Projektes

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 Preisleistungsverhältnis ist der Telekom Company Start 500 [67] Angebot mit einer garantierteren Uploadrate von mindestens 80 Megabit und einer garantierteren 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 [46].

3.5. Einrichtung des NAT-Firewalls

Verfasst von: Mücahit, Naghmeh

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 [33]. 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

Verfasst von: Mücahit, Naghmeh

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 Domain Name System (DNS) Server konfiguriert bekommen, damit die Hosts nicht nur über die IP-Adressen aufs Internet zugreifen.

3.5.2. Durchführung

Verfasst von: Mücahit, Naghmeh

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 3.16). 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

Adapter 5: Intel PRO/1000 MT Desktop (NAT)
Adapter 6: Intel PRO/1000 MT Desktop (NAT)
Adapter 7: Intel PRO/1000 MT Desktop (NAT)
Adapter 8: Intel PRO/1000 MT Desktop (NAT)

Abbildung 3.16.: VirtualBox NAT-Adapter

3. Durchführung des Projektes

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 3.17). Der IPv4-Adressenbereich, der vom NAT-Service vergeben wird, liegt bei **10.0.X.X/24** und ändert sich je nach virtueller Maschine [71] [41].

```
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 3.17.: 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 dieser 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 [70]. 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 [31].

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 [59]. 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 [53] (siehe Abbildung 3.16). 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 [58].

3. Durchführung des Projektes

3.5.3. Ergebnis

Verfasst von: Mücahit, Naghmeh

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 3.18).

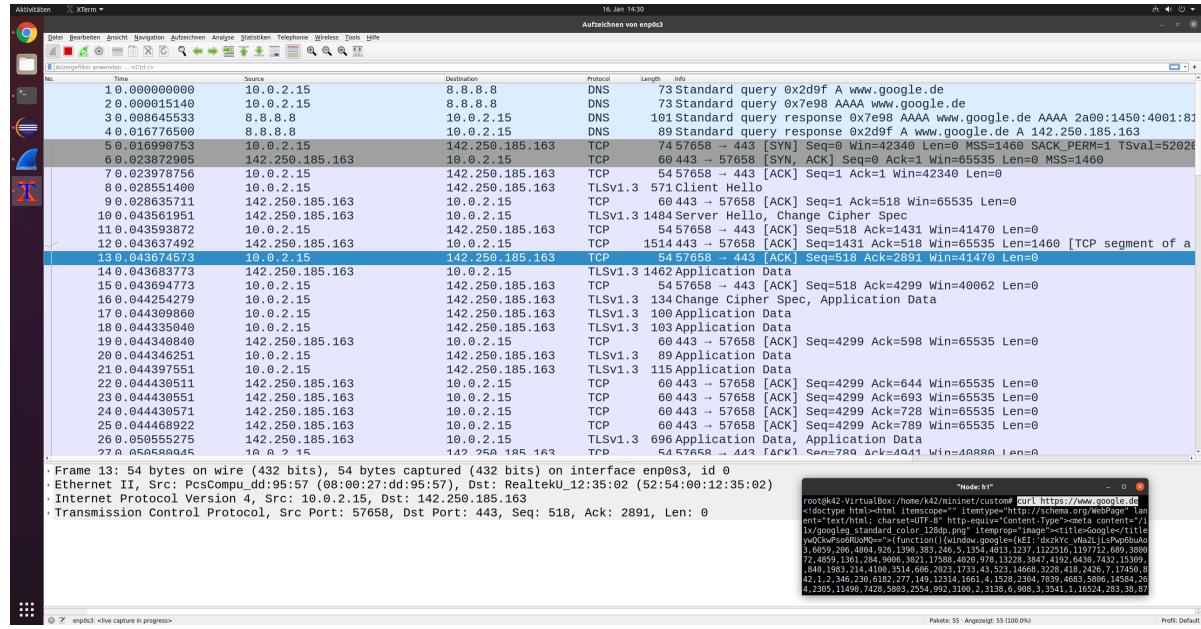


Abbildung 3.18.: Anfrage von H1 wird über Public IP des Routers durchgeführt

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. Durchführung des Projektes

3.6. Implementierung der Webproxy-Funktion

Verfasst von: Tung, Naghmeh, James, Mücahit

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 [57].

3.6.1. Vorüberlegung

Verfasst von: Tung, Naghmeh, James, Mücahit

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

Verfasst von: Tung, Naghmeh, James, Mücahit

Um die Webproxy-Funktion zu realisieren, wurde in allen Lokationen jeweils ein weiterer Host in Mininet konfiguriert. Es wurde allen eine IPv4-Adresse im jeweiligen 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

3. Durchführung des Projektes

sollte eine Anfrage eines Hosts über den Proxy-Server-Host 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 kam keine Verbindung zwischen den zwei VirtualBox-Maschinen zustande. Aus diesem Grund wurde ein alternativer Proxy-Skript aus Github benutzt, um die Webproxy-Funktionalität direkt auf dem Host beziehungsweise Server einzurichten. Dazu wurde per Xterm ein Host-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

Verfasst von: Tung, Naghmeh, James, Mücahit

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 [62].

3. Durchführung des Projektes

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

3.7.1. Topologie

Verfasst von: Mücahit

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 [26]. Anschließend wird auf dem linken Reiter die Option Topology angeklickt, wodurch die Topologie angezeigt wird (siehe Abbildung 3.19). Durch die Benutzung einer VirtualBox 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.

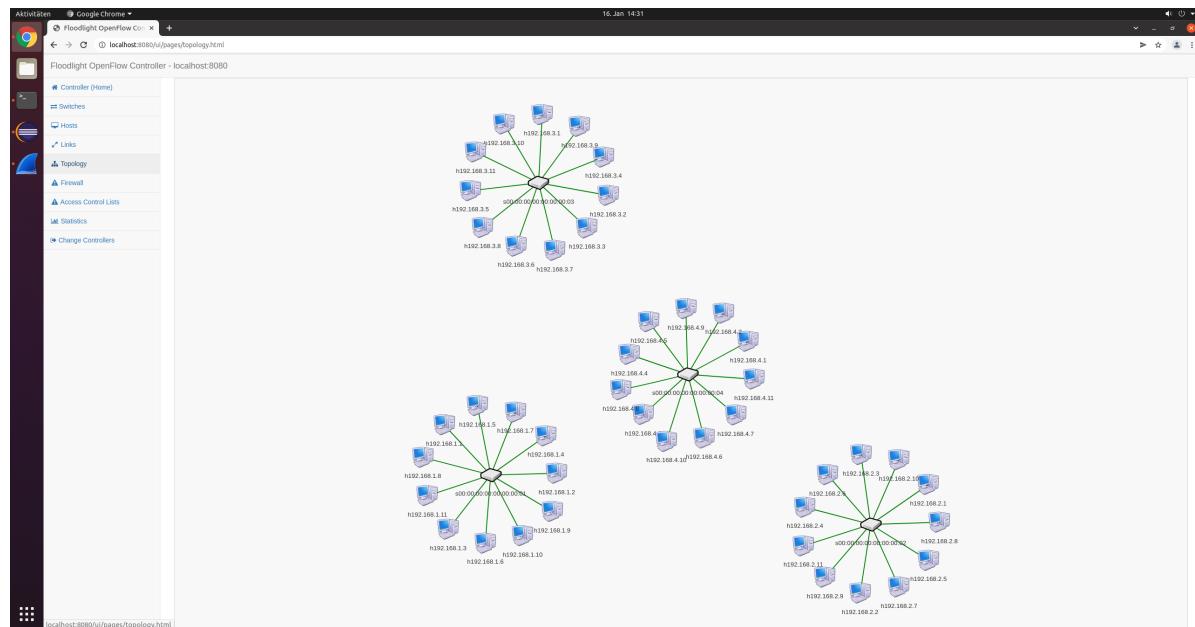


Abbildung 3.19.: Topologie des Netzwerkes auf der Webbenutzeroberfläche von Floodlight

3. Durchführung des Projektes

3.7.2. Monitoring-Lösung

Verfasst von: Mücahit

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 Netzwerkteilnehmern erreicht werden kann. Dabei kann die Verfügbarkeit, 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 [48].

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 3.20). 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 3.20.: 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
```

3. Durchführung des Projektes

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-apache-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
```

3. Durchführung des Projektes

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 [27]. 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 3.21). Danach können sogenannte Services für die Überwachung der Verfügbarkeit, der Geschwindigkeit und weitere Informationen hinzugefügt werden (siehe Abbildung 3.22). 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 [11].

```
# 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 3.21.: 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 3.22.: Definition der Services für die Switch

Nachdem alle Netzwerkeinheiten und -komponenten hinzugefügt und die gewünschten Services eingestellt worden sind, werden alle Informationen auf der Webbenutzeroberfläche angezeigt (siehe Abbildung 3.23). Unter dem Reiter Hosts kann die Verfügbarkeit aller Teilnehmer angezeigt werden. Die Verfügbarkeit kann unter mehreren Zuständen

3. Durchführung des Projektes

unterscheiden. Ein Host kann daher den Zustand **Up**, **Down**, **Unreachable** und **Pending** haben, welche auch mit Farben visualisiert werden. Unter Services werden alle Teilnehmer mit den konfigurierten Services angezeigt. Die Services haben ebenfalls Zustände. Dazu gehört unteranderem **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.

The screenshot shows the Nagios Core web interface at <http://192.168.1.20/nagios/>. The main page displays the 'Current Network Status' and 'Service Status Totals'. It lists hosts under 'Host Groups' such as 'HOST-BUILDN', 'RZ-BEIN', and 'SERV-FRA'. Each host entry includes service details like 'Current Load', 'CPU Usage', 'Memory Usage', and 'Disk Usage'. The 'Status' column indicates the overall status of each service (e.g., OK, CRITICAL). The 'Last Check' column shows the last time the service was checked. The 'Duration' column shows the time since the last check. The 'Attempts' column shows the number of attempts made. The 'Status Information' column provides detailed status information for each service. A legend at the bottom right indicates the color coding for different service states: OK (green), Warning (yellow), Unknown (orange), Critical (red), and Pending (grey).

Abbildung 3.23.: Webbenutzeroberfläche von Nagios Core

3. Durchführung des Projektes

3.8. Realisierung einer Quality of Service Funktion

Verfasst von: Mücahit

Die Quality of Service Funktion gewährleistet die Fähigkeit des Netzwerkes, Anwendungen und Datenverkehr selbst bei begrenzter Netzwerkkapazität zuverlässig mit hoher Priorität auszuführen [66]. 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

Verfasst von: Mücahit

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 [12].

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 [3, S. 952]. Nach der Konfiguration sollte gezeigt werden, ob Pakete tatsächlich über die Queue verlaufen.

3.8.2. Durchführung

Verfasst von: Mücahit

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

3. Durchführung des Projektes

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 [75]. Der Verkehr soll entweder auf die **Queue 0** oder **Queue 1** weitergeleitet werden. So erhält 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

Verfasst von: Mücahit

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 3.24). 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          : aba236b8-768e-4301-8473-0a4518334434
external_ids   : []
other_config   : [max-rate="20000000"]
queues         : [{0=a7fb620-b568-45ad-a932-bfff115c29ad, 1=8260945e-4c06-4a9d-aa34-178e0527755d}]
type           : linux-htb

_uuid          : 5d0cdccb-6157-4791-bdbc-61ce782b289c
external_ids   : []
other_config   : [max-rate="20000000"]
queues         : [{0=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         : [{0=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         : [{0=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"]

```

Abbildung 3.24.: 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 3.25). 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 3.26). Per Wireshark können die zwischen den Standorten Übermittelten UDP-Pakete betrachtet werden (siehe Abbildung 3.27). 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.

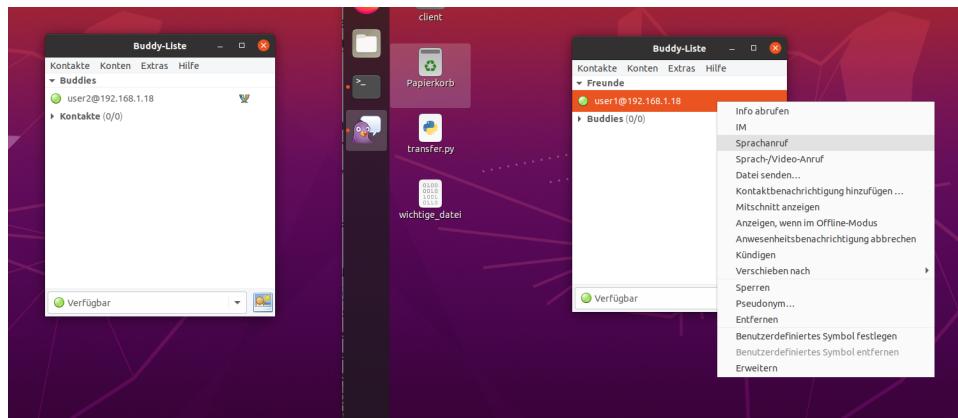


Abbildung 3.25.: Sprachanruf zwischen Berlin und Frankfurt

A screenshot of a terminal window titled 'k42@k42-VirtualBox: ~/mininet/custom'. The command entered is 'sudo ovs-ofctl dump-flows s1'. The output shows two flow entries: 'cookie=0x0, duration=278.397s, table=0, n_packets=3109, n_bytes=385308, priority=65535, udp,nw_src=192.168.0.0/16,nw_dst=192.168.0.0/16 actions=set_queue:1,NORMAL' and 'cookie=0x0, duration=278.408s, table=0, n_packets=83, n_bytes=8083, priority=0 actions=CONTROLLER:65535'. The terminal has a dark theme with white text and a black background.

Abbildung 3.26.: Ausgabe der Flows von s1

3. Durchführung des Projektes

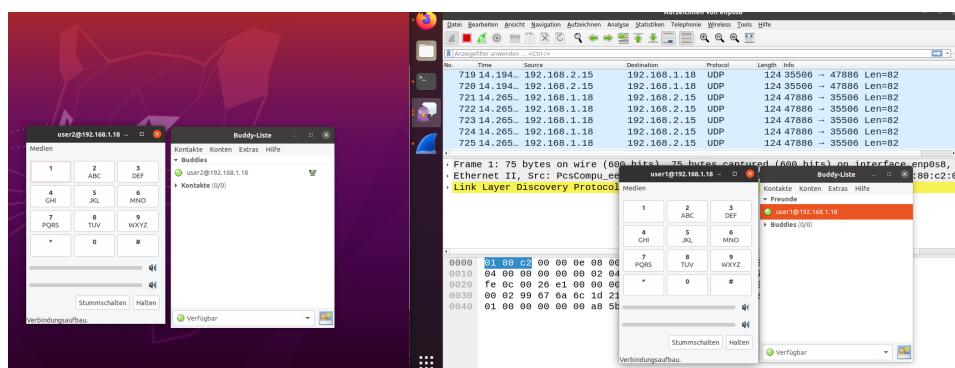


Abbildung 3.27.: UDP-Pakete während dem Sprachanruf

3.9. Priorisierung der Datenübertragung über API

3.9.1. Vorüberlegung

Verfasst von: Mücahit

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 [76]. 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. Durchführung des Projektes

3.9.2. Durchführung

Verfasst von: Mücahit

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 eingesetzt, 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 3.29). 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 Ethernettyp auf IPv4, die Priorität auf ***100*** und die Queue auf 2 gesetzt (siehe Abbildung 3.28). Die Queue

```
# Priorisierung und Drosselung
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"
}

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"
}

dros1={
    "switch": "00:00:00:00:00:00:00:01",
    "name": "dros1",
    "priority": "1050",
    "eth_type": "0x0800",
    "active": "true",
    "actions": "set_queue=2,output=normal"
}
```

Abbildung 3.28.: Flows für die Priorisierung und die Drosselung

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 3.29). 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 Ethernettyp auf IPv4, die Priorität auf ***100*** und die Queue auf 2 gesetzt (siehe Abbildung 3.28). Die Queue

3. Durchführung des Projektes

2 wurde vorher mit Maximal 2 Megabit eingestellt. Auch ist die Priorität im Vergleich 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 3.30). 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 3.29.: 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 3.30.: Übermittlung der Datei über SSH

3.9.3. Ergebnis

Verfasst von: Mücahit

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. Durchführung des Projektes

3.10. Analyse weiterer Netzwerkfunktionen

3.10.1. Hub

Verfasst von: Naghmeh

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 [17].

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 [55]. Aus diesem Grund wurde keinen Hub oder Repeater in das untersuchende Netzwerk verwendet.

3.10.2. Bridge

Verfasst von: Naghmeh

Eine Bridge in einem Computernetzwerk ist eine Art von Netzwerkgerät, das dazu dient, ein Netzwerk in Abschnitte zu unterteilen. Die Bridges sind dann erforderlich, wenn jedoch Frames von einem physikalischen Netzwerk zu einem anderen weitergeleitet werden sollen. Eine Bridge hat nur zwei Anschlüsse und arbeitet auf Schicht 2 im OSI-Modell, der Datenverbindungsschicht [7].

Das Funktionsprinzip einer Bridge besteht darin, dass sie die Daten abhängig von der MAC-Zieladresse blockiert oder weiterleitet, und diese Adresse wird in jeden Datenrahmen. Eine Bridge verwendet eine Datenbank, um zu bestimmen, wohin der Datenrahmen übertragen werden sollen [17].

3. Durchführung des Projektes

3.10.3. Layer-2-Switch

Verfasst von: Naghmeh

Layer-2-Switch sind ähnlich wie Bridge. Sie verbinden Netze auf Schicht 2 und arbeiten als Brücken, indem sie Tabellen für die Übertragung von Rahmen zwischen Netzen erstellen. Der Zweck der Layer-2-Switch ist es, Daten auf der Grundlage von Flusstabellen einträgen weiterzuleiten [17].

Im Falle der Layer-2-Switching, Der Controller ist dafür verantwortlich, den physikalischen Elementen mitzuteilen, wie sie eingehende Datenpakete behandeln sollen. Er ist auch dafür verantwortlich, Datenströme in die Datenstromtabelle des Switches zu schieben. Der Switch ist in der Lage zu lernen, wie Pakete zu behandeln sind, die am Eingangsanschluss ankommen und den Ausgangsanschluss verlassen. Immer wenn ein Paket zum ersten Mal am Eingangsport des Switches ankommt, wird es an den Controller weitergeleitet. Nach dem Empfang eines Pakets, bestimmt der Controller, was damit zu tun ist und ob es weitergeleitet werden soll [55].

In dieser Simulation wurden, wie bereits erwähnt, die Open vSwitches verwendet. Open vSwitch ist ein virtueller Mehrschicht-Switch. Er verwendet virtuelle Netzwerkbrücken und Flussregeln, um Pakete zwischen Hosts weiterzuleiten. Er verhält sich wie ein physischer Switch. Ähnlich wie ein herkömmlicher Switch verwaltet OVS Informationen über angeschlossene Geräte wie z. B. die MAC-Adresse [73]. Obwohl er nicht nur die Aufgaben von Layer-2-Switch, sondern auch die Aufgaben von Layer-3-Switch durchführen kann, waren für diese Simulation die Funktionalitäten des Layer-2 Switches ausreichend. Deswegen wurden die 4 Switches als Layer-2-Switch betrachtet.

3.10.4. Layer-3-Switch

Verfasst von: James

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 [7].

3. Durchführung des Projektes

3.10.5. Dynamic Host Configuration Protocol

3.10.5.1. Vorüberlegung

Verfasst von: James

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. Mithilfe von Mininet konnte eine DHCP Konfiguration realisiert werden. Bis zu diesem Punkt wurde die IP-Adressverwaltung ohne DHCP realisiert. Das heißt, jeder Host musste manuell eine IP-Adresse zugewiesen bekommen. Diese manuelle Konfiguration wurde durch eine for-Schleife im Skript (siehe Abbildung 3.31) durchgeführt [7].

```
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:s0' % (r+1,
72                               self.addLink(host, switch)
```

Abbildung 3.31.: 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, sodass jeder Host von Beginn an die IP-Adresse **0.0.0.0** zugewiesen bekommt. Dadurch wird dem Host keine IP-Adresse manuell zugeordnet. Die IP-Adressverwaltung wird vom DHCP-Server übernommen (siehe Abbildung 3.32).

```
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:s0' % (r+1,
74                               self.addLink(host, switch)
```

Abbildung 3.32.: Dynamische IP-Adressenverwaltung

3. Durchführung des Projektes

3.10.5.2. Durchführung

Verfasst von: James

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 3.33). 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 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 Tabelle 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 [38].

3.10.5.3. Ergebnis

Verfasst von: James

Nachdem das Mininet Skript ausgeführt wird, kann mit Wireshark geprüft werden, ob ein DHCP-Server richtig implementiert wurde. Aus der folgenden Abbildung 3.34 ist ersichtlich, dass die Pakete zwischen h1 (192.168.1.2) und h11 (192.168.2.2) übertragen werden.

```
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 3.33.: DHCP-Konfiguration für alle Standorte

3. Durchführung des Projektes

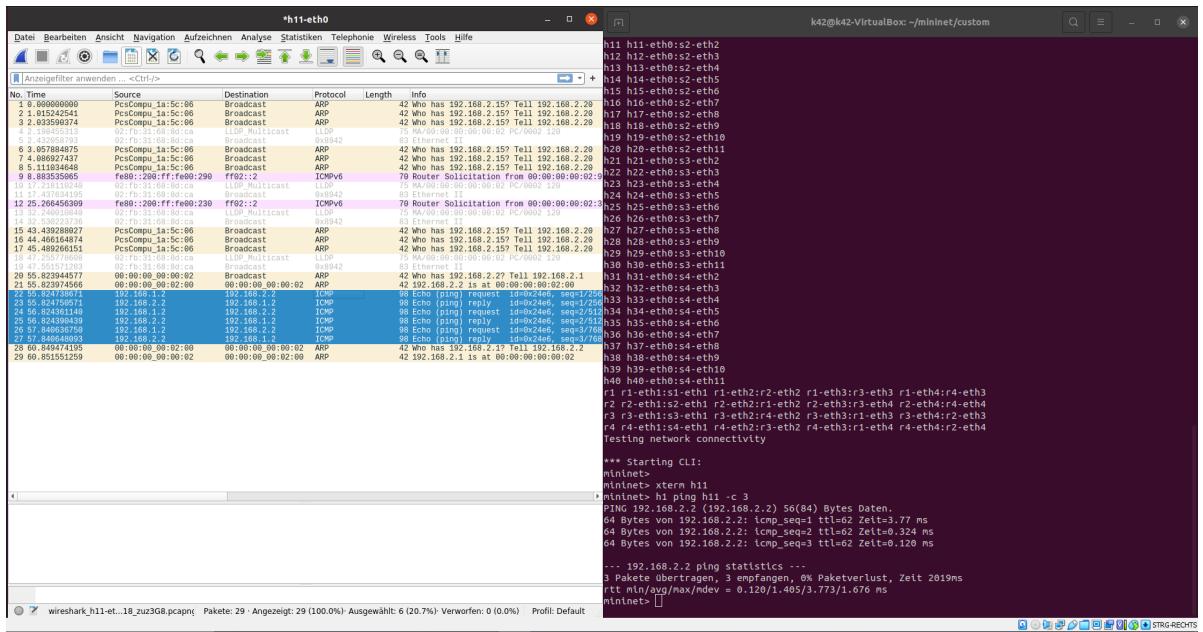


Abbildung 3.34.: Datenverkehr zwischen h1 und h11

Um die Netzwerkkonfiguration vom DHCP-Server zu betrachten, muss Wireshark vor Mininet gestartet und beim Ausführen des Mininet Skripts eine Switch betrachtet werden. In diesem Beispiel wird die Lokation Frankfurt betrachtet (siehe Abbildung 3.35).

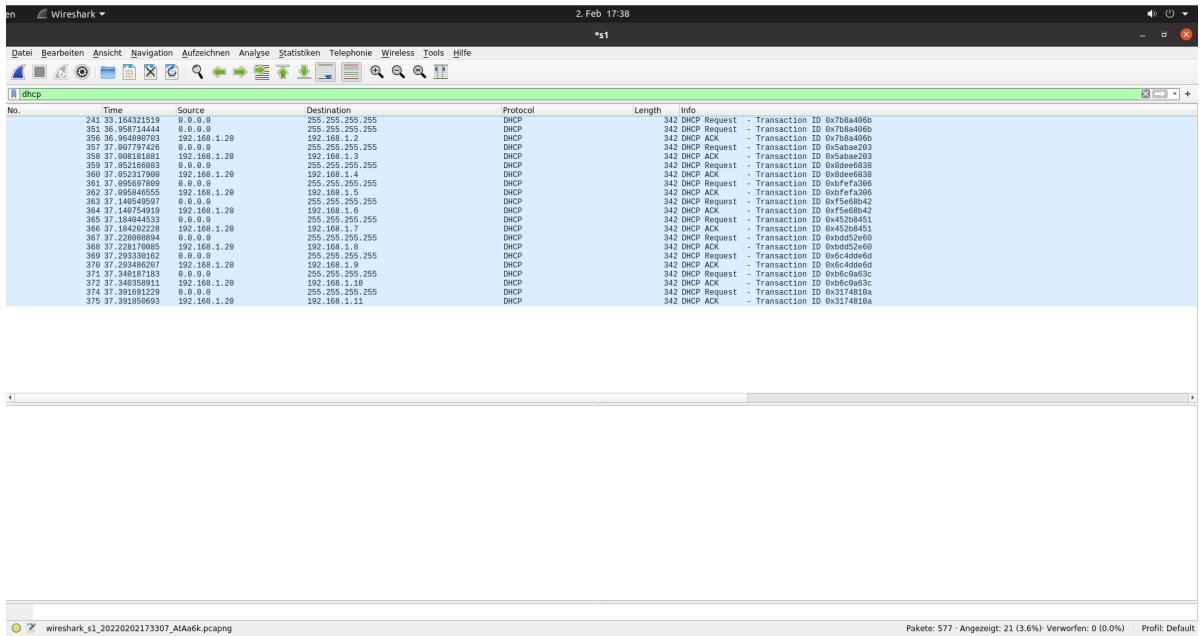


Abbildung 3.35.: Wireshark-Aufnahme von Switch s1

3. Durchführung des Projektes

Anhand von Abbildung 3.38 sind die gewöhnlichen Schritte vom DHCP Prozess zu erkennen. Als erstes schickt ein Client einen **Broadcast** an einem erreichbaren Server (**DHCP Discover**). Die Absender-IP-Adresse ist **0.0.0.0** und die Zieladresse ist **255.255.255.255**. Der voreingestellte DHCP-Server empfängt den **Broadcast** und antwortet (**DHCP Offer**). Diese Antwort wird an die angebotene IP-Adresse (**192.168.1.2**) gesendet (**Unicast**). Der Client nimmt per Broadcast (**255.255.255.255**) das Angebot an (**DHCP Request**). Der Server akzeptiert die Adressanfrage an (**ACK**) und speichert im Adresspool die IP als vergeben ab [7].

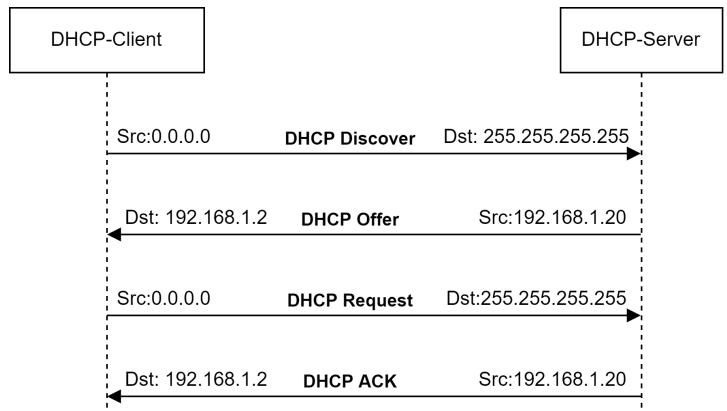


Abbildung 3.36.: Kommunikation zwischen DHCP-Client und DHCP-Server

3. Durchführung des Projektes

Die übermittelten Informationen vom DHCP Prozess werden im Pfad `/var/lib/dhcp/` gespeichert. Die Datei `dhcpd.leases` ist eine DHCP client Datenbank, die eine Reihe von lease Deklarationen enthält. Wenn ein lease erworben, erneuert oder freigegeben wird, kann man dies in der Datei einsehen [39]. In der Datei `dhclient.leases` werden alle gültigen leases gespeichert[34]. Somit kann der Schritt **DHCP Discover** bei einem wiederholten Durchlauf innerhalb der Lease-Time übersprungen werden, da der Client seine IP-Adresse schon erhalten hat [49] (siehe Abbildung 3.37).

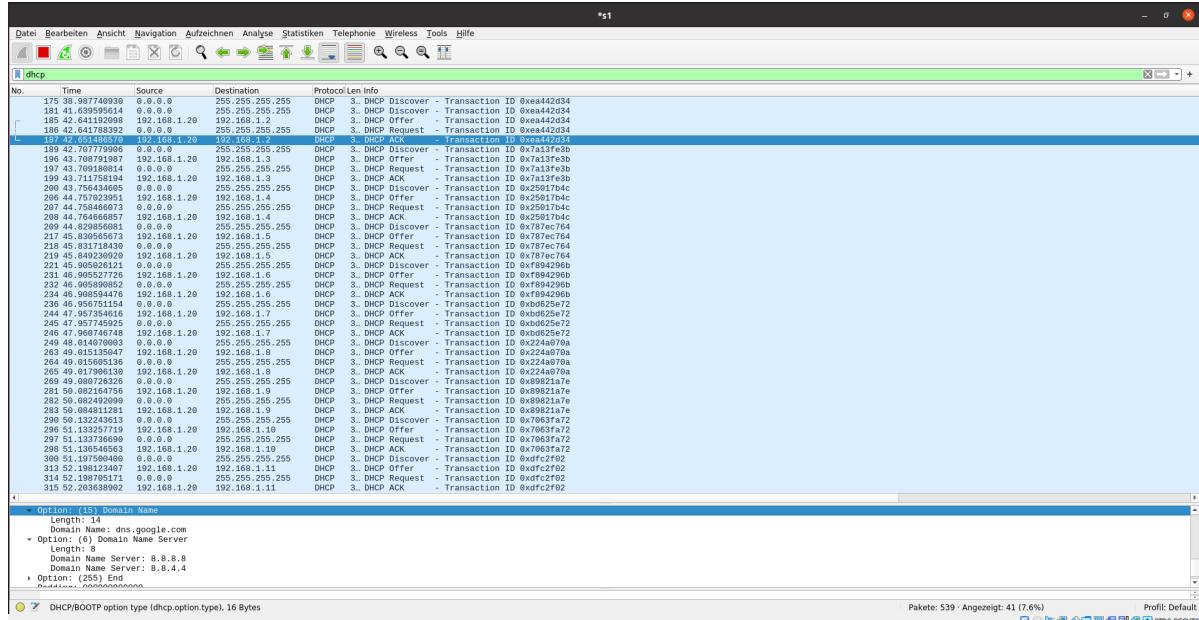


Abbildung 3.37.: Wireshark-Aufnahme vom gekürzten DHCP Prozess

Falls die Protokolle **DHCP Discover** und **DHCP Offer** nicht übersprungen werden sollen, können die zuständigen Dateien `dhcpd.leases` und `dhclient.leases` gelöscht werden. Diese werden beim nächsten Durchlauf neu erstellt.

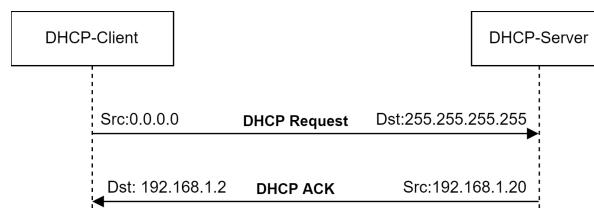


Abbildung 3.38.: Kommunikation zwischen DHCP-Client und DHCP-Server

3. Durchführung des Projektes

3.10.6. Domain Name System

3.10.6.1. Vorüberlegung

Verfasst von: James

Durch das DNS-Protokoll werden Domainnamen zu IP-Adressen umgewandelt [7]. Dies wurde bei der Einrichtung des NAT-Firewalls manuell implementiert, sodass Hosts die Möglichkeit haben, nicht nur per IPv4 auf das Internet zugreifen zu können. Es wurde die IP-Adresse des DNS-Servers in die Datei **/etc/resolvconf/resolv.conf/head** eingetragen und gespeichert. Zudem kann mithilfe des implementierten DHCP Servers und dem dhclient auch ein Ergebnis erzielt werden.

3.10.6.2. Durchführung

Verfasst von: James

Es werden innerhalb der **dhcpd.conf** Datei im Pfad **/etc/dhcp/** zwei Zeilen in jedem Subnetz hinzugefügt. Mit **option domain-name-servers 8.8.8.8, 8.8.4.4;** wird der verfügbare DNS Server für den Client definiert. Durch **option domain-name "dns.google.com";** wird der Domänenamen angegeben, der zur Auflösung von Hostnamen verwendet werden soll [38] (siehe 3.39). Es kann auch der dhclient genutzt werden, um einen DNS Server zu definieren [37]. Im Pfad **/etc/dhcp/** befindet sich die dhclient.conf Datei, die mit prepend domain-name-servers 8.8.8.8 ergänzt werden muss [35].

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.2 192.168.1.254;
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    | option domain-name "dns.google.com";
    option routers 192.168.1.1;
    INTERFACES="r1-eth1";
}
subnet 192.168.2.0 netmask 255.255.255.0 {
    range 192.168.2.2 192.168.2.254;
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    option domain-name "dns.google.com";
    option routers 192.168.2.1;
    INTERFACES="r2-eth1";
}
subnet 192.168.3.0 netmask 255.255.255.0 {
    range 192.168.3.2 192.168.3.254;
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    option domain-name "dns.google.com";
    option routers 192.168.3.1;
    INTERFACES="r3-eth1";
}
subnet 192.168.4.0 netmask 255.255.255.0 {
    range 192.168.4.2 192.168.4.254;
    option domain-name-servers 8.8.8.8, 8.8.4.4;
    option domain-name "dns.google.com";
    option routers 192.168.4.1;
    INTERFACES="r4-eth1";
}
```

Abbildung 3.39.: DHCP-Konfiguration für alle Standorte

3. Durchführung des Projektes

3.10.6.3. Ergebnis

Verfasst von: James

Die Übermittlung des Domain Namens und des Servers kann beobachtet werden, indem die DHCP Protokolle in Wireshark betrachtet werden [47](siehe Abbildung 3.40).

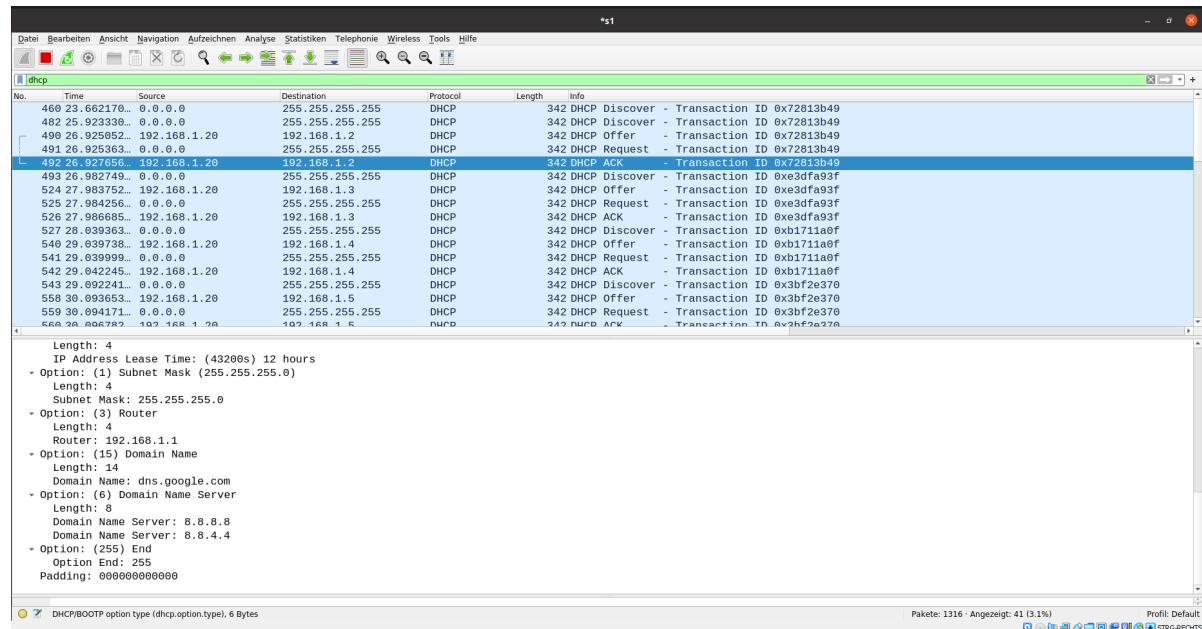


Abbildung 3.40.: Im DHCP ACK Protokoll werden die DNS Informationen vermittelt

Nachdem Mininet gestartet wurde, kann mithilfe von `systemd-resolve --status` der genutzte DNS Server betrachtet werden [36](Siehe Abb. 3.41).

```
root@k42-VirtualBox:/home/k42/mininet/custom# systemd-resolve --status
Global
  LLMNR setting: no
  MulticastDNS setting: no
  DNSOverTLS setting: no
    DNSSEC setting: no
    DNSSEC supported: no
  Current DNS Server: 8.8.8.8
    DNS Servers: 8.8.8.8
                  192.168.0.1
                  8.8.4.4
  DNS Domain: dns.google.com
  DNSSEC NTA: 10.in-addr.arpa
                16.172.in-addr.arpa
                168.192.in-addr.arpa
                17.172.in-addr.arpa
                18.172.in-addr.arpa
                19.172.in-addr.arpa
                20.172.in-addr.arpa
                21.172.in-addr.arpa
                22.172.in-addr.arpa
                23.172.in-addr.arpa
                24.172.in-addr.arpa
                25.172.in-addr.arpa
```

Abbildung 3.41.: Terminalausgabe von h1

4. Diskussion der Ergebnisse

Verfasst von: James

Am Anfang des Projektberichtes wurde von einer “dramatischen Zunahme der Netzwerkkomplexität” gesprochen, wobei eine traditionelle Vorgehensweise der Netzwerkadministration ineffizient sei. Mithilfe des Konzepts SDN, sollte eine effizientere Methode der Netzwerkadministration realisiert werden. Durch die Lösung der Problemstellungen konnten diese zugrunde gelegte Behauptung bestätigt werden. Beispielsweise mussten vier Lokationen realisiert werden, also wurde zu Beginn eine Lokation mit den gewünschten Funktionen erstellt. Anschließend wurden Kopien dieser Lokation für alle weiteren Lokationen genutzt. Durch die Strategie eine kleinstmögliche Umgebung mit den gewünschten Kriterien zu konstruieren, erleichterte es die Vorgehensweise auf Ergebnisse zu kommen. Mininet und der Floodlight Controller ermutigten diese Vorgehensweise, da die Anpassungen leicht umsetzbar waren. Während der Durchführung der Problemstellungen wurde auch festgestellt, dass viele manuelle Einrichtungen mithilfe von SDN automatisiert werden konnten. Die erforschten Ergebnisse werden hier nochmals zusammengefasst und die automatisierten Netzwerkfunktionen durch SDN mit den manuellen Vorgehensweisen verglichen. Anschließend wird kritisch reflektiert, was in der Durchführung mithilfe von SDN nicht reibungslos funktioniert hat.

4.1. Analyse der Ergebnisse

Verfasst von: James

Bei der Verschlüsselung der Netzwerkverbindung zwischen den einzelnen Lokationen konnte über Mininet zwischen den Routern von jedem Standort die VPN Verbindung hergestellt werden. Anstatt eine manuelle Verschlüsselung an jedem Router einzurichten, konnte zentralisiert über Mininet ein Ergebnis erzielt werden. Um private IP-Adressen auf öffentliche IP-Adressen abilden zu lassen, musste ein NAT eingerichtet werden. Normalerweise wird dies mit der firewalling software *iptables* umgesetzt. Mit *iptables* werden Regeln definiert die für das Filtern und Modifizieren von Paketen zuständig sind. Diese manuelle Einrichtung des NAT-Firewalls an einem Router konnte mithilfe von Mininet übersprungen werden und automatisiert im Skript ergänzt werden. Somit regelt Mininet die Anbindung von einem privaten Netz über **NAT** an das Internet bei jedem Standort. Der Aufbau eines zentralen Topologie-Viewers des Netzwerkes konnte mit dem standardinstallierten Floodlight-Modul dargestellt werden. Indem die Benutzeroberfläche des Controllers in einem Webbrower aufgerufen wird und auf den Reiter **Topology**

4. Diskussion der Ergebnisse

klickt, wird eine Topologie dargestellt. Die Erstellung der Topologie vom Controller ist nur möglich, da der Controller die verbundenen Links von den Hosts und den Switches erkennt. Die Realisierung einer zentralen Monitoring Lösung konnte durch eine Konfigurationsänderung in Mininet erstellt werden. Dadurch hatte jeder Netzwerkteilnehmer auf Nagios Core und dem Floodlight Controller Zugriff bekommen. Anstatt manuell bei jedem Netzwerkteilnehmer dies einzustellen zu müssen, konnte durch eine Ergänzung des Mininet Skripts die Konfiguration automatisch für alle Hosts eingestellt werden. Eine QoS Funktion innerhalb eines Netzwerkes ohne SDN zu realisieren, ist sehr aufwendig und enthält viele Beschränkungen. Mit SDN kann wegen der zentralisierten Struktur eine Queueliste erstellt werden, die bestimmten Datenverkehr selbst bei begrenzter Netzwerkkapazität priorisiert. Diese Idee wurde mit zwei verschiedenen **Queues** nachgeahmt. Audio- und Videokonferenzen wurden in einer **Queue** definiert und der restliche Datenverkehr in einer anderen. Umgesetzt wurde es im Mininet Skript. Zudem konnte durch eine weitere Ergänzung des Skripts eine Priorisierung der Datenübertragung hergestellt werden. Bei einer Datenübertragung zwischen Berlin und Frankfurt sollte der Datenaustausch priorisiert und alle parallel dazu laufenden Datenübertragungen gedrosselt werden. Indem ein SSH Client in Berlin und ein SSH Server in Frankfurt eingerichtet wird, konnte mithilfe eines Python Skripts auf dem Host die gewünschte Prioritätensetzung ins Netzwerk implementiert werden. Die Realisierung eines DHCP Servers sowohl eines DNS-Dienst für jede Lokation kann automatisiert werden. Somit kann durch Änderung des Mininet Skripts und der **dhcpd.conf** sowie **dhclient.conf** Datei ein DHCP Server und DNS-Dienst realisiert werden. Dies erspart die Zeit an jeder Lokation eine manuelle Konfiguration durchzuführen und kann in einer Weise implementiert werden, dass beim Ausführen des Mininet Skripts die Dienste für alle Hosts konfiguriert werden.

Bei einer Änderung der vorgelegten Anforderungen, wie beispielsweise eine **Quality of Service Funktion** von einer anderen Datenübertragung, kann dies durch SDN und mithilfe des Mininet Skript leicht verändert werden. Somit müsste ein Administrator nicht alle Schnittstellen umprogrammieren, sondern kann effizienter in einem automatisierten Schritt das Netzwerk ändern.

4.2. Kritische Betrachtung

Verfasst von: James

Obwohl SDN viele Vorteile hat, war der Einstieg in das Thema sehr zeitaufwendig. Viele Installationsanleitungen haben auf Anhieb nicht funktioniert und mussten individuell mit Alternativen Möglichkeiten angepasst werden. Vorkonfigurierte VMs mit mininet und einem Floodlight Controller haben nicht funktioniert, somit musste selbstständig eine VM konstruiert werden. Zudem war die Vorstellung des Netzwerkes zu Beginn anders als das am Ende erstellte Ergebnis, da SDN viele Funktionalitäten und Alternativmöglichkeiten besitzt. Dadurch kann eine Art der Entscheidungslähmung entstehen, da die Optionen und Einstellungen sehr vielfältig sind. Da der Floodlight Controller in

4. Diskussion der Ergebnisse

SDN an zentraler Stelle steht, müssen alle Konfigurationen korrekt eingestellt werden. Es muss logischerweise der Controller konstant im Hintergrund laufen und wenn der Controller ausfällt, fällt das gesamte System aus. Das heißt, der Controller muss fehlersicher und mit allen Lokationen verbunden sein. Der Floodlight Controller befindet sich im Netzwerkplan am Standort Frankfurt, jedoch sollte in der Theorie ein zweiter Controller an einer anderen Lokation existieren, der beim Ausfall des Frankfurter Controllers einspringt.

5. Fazit

Verfasst von: James, Naghmeh

Auf Grundlage einer umfassenden Recherche, die Durchführung von neun definierten Aufgabenstellungen sowie ein daraus resultierendes Ergebnis sind neue Erkenntnisse über das Konzept SDN für die Gruppenmitglieder gewonnen worden. Ziel dieses Projektes war es nicht nur die Aufgaben zu lösen, sondern verstehen zu können, warum das Konzept SDN die Zukunft von Netzwerkadministration sein könnte.

Nach der Installation und Konfiguration der notwendigen Tools wie die VMs, Mininet und dem Floodlight-Controller und einer erfolgreichen Analyse der Aufgaben hat die Ausarbeitung des Projektes angefangen. Bei der Verschlüsselung des Datenverkehrs zwischen Lokationen wurde die Methode **IPSEC over GRE** implementiert und wurde damit ein Site-to-Site-VPN Verbindung hergestellt. Da die Lokationen über einen Tunnel durch das Internet miteinander verbunden sind, benötigen alle Lokationen eine DSL-Verbindung. Neben einer DSL-Leitung war auch eine Standleitung für alle Lokationen erforderlich. Nach einiger Recherche, Nachfragen und Vergleiche wurde eine passende Wahl für die DSL-Leitung und Standleitung gefunden. NAT-Firewall wurden am Router jeder Lokation implementiert. Das Programm *iptables* wurde umgesetzt, um Regeln für die jeweiligen Router zu definieren. Mithilfe von Mininet wurden Einrichtungen des NAT-Firewalls implementiert, anstatt manuell an jedem Router im Skript dies umzusetzen. Somit ist Mininet zuständig für die Anbindung von einem privaten Netz über **NAT** an das Internet. Eine andere Anforderung für die Sicherstellung des Systems war die Implementierung einer Web Proxy Funktion. Die HTTP und HTTPS Anfragen der Hosts wurden durch den Proxy durchgeführt und die Antwort an dem Host weitergeleitet. Jedoch konnte die Web Proxy Funktion zum Schluss nicht korrekt implementiert werden. Für das zentrale Monitoring wurde Nagios Core verwendet. Dies ermöglicht die Überprüfung der Verfügbarkeit, Geschwindigkeit und Dienste der Netzwerkkomponenten. Bei entstandenen Fehlern werden Administratoren für die Fehlerbehebung benachrichtigt. Mithilfe der **Quality of Service** Funktion ist das Netzwerk in der Lage, Anwendungen und Datenverkehr selbst bei begrenzter Netzwerkkapazität zuverlässig mit hoher Priorität auszuführen. Den Unternehmen war es wichtig, dass bei einer Auslastung des Netzwerkes die Audio- und Videokonferenzen stabil und flüssig laufen. Dafür wurden zwei **Queues** benötigt, einmal für die Audio- und Videokonferenzen(**Queue 1**) und eine für den restlichen Datenverkehr(**Queue 0**). Dies wurde im Mininet Skript umgesetzt. Dabei werden alle UDP Pakete auf die **Queue 1** mit einer minimalen Bandbreite von sieben und einer maximalen Bandbreite von 20 Megabit

5. Fazit

weitergeleitet. Bei den Bedingungen wurde zusätzlich auch die höchste Priorität eingegeben. Bei den wichtigen Datenübertragungen zwischen Berlin und Frankfurt sollen die Flows priorisiert und alle parallel dazu laufenden Datenübertragungen gedrosselt werden. Die Datenübertragung soll über einem in der Zentrale eingerichteten SSH Server laufen. Dazu wird auf dem Host in Berlin und in Frankfurt ein SSH-Client benötigt. Bei der Übertragung nutzt SSH üblicherweise den Port 22 über dem TCP. Mit dieser Information wird die Bedingung für die Priorisierung des Flows festgelegt. Die Datenübertragung und die Priorisierung wurden in einem Python-Skript realisiert. Wegen Mangel an Zeit wurden nur die Netzwerkfunktionen, die die Gesamtleistung des Netzwerks verbessern konnten und notwendig waren, realisiert, wie beispielsweise Layer-2-Switch, DHCP und DNS. Die Layer-2-Switches waren notwendig, um eine Verbindung zum Controller herzustellen und die 10 Hosts in jeder Lokation miteinander zu verbinden. Im Vergleich zu Layer-2-Switches sind Hubs ineffizient für das Netzwerk, da bei jedem Datenverkehr der Controller entscheiden muss, alle mit ihm verbundenen Hosts zu überfluten. Dies beeinträchtigt die Gesamtleistung des Netzwerkes. Deshalb wurde auch kein Hub realisiert. Nach Recherche zeigte sich, dass der Floodlight Controller DHCP unterstützt und umsetzbar ist. Damit konnte ein DHCP Server bei allen Lokationen implementiert werden und die individuelle Konfiguration an jedem Standort musste nicht durchgeführt werden. Innerhalb der Konfiguration des DHCP Servers konnte eine Alternativmöglichkeit für einen DNS-Dienst gefunden werden, der mithilfe von DHCP alle Clients mit einem DNS-Dienst ausstattet. Durch zusätzliche Einstellungen konnte dies realisiert werden.

Aus den Ergebnissen geht hervor, dass durch SDN eine dynamischere Administration möglich ist. Beispielsweise zeigt die Realisierung einer *QoS* Funktion, dass Benutzern bei bestimmten Anwendungen mehr Bandbreite zur Verfügung steht, da eine agile Verwaltung des Datenverkehrs durch SDN ermöglicht wird. Zudem verändert sich durch SDN die Art der Netzwerkkonfiguration. Normalerweise müsste ein Netzwerkadministrator für das Hinzufügen eines Gerätes in einem Netzwerk mehrere Schritte absolvieren, wie die manuelle Konfiguration von Switches und Routern etc.. Mit SDN ist diese Konfiguration programmierbar, sodass viele Konfigurationsschritte automatisiert oder übersprungen werden können. Ebenfalls wichtig ist, dass durch die zentrale Verwaltung des Netzes über den Floodlight Controller eine einfache Einstellung eines Topologie-Viewers ermöglicht wird und dadurch keine weiteren Tools benötigt werden. Aus dem Ergebnis der Priorisierung der Datenübertragung von Berlin nach Frankfurt geht hervor, dass eine Priorisierung der Datenübertragung von einer anderen Lokation nach Frankfurt durch einfache Änderungen auch realisierbar sei. Kleinere Änderungen an der programmierten Konfiguration könnten ausreichen, um ein gleiches Ergebnis zu erzielen. Des Weiteren konnte bei der Einrichtung des NAT-Firewalls sowie bei der **VPN** Einrichtung die verringerte Komplexität der Implementierung nachvollziehbar werden. Aus Wissen von vorherigen Modulen war die Einstellung des NAT-Firewalls auf zwei Host-Maschinen zeitaufwendiger und komplexer als die Realisierung im Projekt. Insbesondere ist die Skalierbarkeit des Netzwerkes effizienter umsetzbar, da automatisierte Voreinstellungen angepasst werden können, um die gewünschte Infrastruktur aufzubauen. Zudem verdeutlichen die Ergebnisse bei der Einstellung vom DHCP Prozess sowie

des DNS-Dienstes die vereinfachte Erweiterung von Netzwerkfunktionen.

5.1. Zukunftsaussichten

Ein wesentlicher Vorteil von SDN ist die vereinfachte Skalierbarkeit des Netzwerkes. Das Netzwerkergebnis vom Projekt beinhaltet mehrere verschiedene Netzwerkfunktionalitäten und wenn beispielsweise neue Anforderungen verlangt werden, wie an einem Standort weitere Hosts zu konfigurieren, könnte durch SDN dieses Problem einfach gelöst werden. Folgendes Szenario wird angenommen: Das Unternehmen hat einen neuen Mitarbeiter in Frankfurt eingestellt. Der Netzwerkadministrator muss das Netzwerk neu konfigurieren, um einen weiteren Mitarbeiter an das Netz zu bringen.

Mithilfe von Mininet ist dies leicht umsetzbar. Während Mininet läuft müssen folgende Konfigurationen eingestellt werden: Mit `py net.addHost(name='h0', ip='0.0.0.0', defaultRouter='via 192.168.1.1')` wird ein neuer Host erstellt. Nun benötigt der neue Host **h0** eine Verbindung zur Switch s1, eine IP-Adresse, einen Broadcast, einen Default Route, einen neuen Port an der Switch s1 und eine loopback Adresse. In Mininet wird der Befehl `py net.addLink(h0, s1)` zur Verbindung von s1 und h0 ausgeführt. Als nächstes bekommt **h0** die IP-Adresse **192.168.1.35/24** mit dem Befehl `h0 ip addr add 192.168.1.35/24 dev h0-eth0`. Der Broadcast wird mit `h0 ifconfig h0-eth0 broadcast 192.168.1.255` gesetzt und die Default Route auf **192.168.1.1** eingerichtet mit `h0 ip route add default via 192.168.1.1`. Anschließend wird mit `h0 ifconfig lo 127.0.0.1` eine loopback Adresse eingestellt. In einem neuen Terminal kann mit `sudo ovs-ofctl dump-ports s1` alle angeschlossenen Ports an der Switch betrachtet werden. Dadurch ist zu erkennen, dass noch kein Port für h0 existiert. Mit `sudo ovs-vsctl add-port s1 s1-eth13` wird ein neuer Port hergestellt und somit entsteht eine Verbindung vom Netz zur neu erstellten Host h0. Jetzt besitzt die Host h0 die gleiche Funktionalitäten wie alle anderen Hosts im Netzwerk. Abbildung 5.1 zeigt, dass die Topologie von Frankfurt den Host h0 berücksichtigt.

Das Potenzial welches SDN bietet, ist durch den Umfang dieses Projektes noch nicht komplett ausgeschöpft. Dies zeigt insbesondere das letzte gegebene Beispiel, welches die in Software Defined Networking liegenden Stärken verdeutlicht.

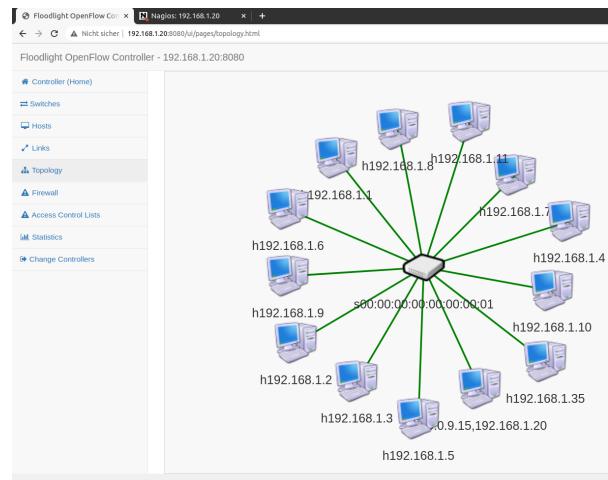


Abbildung 5.1.: Die Topologie zeigt, dass h0 in Frankfurt existiert

Literaturverzeichnis

- [1] 1und1. *Internet- und DSL-Tarife*. URL: <https://dsl.1und1.de/business> (besucht am 15.01.2022).
- [2] Konstantin Agouros. *Software Defined Networking*. Walter de Gruyter, 2017. ISBN: 978-3-11-044984-6.
- [3] Dalia A.Hamid Ammar D.Jasim. „Enhancing the Performance of OpenFlow Network by Using QoS“. In: *International Journal of Scientific and Engineering Research* (Mai 2016). DOI: ISSN2229-5518.
- [4] AVI. *Software Defined Networking*. URL: <https://avinetworks.com/glossary/software-defined-networking/> (besucht am 16.12.2021).
- [5] Siamak Azodolmolky. *Software Defined Networking with OpenFlow*. Packt Publishing, 2013. ISBN: 978-1-84969-872-6.
- [6] Sumit Badotra. „A Review Paper on Software Defined Networking“. In: *International Journal of Advanced Computer Research* 8 (März 2017).
- [7] Christian Baun. „Grundlagen der Computervernetzung“. In: *Computer Networks/-Computernetze*. Springer, 2019, S. 15–33.
- [8] Ismail Baydan. *Add Route In Linux*. 2021. URL: <https://linuxtect.com/ip-route-add-add-route-in-linux/> (besucht am 04.01.2022).
- [9] Michael Bredel. *OpenFlow Controller*. URL: <https://www.admin-magazine.com/Articles/Floodlight-Welcome-to-the-World-of-Software-Defined-Networking> (besucht am 24.12.2021).
- [10] Dor Cohen. *MTU and MSS: What you need to know*. 2017. URL: <https://www.imperva.com/blog/mtu-mss-explained/> (besucht am 07.12.2021).
- [11] Nagios Core. *Monitoring Routers and Switches*. URL: <https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/monitoring-routers.html> (besucht am 27.01.2022).
- [12] Sudi David. *Importance of QoS*. 2021. URL: <https://www.section.io/engineering-education/understanding-quality-of-service/> (besucht am 23.01.2021).
- [13] Andreas Donner. *Was ist Software-Defined Networking*. 2018. URL: <https://www.ip-insider.de/was-ist-software-defined-networking-sdn-a-657442/> (besucht am 16.12.2021).

- [14] Bob Emmerson. *The Case for SDN*. 2012. URL: <https://www.nojitter.com/case-sdn> (besucht am 16.12.2021).
- [15] firewall.cx. *UNDERSTANDING VPN IPSEC TUNNEL MODE AND IPSEC TRANSPORT MODE – WHAT'S THE DIFFERENCE?* URL: <https://www.firewall.cx/networking-topics/protocols/870-ipsec-modes.html> (besucht am 05.12.2021).
- [16] Firewall.cx. *Configuring site to site ipsec vpn tunnel between cisco routers*. URL: <https://www.firewall.cx/cisco-technical-knowledgebase/cisco-routers/867-cisco-router-site-to-site-ipsec-vpn.html> (besucht am 05.12.2021).
- [17] GeeksforGeeks. *Network Devices (Hub, Repeater, Bridge, Switch, Router, Gateways and Router)*. URL: <https://www.geeksforgeeks.org/network-devices-hub-repeater-bridge-switch-router-gateways/> (besucht am 24.12.2021).
- [18] Alexander Gelberger, Niv Yemini und Ran Giladi. „Performance analysis of software-defined networking (SDN)“. In: *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE. 2013, S. 389–393.
- [19] Grotto. *Working with the Mininet VM*. URL: <https://www.grotto-networking.com/SDNfun.html> (besucht am 11.12.2022).
- [20] John Hales. *SDN and Cloud Computing*. URL: <http://novacontext.com/sdn-and-cloud-computing/> (besucht am 16.12.2021).
- [21] Muhamad Hasan u.a. „SDN Mininet Emulator Benchmarking and Result Analysis“. In: *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. IEEE. 2020, S. 355–360.
- [22] Nick Feamster Hyojoon Kim. „Improving Network Management with Software Defined Networking“. In: *IEEE Communications Magazine* (2013).
- [23] ingrammicro. *7 Advantages of Software Defined Networking*. 2021. URL: <https://imaginext.grammicro.com/data-center/7-advantages-of-software-defined-networking> (besucht am 16.12.2021).
- [24] Ryan Izard. *Floodlight Controller Load Balancer*. URL: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343617/Load%5C+Balancer> (besucht am 26.12.2021).
- [25] Ryan Izard. *How to Collect Switch Statistics (and Compute Bandwidth Utilization)*. 2019. URL: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/21856267/How+to+Collect+Switch+Statistics+and+Compute+Bandwidth+Utilization> (besucht am 13.12.2021).
- [26] Ryan Izard. *Web GUI*. URL: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/40403023/Web+GUI> (besucht am 28.12.2021).
- [27] jolson. *Nagios Core - Installing Nagios Core From Source*. URL: <https://support.nagios.com/kb/article.php?id=96> (besucht am 29.01.2022).

- [28] Navtej Ghuman Karamjeet Kaur Japinder Singh. „Mininet as Software Defined Networking Testing Platform“. In: *International Conference on Communication, Computing & Systems* (2014).
- [29] Michael Kerrisk. *ip-xfrm(8) — Linux manual page*. 2021. URL: <https://man7.org/linux/man-pages/man8/ip-xfrm.8.html> (besucht am 06.01.2022).
- [30] Rahamatullah Khondoker. *SDN and NFV Security - Security Analysis of Software-Defined Networking and Network Function Virtualization*. Springer, 2018. ISBN: 978-3-319-71760-9.
- [31] Aaron Kili. *How to set permanent DNS Nameservers in Ubuntu and Debian*. 2021. URL: <https://www.tecmint.com/set-permanent-dns-nameservers-in-ubuntu-debian/> (besucht am 15.01.2022).
- [32] Elektronik Kompendium. *ESP - Encapsulating Security Payload*. URL: <https://www.elektronik-kompendium.de/sites/net/1410261.htm> (besucht am 09.01.2022).
- [33] Elektronik Kompendium. *NAT-Network Address Translation*. URL: <https://www.elektronik-kompendium.de/sites/net/0812111.htm> (besucht am 10.01.2022).
- [34] Ted Lemon. *Linux man page - dhclient.conf(5)*. URL: <https://linux.die.net/man/5/dhclientleases> (besucht am 25.01.2022).
- [35] Ted Lemon. *Linux man page - dhclient.conf(5)*. URL: <https://linux.die.net/man/5/dhclient.conf> (besucht am 28.01.2022).
- [36] Ted Lemon. *Linux man page - dhclient.conf(5)*. URL: <https://wiki.ubuntuusers.de/systemd/systemd-resolved/> (besucht am 28.01.2022).
- [37] Ted Lemon. *Linux man page - dhclient.conf(8)*. URL: <https://linux.die.net/man/8/dhclient> (besucht am 28.01.2022).
- [38] Ted Lemon. *Linux man page - dhcpd.conf(5)*. URL: <https://linux.die.net/man/5/dhcpd.conf> (besucht am 22.01.2022).
- [39] Ted Lemon. *Linux man page - dhcpd.conf(8)*. URL: <https://linux.die.net/man/8/dhcpd> (besucht am 23.01.2022).
- [40] Donner Andreas Luber Stefan. *Was ist Software Defined Networking*. URL: <https://www.ip-insider.de/was-ist-software-defined-networking-sdn-a-657442/> (besucht am 01.11.2021).
- [41] Microsoft. *Design virtual networks with NAT gateway*. URL: <https://docs.microsoft.com/en-us/azure/virtual-network/nat-gateway/nat-gateway-resource> (besucht am 11.01.2022).
- [42] Mininet. *Mininet Download*. 2021. URL: <http://mininet.org/download/> (besucht am 05.12.2021).
- [43] Mininet. *Mininet Overview*. 2021. URL: <http://mininet.org/overview/> (besucht am 05.12.2021).

- [44] Laura Victoria Morales, Andres Felipe Murillo und Sandra Julieta Rueda. „Extending the floodlight controller“. In: *2015 IEEE 14th International Symposium on Network Computing and Applications*. IEEE. 2015, S. 126–133.
- [45] Sergey Morzhov, Igor Alekseev und Mikhail Nikitinskiy. „Firewall application for Floodlight SDN controller“. In: *2016 International Siberian Conference on Control and Communications (SIBCON)*. 2016, S. 1–5. DOI: 10.1109/SIBCON.2016.7491821.
- [46] MPC. *Standleitung Preise*. URL: <https://www.mpcservice.com/standleitung/standleitung-preise/> (besucht am 15.01.2022).
- [47] Sameena Naaz und Firdos Badroo. „Investigating DHCP and DNS Protocols Using Wireshark“. In: *IOSR Journal of Computer Engineering (IOSR-JCE)* 18 (Juni 2016), S. 2278–661. DOI: 10.9790/0661-1803020108.
- [48] Nagios. *Server Monitoring with Nagios*. URL: <https://www.nagios.com/solutions/server-monitoring/> (besucht am 28.01.2022).
- [49] Nokia. *TRIPLE PLAY RELEASE 21.2.R1*. URL: https://documentation.nokia.com/cgi-bin/dbaccessfilename.cgi/3HE17164AAAATQZZA01%5C_V1%5C_7450%5C%20ESS%5C%207750%5C%20SR%5C%20and%5C%20VSR%5C%20Triple%5C%20Play%5C%20Service%5C%20Delivery%5C%20Architecture%5C%20Guide%5C%2021.2.R1.pdf (besucht am 25.01.2022).
- [50] o2. *Internet- und DSL-Tarife*. URL: <https://www.o2business.de/festnetz-internet/dsl-business/> (besucht am 15.01.2022).
- [51] ONF. *Software-Defined Networking (SDN) Definition*. URL: <https://opennetworking.org/sdn-definition/> (besucht am 16.12.2021).
- [52] paloalto. *What is a Site-to-Site VPN?* URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-a-site-to-site-vpn> (besucht am 30.11.2021).
- [53] Keerthi PS. *Iptables nat masquerade – How we do it?* 2019. URL: <https://bobcares.com/blog/iptables-nat-masquerade/> (besucht am 22.01.2022).
- [54] Sangeetha Elango Raphael Eweka. „IMPLEMENTATION OF ADDRESS LEARNING/PACKET FORWARDING, FIREWALL AND LOAD BALANCING IN FLOODLIGHT CONTROLLER FOR SDN NETWORK MANAGEMENT“. In: (2015). URL: <https://www.ftms.edu.my/journals/pdf/IJISE/Apr2015/160-170.pdf> (besucht am 17.12.2021).
- [55] Abhishek Rastogi und Abdul Bais. „Comparative analysis of software defined networking (SDN) controllers — In terms of traffic handling capabilities“. In: *2016 19th International Multi-Topic Conference (INMIC)*. 2016, S. 1–6. DOI: 10.1109/INMIC.2016.7840116.
- [56] Abhishek Rastogi und Abdul Bais. „Comparative analysis of software defined networking (SDN) controllers—In terms of traffic handling capabilities“. In: *2016 19th International Multi-Topic Conference (INMIC)*. IEEE. 2016, S. 1–6.

- [57] ProSec Security redefined. *Proxy Server*. URL: <https://www.prosec-networks.com/blog/proxy-server/> (besucht am 26.01.2022).
- [58] Karl Rupp. *NAT- Network Address Translation*. URL: https://www.karlrupp.net/en/computer/nat_tutorial (besucht am 23.01.2022).
- [59] Rusty Russel. *iptables*. URL: <https://linux.die.net/man/8/iptables> (besucht am 17.01.2022).
- [60] Ola Salman u. a. „QoS guarantee over hybrid SDN/non-SDN networks“. In: *2017 8th International Conference on the Network of the Future (NOF)*. 2017, S. 141–143. DOI: 10.1109/NOF.2017.8251237.
- [61] Ola Salman u. a. „SDN controllers: A comparative study“. In: *2016 18th Mediterranean Electrotechnical Conference (MELECON)*. 2016, S. 1–6. DOI: 10.1109/MELCON.2016.7495430.
- [62] Networx Security. *Web Proxy*. URL: <https://www.networxsecurity.org/de/mitgliederbereich/glossary/w/web-proxies.html> (besucht am 01.02.2022).
- [63] Sakir Sezer u. a. „Are we ready for SDN? Implementation challenges for software-defined networks“. In: *IEEE Communications Magazine* 51.7 (2013), S. 36–43. DOI: 10.1109/MCOM.2013.6553676.
- [64] Blu Smurf. *XFRM—A Kernel Implementation Framework of IPsec Protocol*. 2019. URL: <https://programmer.ink/think/xfrm-a-kernel-implementation-framework-of-ipsec-protocol.html> (besucht am 08.01.2022).
- [65] sonicwall. *Working with the Mininet VM*. URL: <https://www.sonicwall.com/support/knowledge-base/how-can-i-configure-nat-over-vpn-in-a-site-to-site-vpn/170515155805172/> (besucht am 04.12.2021).
- [66] TechTarget. *quality of service(QoS)*. URL: <https://www.techtarget.com/searchunifiedcommunications/definition/QoS-Quality-of-Service> (besucht am 24.01.2022).
- [67] Telekom. *Company Start 500 (Festnetz)*. URL: <https://geschaeftskunden.telekom.de/hilfe-und-service/hilfe-themen/downloads/produktinformationsblaetter/company-start-500> (besucht am 15.01.2022).
- [68] Telekom. *Internet- und DSL-Tarife*. URL: <https://geschaeftskunden.telekom.de/internet-dsl/tarife/festnetz-internet-dsl> (besucht am 15.01.2022).
- [69] Bob Lantz u.a. *Introduction to Mininet*. 2021. URL: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet> (besucht am 08.12.2021).
- [70] Ubuntuusers. *resolvconf*. URL: <https://wiki.ubuntuusers.de/resolvconf/> (besucht am 11.01.2022).
- [71] VirtualBox. *Network Address Translation*. URL: <https://www.virtualbox.org/manual/ch06.html> (besucht am 11.01.2022).
- [72] Vodafone. *Internet- und DSL-Tarife*. URL: <https://zuhauseplus.vodafone.de/business/dsl/> (besucht am 15.01.2022).

Literaturverzeichnis

- [73] Open vSwitch. *Production Quality, Multilayer Open Virtual Switch*. URL: <https://www.openvswitch.org/> (besucht am 26.12.2021).
- [74] Open vSwitch. *Quality of Service (QoS)*. URL: <https://docs.openvswitch.org/en/latest/faq/qos/> (besucht am 12.12.2022).
- [75] Open vSwitch. *Quality of Service (QoS)*. URL: <https://docs.openvswitch.org/en/latest/faq/qos/> (besucht am 20.01.2022).
- [76] Open vSwitch. *Using OpenFlow*. URL: <https://docs.openvswitch.org/en/latest/faq/openflow/> (besucht am 21.01.2022).
- [77] Qing Wang. *Floodlight Controller Installation Guide*. URL: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343544/Installation+Guide> (besucht am 20.11.2021).
- [78] Github Wiki. *Openflow Tutorial*. URL: <https://github-wiki-see.page/m/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch> (besucht am 27.12.2021).

A. Anhang

```
1 from mininet.topo import Topo
2 from mininet.net import Mininet
3 from mininet.util import dumpNodeConnections, quietRun
4 from mininet.log import setLogLevel, info, info, error, debug,
5     output, warn
6 from mininet.node import Controller, RemoteController,
7     OVSSwitch
8 from mininet.link import TCLink, Intf
9 from mininet.cli import CLI
10 from mininet.node import Node, Switch
11 import time, re, os
12
13 class Router(Node):
14     def config(self, **params):
15         super(Router, self).config(**params)
16         self.cmd('sysctl net.ipv4.ip_forward=1')
17
18     def terminate(self):
19         self.cmd('sysctl net.ipv4.ip_forward=0')
20         super(Router, self).terminate()
21
22 def checkIntf(intf):
23     "Make sure intf exists and is not configured."
24     config = quietRun('ifconfig %s 2>/dev/null' % intf, shell=True)
25     if not config:
26         error('Error: ', intf, 'does not exist!\n')
27         exit(1)
28     ips = re.findall(r'\d+\.\d+\.\d+\.\d+', config)
29     if ips:
30         error('Error: ', intf, 'has an IP address, '
31               'and is probably in use!\n')
32         exit(1)
```

A. Anhang

```
32 class Netzwerk(Topo):
33     def build(self, **_opts):
34
35         defaultIP = '192.168.%s.1/24'
36
37         routers = []
38
39         for r in range(4):
40             router = self.addNode('r%s' % (r+1),
41                                   cls=Router, ip=defaultIP % (r
42                                         +1),
43                                         mac='00:00:00:00:00:0%s' % (r
44                                         +1)
45                                         )
46
47             routers.append(router)
48
49             switch = self.addSwitch('s%s' % (r+1))
50
51             for h in range(10):
52                 name = ((r)*10)+(h+1)
53                 host = self.addHost(name='h%s' % (name),
54                                   ip='0.0.0.0',
55                                   defaultRoute='via 192.168.%s.1' % (r+1),
56                                   mac='00:00:00:00:0%s:0%s' % (r+1, h)
57                                         )
58                 self.addLink(host, switch)
59
60             self.addLink(routers[0],
61                         routers[1],
62                         intfName1='r1-eth2',
63                         intfName2='r2-eth2',
64                         params1={'ip': '10.100.12.1/24'},
65                         params2={'ip': '10.100.12.2/24'},
66                         bw=20
67                         )
68             self.addLink(routers[2],
69                         routers[3],
70                         intfName1='r3-eth2',
71                         intfName2='r4-eth2',
```

A. Anhang

```
72     params1={'ip': '10.100.34.3/24'},  
73     params2={'ip': '10.100.34.4/24'},  
74     bw=20  
75     )  
76     self.addLink(routers[0],  
77                   routers[2],  
78                   intfName1='r1-eth3',  
79                   intfName2='r3-eth3',  
80                   params1={'ip': '10.100.13.1/24'},  
81                   params2={'ip': '10.100.13.3/24'},  
82                   bw=20  
83     )  
84     self.addLink(routers[0],  
85                   routers[3],  
86                   intfName1='r1-eth4',  
87                   intfName2='r4-eth3',  
88                   params1={'ip': '10.100.14.1/24'},  
89                   params2={'ip': '10.100.14.4/24'},  
90                   bw=20  
91     )  
92     self.addLink(routers[1],  
93                   routers[2],  
94                   intfName1='r2-eth3',  
95                   intfName2='r3-eth4',  
96                   params1={'ip': '10.100.23.2/24'},  
97                   params2={'ip': '10.100.23.3/24'},  
98                   bw=20  
99     )  
100    self.addLink(routers[1],  
101                  routers[3],  
102                  intfName1='r2-eth4',  
103                  intfName2='r4-eth4',  
104                  params1={'ip': '10.100.24.2/24'},  
105                  params2={'ip': '10.100.24.4/24'},  
106                  bw=20  
107     )  
108  
109 def Main():  
110     topo = Netzwerk()  
111  
112     os.system("sudo ovs-vsctl add-br s1")  
113     os.system("sudo ovs-vsctl add-br s2")  
114     os.system("sudo ovs-vsctl add-br s3")  
115     os.system("sudo ovs-vsctl add-br s4")
```

A. Anhang

```
116     os.system("sudo ip addr add 192.168.1.20/24 dev s1")
117     os.system("sudo ip link set s1 up")
118     os.system("sudo ip addr add 192.168.2.20/24 dev s2")
119     os.system("sudo ip link set s2 up")
120     os.system("sudo ip addr add 192.168.3.20/24 dev s3")
121     os.system("sudo ip link set s3 up")
122     os.system("sudo ip addr add 192.168.4.20/24 dev s4")
123     os.system("sudo ip link set s4 up")
124
125     os.system("sudo ip route add default via 192.168.1.1")
126
127 c0 = RemoteController('c0', controller=RemoteController,
128                      ip="192.168.1.20", port=6653)
129
130 net = Mininet(topo=topo, controller=c0,
131                link=TCLink, switch=OVSSwitch, waitConnected=
132                True)
133
134 info(net['r1'].cmd(
135     "ip tunnel add gre12 mode gre local 10.100.12.1 remote
136         10.100.12.2 ttl 255"))
137 info(net['r1'].cmd("ip link set gre12 up"))
138 info(net['r1'].cmd("ip addr add 10.10.12.1/24 dev gre12"))
139
140 info(net['r1'].cmd(
141     "ip tunnel add gre13 mode gre local 10.100.13.1 remote
142         10.100.13.3 ttl 255"))
143 info(net['r1'].cmd("ip link set gre13 up"))
144 info(net['r1'].cmd("ip addr add 10.10.13.1/24 dev gre13"))
145
146 info(net['r1'].cmd(
147     "ip tunnel add gre14 mode gre local 10.100.14.1 remote
148         10.100.14.4 ttl 255"))
149 info(net['r1'].cmd("ip link set gre14 up"))
150 info(net['r1'].cmd("ip addr add 10.10.14.1/24 dev gre14"))
151
152 info(net['r2'].cmd(
153     "ip tunnel add gre21 mode gre local 10.100.12.2 remote
154         10.100.12.1 ttl 255"))
155 info(net['r2'].cmd("ip link set gre21 up"))
156 info(net['r2'].cmd("ip addr add 10.10.12.2/24 dev gre21"))
```

A. Anhang

```
154     "ip tunnel add gre23 mode gre local 10.100.23.2 remote  
155         10.100.23.3 ttl 255"))  
156     info(net['r2'].cmd("ip link set gre23 up"))  
157     info(net['r2'].cmd("ip addr add 10.10.23.2/24 dev gre23"))  
158  
159     info(net['r2'].cmd(  
160         "ip tunnel add gre24 mode gre local 10.100.24.2 remote  
161             10.100.24.4 ttl 255"))  
162     info(net['r2'].cmd("ip link set gre24 up"))  
163     info(net['r2'].cmd("ip addr add 10.10.24.2/24 dev gre24"))  
164  
165     info(net['r3'].cmd(  
166         "ip tunnel add gre31 mode gre local 10.100.13.3 remote  
167             10.100.13.1 ttl 255"))  
168     info(net['r3'].cmd("ip link set gre31 up"))  
169     info(net['r3'].cmd("ip addr add 10.10.13.3/24 dev gre31"))  
170  
171     info(net['r3'].cmd(  
172         "ip tunnel add gre32 mode gre local 10.100.23.3 remote  
173             10.100.23.2 ttl 255"))  
174     info(net['r3'].cmd("ip link set gre32 up"))  
175     info(net['r3'].cmd("ip addr add 10.10.23.3/24 dev gre32"))  
176  
177     info(net['r3'].cmd(  
178         "ip tunnel add gre34 mode gre local 10.100.34.3 remote  
179             10.100.34.4 ttl 255"))  
180     info(net['r3'].cmd("ip link set gre34 up"))  
181     info(net['r3'].cmd("ip addr add 10.10.34.3/24 dev gre34"))  
182  
183     info(net['r4'].cmd(  
184         "ip tunnel add gre41 mode gre local 10.100.14.4 remote  
185             10.100.14.1 ttl 255"))  
186     info(net['r4'].cmd("ip link set gre41 up"))  
187     info(net['r4'].cmd("ip addr add 10.10.14.4/24 dev gre41"))  
188  
189     info(net['r4'].cmd(  
190         "ip tunnel add gre42 mode gre local 10.100.24.4 remote  
191             10.100.24.2 ttl 255"))  
192     info(net['r4'].cmd("ip link set gre42 up"))  
193     info(net['r4'].cmd("ip addr add 10.10.24.4/24 dev gre42"))  
194  
195     info(net['r4'].cmd(  
196         "ip tunnel add gre43 mode gre local 10.100.34.4 remote  
197             10.100.34.3 ttl 255"))
```

A. Anhang

```
190 info(net['r4'].cmd("ip link set gre43 up"))
191 info(net['r4'].cmd("ip addr add 10.10.34.4/24 dev gre43"))
192
193 info(net['r1'].cmd("ip route add 192.168.2.0/24 via
194     10.10.12.2 dev gre12"))
195 info(net['r2'].cmd("ip route add 192.168.1.0/24 via
196     10.10.12.1 dev gre21"))
197
198 info(net['r1'].cmd("ip route add 192.168.3.0/24 via
199     10.10.13.1 dev gre13"))
200 info(net['r3'].cmd("ip route add 192.168.1.0/24 via
201     10.10.13.3 dev gre31"))
202
203 info(net['r1'].cmd("ip route add 192.168.4.0/24 via
204     10.10.14.1 dev gre14"))
205 info(net['r4'].cmd("ip route add 192.168.1.0/24 via
206     10.10.14.4 dev gre41"))
207
208 info(net['r2'].cmd("ip route add 192.168.3.0/24 via
209     10.10.23.2 dev gre23"))
210 info(net['r3'].cmd("ip route add 192.168.2.0/24 via
211     10.10.23.3 dev gre32"))
212
213 info(net['r2'].cmd("ip route add 192.168.4.0/24 via
214     10.10.24.2 dev gre24"))
215 info(net['r4'].cmd("ip route add 192.168.2.0/24 via
216     10.10.24.4 dev gre42"))
217
218 info(net['r3'].cmd("ip route add 192.168.4.0/24 via
219     10.10.34.3 dev gre34"))
220 info(net['r4'].cmd("ip route add 192.168.3.0/24 via
221     10.10.34.4 dev gre43"))
222
223 key12 = "0xf1e125c62a8f68169ff9d0375d901\
224 7b76c700354e3060ef78a61e43547babd0d"
225 key21 = "0xc9d786b186394addc49c08fc551cd\
226 5cf580f7cd600da9ed5c47d1ab7b7e510ec"
227 spi12 = "0x1cac64a1"
228 spi21 = "0xd03e561e"
229
230 info(net['r1'].cmd("ip xfrm state add src 10.100.12.1 dst
231     10.100.12.2 proto esp spi " +
232             spi12 + " enc 'cbc(aes)' " + key12 + "
233                 mode transport"))
```

A. Anhang

```
220 info(net['r1'].cmd("ip xfrm state add src 10.100.12.2 dst  
221     10.100.12.1 proto esp spi " +  
222         spi21 + " enc 'cbc(aes)' " + key21 + "  
223             mode transport"))  
224  
225 info(net['r2'].cmd("ip xfrm state add src 10.100.12.1 dst  
226     10.100.12.2 proto esp spi " +  
227         spi12 + " enc 'cbc(aes)' " + key12 + "  
228             mode transport"))  
229 info(net['r2'].cmd("ip xfrm state add src 10.100.12.2 dst  
230     10.100.12.1 proto esp spi " +  
231         spi21 + " enc 'cbc(aes)' " + key21 + "  
232             mode transport"))  
233  
234 info(net['r1'].cmd(  
235     "ip xfrm policy add dir out src 10.100.12.1 dst  
236         10.100.12.2 tmpl proto esp mode transport"))  
237 info(net['r1'].cmd(  
238     "ip xfrm policy add dir in src 10.100.12.2 dst  
239         10.100.12.1 tmpl proto esp mode transport"))  
240 info(net['r2'].cmd(  
241     "ip xfrm policy add dir out src 10.100.12.2 dst  
242         10.100.12.1 tmpl proto esp mode transport"))  
243  
244  
245 info(net['r2'].cmd("ip xfrm state add src 10.100.13.1 dst  
246     10.100.13.3 proto esp spi " +  
247         spi13 + " enc 'cbc(aes)' " + key13 + "  
248             mode transport"))  
249 info(net['r1'].cmd("ip xfrm state add src 10.100.13.3 dst  
250     10.100.13.1 proto esp spi " +  
251         spi31 + " enc 'cbc(aes)' " + key31 + "  
252             mode transport"))  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1898  
1899  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
19
```

A. Anhang

```
250 info(net['r3'].cmd("ip xfrm state add src 10.100.13.1 dst  
251           10.100.13.3 proto esp spi " +  
252           spi13 + " enc 'cbc(aes)' " + key13 + "  
253           mode transport"))  
254 info(net['r3'].cmd("ip xfrm state add src 10.100.13.3 dst  
255           10.100.13.1 proto esp spi " +  
256           spi31 + " enc 'cbc(aes)' " + key31 + "  
257           mode transport"))  
258 info(net['r1'].cmd(  
259           "ip xfrm policy add dir out src 10.100.13.1 dst  
260           10.100.13.3 tmpl proto esp mode transport"))  
261 info(net['r1'].cmd(  
262           "ip xfrm policy add dir in src 10.100.13.3 dst  
263           10.100.13.1 tmpl proto esp mode transport"))  
264 info(net['r3'].cmd(  
265           "ip xfrm policy add dir out src 10.100.13.3 dst  
266           10.100.13.1 tmpl proto esp mode transport"))  
267 info(net['r3'].cmd(  
268           "ip xfrm policy add dir in src 10.100.13.1 dst  
269           10.100.13.3 tmpl proto esp mode transport"))  
270  
271 key14 = "0x8527ef297dc35bed418d635ef15 \  
272 e219feaf5f5597699b6271534697bfd940c9"  
273 key41 = "0x69091e7f7c1c162c7c6b44fb8e3 \  
274 89e113eb77746c1d6a2d73074e9609a991179"  
275 spi14 = "0x19de2473"  
276 spi41 = "0x65cb9866"  
277  
278 info(net['r1'].cmd("ip xfrm state add src 10.100.14.1 dst  
279           10.100.14.4 proto esp spi " +  
280           spi14 + " enc 'cbc(aes)' " + key14 + "  
281           mode transport"))  
282 info(net['r1'].cmd("ip xfrm state add src 10.100.14.4 dst  
283           10.100.14.1 proto esp spi " +  
284           spi41 + " enc 'cbc(aes)' " + key41 + "  
285           mode transport"))  
286  
287 info(net['r4'].cmd("ip xfrm state add src 10.100.14.1 dst  
288           10.100.14.4 proto esp spi " +  
289           spi14 + " enc 'cbc(aes)' " + key14 + "  
290           mode transport"))
```

A. Anhang

```
279 info(net['r4'].cmd("ip xfrm state add src 10.100.14.4 dst  
280           10.100.14.1 proto esp spi " +  
281           spi41 + " enc 'cbc(aes)' " + key41 + "  
282           mode transport"))  
283  
284 info(net['r1'].cmd(  
285           "ip xfrm policy add dir out src 10.100.14.1 dst  
286           10.100.14.4 tmpl proto esp mode transport"))  
287 info(net['r1'].cmd(  
288           "ip xfrm policy add dir in src 10.100.14.4 dst  
289           10.100.14.1 tmpl proto esp mode transport"))  
290  
291  
292 key23 = "0xe6709e851cc4f247729bb592147\"  
293 663ab4e4fe26cced514120e92aaaf3034061f8"  
294 key32 = "0xc765d3e9382d7012963fc2c0d66\"  
295 e20724ba2de74928e6c9f08c0250d6ac5f823"  
296 spi23 = "0x18d7e3e4"  
297 spi32 = "0x93e5489f"  
298  
299 info(net['r2'].cmd("ip xfrm state add src 10.100.23.2 dst  
300           10.100.23.3 proto esp spi " +  
301           spi23 + " enc 'cbc(aes)' " + key23 + "  
302           mode transport"))  
303 info(net['r2'].cmd("ip xfrm state add src 10.100.23.3 dst  
304           10.100.23.2 proto esp spi " +  
305           spi32 + " enc 'cbc(aes)' " + key32 + "  
306           mode transport"))  
307 info(net['r3'].cmd("ip xfrm state add src 10.100.23.2 dst  
308           10.100.23.3 proto esp spi " +  
309           spi23 + " enc 'cbc(aes)' " + key23 + "  
310           mode transport"))  
311 info(net['r3'].cmd("ip xfrm state add src 10.100.23.3 dst  
312           10.100.23.2 proto esp spi " +  
313           spi32 + " enc 'cbc(aes)' " + key32 + "  
314           mode transport"))
```

```

309 info (net['r2'].cmd(
310     "ip xfrm policy add dir out src 10.100.23.2 dst
311         10.100.23.3 tmpl proto esp mode transport"))
312 info (net['r2'].cmd(
313     "ip xfrm policy add dir in src 10.100.23.3 dst
314         10.100.23.2 tmpl proto esp mode transport"))
315
316 info (net['r3'].cmd(
317     "ip xfrm policy add dir out src 10.100.23.3 dst
318         10.100.23.2 tmpl proto esp mode transport"))
319 key24 = "0xd2851d694a952d4e14b1eda4ee20 \
320 04e94f601e7422e47c1872ad6d333b7e1d37"
321 key42 = "0xcb7698938b9393686afde8b29e2c \
322 1e620b99f1fe2435c24709a9ffccfea050f6"
323 spi24 = "0x7d99d7e8"
324 spi42 = "0x508ef1f2"
325
326 info (net['r2'].cmd("ip xfrm state add src 10.100.24.2 dst
327         10.100.24.4 proto esp spi " +
328             spi24 + " enc 'cbc(aes)' " + key24 + "
329                 mode transport"))
330 info (net['r2'].cmd("ip xfrm state add src 10.100.24.4 dst
331         10.100.24.2 proto esp spi " +
332             spi42 + " enc 'cbc(aes)' " + key42 + "
333                 mode transport"))
334
335 info (net['r4'].cmd("ip xfrm state add src 10.100.24.2 dst
336         10.100.24.4 proto esp spi " +
337             spi24 + " enc 'cbc(aes)' " + key24 + "
338                 mode transport"))
339 info (net['r2'].cmd(
340     "ip xfrm policy add dir out src 10.100.24.2 dst
341         10.100.24.4 tmpl proto esp mode transport"))
342 info (net['r2'].cmd(

```

```

339         "ip xfrm policy add dir in src 10.100.24.4 dst
340             10.100.24.2 tmpl proto esp mode transport"))
341     info (net['r4'].cmd(
342         "ip xfrm policy add dir out src 10.100.24.4 dst
343             10.100.24.2 tmpl proto esp mode transport"))
344     info (net['r4'].cmd(
345         "ip xfrm policy add dir in src 10.100.24.2 dst
346             10.100.24.4 tmpl proto esp mode transport"))
347
348     key34 = "0xa5036e96c1ee40de3cb7ebc6a455f\
349             a816053b6106a352634da87e67b1137c058"
350     key43 = "0x4e2ea7cb3ec6d11704d2a85b7f7db\
351             3518ddcf970ff54502ff8ea6be653c6b456"
352     spi34 = "0x7302a4a9"
353     spi43 = "0xc70a7221"
354
355     info (net['r3'].cmd("ip xfrm state add src 10.100.34.3 dst
356             10.100.34.4 proto esp spi " +
357                 spi34 + " enc 'cbc(aes)' " + key34 + "
358                     mode transport"))
359     info (net['r3'].cmd("ip xfrm state add src 10.100.34.4 dst
360             10.100.34.3 proto esp spi " +
361                 spi43 + " enc 'cbc(aes)' " + key43 + "
362                     mode transport"))
363
364     info (net['r3'].cmd(
365         "ip xfrm policy add dir out src 10.100.34.3 dst
366             10.100.34.4 tmpl proto esp mode transport"))
367     info (net['r4'].cmd(
368

```

```

369     "ip xfrm policy add dir out src 10.100.34.4 dst
370         10.100.34.3 tmpl proto esp mode transport"))
370 info (net['r4'].cmd(
371     "ip xfrm policy add dir in src 10.100.34.3 dst
372         10.100.34.4 tmpl proto esp mode transport"))
372
373 info ('*** Routing Table on Router:\n')
374 info (net['r1'].cmd('route'))
375 info (net['r2'].cmd('route'))
376 info (net['r3'].cmd('route'))
377 info (net['r4'].cmd('route'))
378
379 checkIntf('enp0s9')
380 s1 = net.getNodeByName('s1')
381 _intf = Intf('enp0s9', node=s1)
382
383 checkIntf('enp0s10')
384 s2 = net.getNodeByName('s2')
385 _intf = Intf('enp0s10', node=s2)
386
387 net.start()
388
389 print("Dumping host connections")
390 dumpNodeConnections(net.hosts)
391 print("Testing network connectivity")
392
393 r1 = net.getNodeByName('r1')
394 Intf('enp0s16', node=r1)
395 info (net['r1'].cmd("dhclient enp0s16"))
396 info (net['r1'].cmd("sudo iptables -t nat -A POSTROUTING -o
397             enp0s16 -j MASQUERADE"))
398
398 r2 = net.getNodeByName('r2')
399 Intf('enp0s17', node=r2)
400 info (net['r2'].cmd("dhclient enp0s17"))
401 info (net['r2'].cmd("sudo iptables -t nat -A POSTROUTING -o
402             enp0s17 -j MASQUERADE"))
403
403 r3 = net.getNodeByName('r3')
404 Intf('enp0s18', node=r3)
405 info (net['r3'].cmd("dhclient enp0s18"))
406 info (net['r3'].cmd("sudo iptables -t nat -A POSTROUTING -o
407             enp0s18 -j MASQUERADE"))

```

```

408     r4 = net.getNodeByName('r4')
409     Intf('enp0s19', node=r4)
410     info(net['r4'].cmd("dhclient enp0s19"))
411     info(net['r4'].cmd("sudo iptables -t nat -A POSTROUTING -o
412         enp0s19 -j MASQUERADE"))
413
414     os.system("service isc-dhcp-server restart")
415
416     for i in range(40):
417         host = net.getNodeByName('h%s' % (i+1))
418         host.cmd("dhclient h%s-eth0" % (i+1))
419
420         os.system("sudo ovs-ofctl add-flow s1 priority=1000,actions
421             =set_queue:0,normal")
422         os.system("sudo ovs-ofctl add-flow s2 priority=1000,actions
423             =set_queue:0,normal")
424         os.system("sudo ovs-ofctl add-flow s3 priority=1000,actions
425             =set_queue:0,normal")
426         os.system("sudo ovs-ofctl add-flow s4 priority=1000,actions
427             =set_queue:0,normal")
428
429     CLI(net)
430
431     info(net['r1'].cmd("ip link set enp0s16 netns 1"))
432     info(net['r2'].cmd("ip link set enp0s17 netns 1"))
433     info(net['r3'].cmd("ip link set enp0s18 netns 1"))
434     info(net['r4'].cmd("ip link set enp0s19 netns 1"))
435
436     os.system("sudo ip addr del 192.168.1.20/24 dev s1")
437     os.system("sudo ip addr del 192.168.2.20/24 dev s2")
438     os.system("sudo ip addr del 192.168.3.20/24 dev s3")

```

```
439     os.system("sudo ip addr del 192.168.4.20/24 dev s4")
440
441     net.stop()
442
443     os.system("sudo mn -c")
444
445
446 if __name__ == '__main__':
447     setLogLevel('info')
448     Main()
```

Listing A.1: Das in Python geschrieben Mininet-Skript

```

1 import socket
2 import ipaddress
3 import http.client
4 import json
5 import time
6 import paramiko
7 from scp import SCPClient
8
9 class StaticEntryPusher(object):
10
11     def __init__(self, server):
12         self.server = server
13
14     def get(self, data):
15         ret = self.rest_call({}, 'GET')
16         return json.loads(ret[2])
17
18     def set(self, data):
19         ret = self.rest_call(data, 'POST')
20         return ret[0] == 200
21
22     def remove(self, data):
23         ret = self.rest_call(data, 'DELETE')
24         return ret[0] == 200
25
26     def rest_call(self, data, action):
27         path = '/wm/staticentrypusher/json'
28         headers = {
29             'Content-type': 'application/json',
30             'Accept': 'application/json',
31         }
32         body = json.dumps(data)
33         conn = http.client.HTTPConnection(self.server, 8080)
34         conn.request(action, path, body, headers)
35         response = conn.getresponse()
36         ret = (response.status, response.reason, response.read())
37         print(ret)
38         conn.close()
39         return ret
40
41     def check_file(path):
42         try:

```

```

43     f = open(path)
44     return True
45 except FileNotFoundError:
46     print("Solch eine Datei ist nicht vorhanden!")
47     return False
48 finally:
49     try:
50         f.close()
51     except UnboundLocalError:
52         pass
53
54 def check_ip(address):
55     try:
56         ip = IPAddress.ip_address(address)
57         print("Die IP-Adresse {} ist gueltig.".format(address))
58         return True
59     except ValueError:
60         print("Die IP-Adresse {} ist nicht gueltig.".format(
61             address))
62         return False
63
64 def get_ip():
65     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
66     s.settimeout(0)
67     try:
68         s.connect(( '192.168.2.1' , 1))
69         IP = s.getsockname() [0]
70     except Exception:
71         IP = '127.0.0.1'
72     finally:
73         s.close()
74     return IP
75
76 source_ip = get_ip()
77
78 valid = False
79 while not valid:
80     file_path = str(input(
81         "Pfad der Datei zum Transfer eingeben (Beispiel:/home/
82             username/Schreibtisch/file)\n"))
83     valid = check_file(file_path)
84
85 valid = False
86 while not valid:

```

```

85 destination_ip = str(
86     input("IPv4-Adresse des Ziels eingeben (Beispiel
87         :192.168.1.1)\n"))
88 valid = check_ip(destination_ip)
89
90 destination_host_name = str(input('Hostname fuer die IP %s
91         eingeben\n' % destination_ip))
92
93 destination_pass = str(input('Passwort fuer Host %s eingeben\n'
94         '%s destination_host_name'))
95
96 pusher=StaticEntryPusher('192.168.1.20')
97
98 prio1_dst={
99     'switch': "00:00:00:00:00:00:00:01",
100    "name": "prio1_dst",
101    "priority": "32768",
102        "eth_type": "0x0800",
103        "ip_proto": "6",
104        "tcp_dst": "22",
105        "ipv4_dst": destination_ip,
106        "ipv4_src": source_ip,
107        "active": "true",
108        "actions": "output=normal"
109    }
110
111 prio1_src={
112     'switch': "00:00:00:00:00:00:00:01",
113    "name": "prio1_src",
114    "priority": "32768",
115        "eth_type": "0x0800",
116        "ip_proto": "6",
117        "tcp_src": "22",
118        "ipv4_dst": source_ip,
119        "ipv4_src": destination_ip,
120        "active": "true",
121        "actions": "output=normal"
122    }
123
124 dros1={
```

```

125     'switch': "00:00:00:00:00:00:00:00:01",
126     "name": "dros1",
127     "priority": "1050",
128     "eth_type": "0x0800",
129     "active": "true",
130     "actions": "set_queue=2,output=normal"
131 }
132
133 prio2_dst={
134     'switch': "00:00:00:00:00:00:00:00:02",
135     "name": "prio2_dst",
136     "priority": "32768",
137     "eth_type": "0x0800",
138     "ip_proto": "6",
139     "tcp_dst": "22",
140     "ipv4_dst": destination_ip,
141     "ipv4_src": source_ip,
142     "active": "true",
143     "actions": "output=normal"
144 }
145
146 prio2_src={
147     'switch': "00:00:00:00:00:00:00:00:02",
148     "name": "prio2_src",
149     "priority": "32768",
150     "eth_type": "0x0800",
151     "ip_proto": "6",
152     "tcp_src": "22",
153     "ipv4_dst": source_ip,
154     "ipv4_src": destination_ip,
155     "active": "true",
156     "actions": "output=normal"
157 }
158
159 dros2={
160     'switch': "00:00:00:00:00:00:00:00:02",
161     "name": "dros2",
162     "priority": "1050",
163     "eth_type": "0x0800",
164     "active": "true",
165     "actions": "set_queue=2,output=normal"
166 }
167
168 pusher.set(prio1_dst)

```

A. Anhang

```
169 pusher.set(prio1_src)
170 pusher.set(dros1)
171 pusher.set(prio2_dst)
172 pusher.set(prio2_src)
173 pusher.set(dros2)
174
175 def createSSHClient(server, port, user, password):
176     client=paramiko.SSHClient()
177     client.load_system_host_keys()
178     client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
179     client.connect(server, port, user, password)
180     return client
181
182 ssh=createSSHClient(server=destination_ip, port=22,
183                      user=destination_host_name, password=
184                      destination_pass)
185 scp=SCPClient(ssh.get_transport())
186 scp.put(file_path, remote_path=file_path_remote,
187          recursive=False, preserve_times=False)
```

Listing A.2: Das in Python geschrieben SSH-Transfer-Skript