Network Unit Testing System

Studienarbeit FS-2020

26. Mai 2020

Autoren:

Mike Schmid
mike.schmid@hsr.ch

Janik Schlatter
janik.schlatter@hsr.ch

Supervisors:

Prof. Stettler Beat
beat.stettler@hsr.ch

Baumann Urs
urs.baumann@hsr.ch

Aufgabenstellung

Änderungen an Netzwerkumgebungen werden in der Praxis auch heute noch durch Kommandozeilenbefehle oder Skripte getestet. Diese Tests beinhalten oft einfache Befehle wie 'ping' oder 'traceroute'. Im Vergleich dazu werden Softwareprojekte durch automatisierte Tests, welche regelmässig ausgeführt werden, getestet. Sogenannte Unit Tests werden vor und nach einer Änderung durchgeführt, um zu Testen, ob sich ein Programm weiterhin innerhalb der geforderten Betriebsparameter verhält. Somit können Fehler schnell gefunden und behoben werden und die Robustheit der Software wird erhöht. Ein vergleichbarer Arbeitsablauf soll auch für den Netzwerkbereich ermöglicht werden.

Eine frühere Studienarbeit hat sich mit der Entwicklung einer Beschreibungssprache befasst, mit derer solche Tests möglich wären. Die Vorarbeit wurde dabei so entwickelt, dass das Programm mit dem Automationsframework SaltStack ausgeführt wurde. Diese Arbeit soll ein Programm entwickeln, welches selbstständig und unabhängig von anderen Programmen arbeiten kann.

Die Studierenden erhalten die Aufgabe, ein Python-Programm zu entwickeln, welches automatisierte Tests auf ein Netzwerk durchführen kann. Das Programm soll gemäss einer Testdefinition selbstständig die auszuführenden Tests erstellen, durchführen und die Testresultate mit einem Erwartungswert vergleichen. Die Auswertung der Tests soll direkt bei der Ausführung auf der Konsole angezeigt werden und zusätzlich in einem Testreport für die spätere Ansicht gespeichert werden. Die Programmausführung kann manuell oder automatisch über einen Deployment-Prozess gestartet werden.

Prof.Beat Stettler

Urs Baumann

Abstract

TODO

Management Summary

Ausgangslage

Fehler in Teilbereichen von Netzwerksystemen können dazu führen, dass das ganze System nicht mehr funktioniert. Aus diesem Grund ist es essentiell, dass selbst kleine Änderungen an Netzwerken getestet werden können. Diese Tests werden meistens von Hand oder durch Skripte durchgeführt. Ein Tool, welches das automatisierte Testen von Netzwerksystemen ermöglicht, kann dabei helfen, Fehler zu erkennen, bevor sie zu einem Problem werden.

Vorgehen, Technologien

Zu Beginn wurde eine Domänenanalyse durchgeführt, um die Akteure und Bestandteile einer Netzwerkumgebung zu bestimmen und die zu entwickelnden Netzwerktests zu evaluieren. Darauf aufbauend wurden die funktionalen und nichtfunktionalen Anforderungen an die Software spezifiziert. Auf dieser Basis wurde die Softwarearchitekur ausgearbeitet und mit der Entwicklung begonnen.

Das Programm wurde in der Programmiersprache Python geschrieben und beinhaltet das Modul "Nornir", ein Framework, welches automatisierte Tasks auf Netzwerksysteme, wie z.B. Konfiguration oder Tests, ermöglicht.

Ergebnisse

Aus dieser Studienarbeit ist ein Python-Programm entstanden, welches Netzwerktests, die in einer Definitionssprache spezifiziert werden, gegen ein Netzwerk durchführt und die Ergebnisse selbstständig auswertet und dem Benutzer anzeigt. Nornir erlaubt dabei, eine Vielzahl von Netzwerkgräten anzusprechen, welche über herkömmliche Methoden wie SSH umfänglicher zu testen wären.

Die Software lässt sich ohne Installation auf jedem Gerät ausführen, welches Python-Code ausführen kann, unabhängig vom Betriebssystem. Geräte, auf denen Python nicht installiert ist, müssen dies zuerst Installieren, können das Programm danach aber ohne weiteres ausführen.

Dadurch, dass das Programm reiner Pyhton Code ist, lässt es sich einfach in ein bestehendes Tool für die kontinuierliche Integration einbinden. Die Testdefinitionen lassen sich über ein Versionsverwaltungstool zentralisieren, so dass mehrere Netzwerkleute gleichzeitig Tests für eine Umgebung entwickeln können.

Inhaltsverzeichnis

I. Technischer Bericht

1 Einleitung

1.1 Problemstellung

Netzwerke bestehen aus dutzenden bis tausenden Komponenten. Jede dieser Komponente hat eine eigene Konfiguration und Aufgabe. In den letzten Jahren ist es durch die softwaregesteuerte Konfiguration von Netzwerkgeräten einfacher geworden, die Komponenten für ihre Aufgabe einzustellen. Trotzdem werden die Überprüfungen der Konfiguration auch heute noch manuell vorgenommen. Dies kann dazu führen, dass wegen menschlicher Fehler ein Fehlverhalten eines der Netzwerkkomponente dazu führt, dass das gesammte Netzwerk gestört wird. Weiterhin wird die Überprüfung noch komplizierter, da neben der statischen Konfiguration sich das Netzwerk zur Laufzeit dynamisch anpasst, um die Performanz des Systems zu optimieren und Fehler sowie Ausfälle zu korrigieren. Dieses Verhalten wird über verschiedene Netzwerkprotokolle gesteuert, z.B. OSPF oder BGP.

In der Softwareentwicklung werden schon seit Ende der 80er-Jahre Komponententests, sogenannte Unit-Tests durchgeführt, um einzelne Komponenten (Units) automatisiert zu Testen. Dabei wird ein Computerprogramm ausgeführt, welches mit verschiedenen Eingabeparametern überprüft, ob die Ausgabe des zu testenden Programms den erwarteten Ergebnissen entspricht.

Dabei ist es möglich die Tests vor und nach einer geplanten Änderung durchzuführen, um zu überprüfen, ob die Software innerhalb der definierten Funktionsparameter operiert. Tests sollen dafür so geschrieben werden, dass möglichst jede Situation mit den Eingabeparametern abgebildet wird. Unittests sollen regelmässig durchgeführt werden, damit Fehler früh gefunden werden und sich nicht auf das gesammte System auswirken können.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist, ein Programm zu entwickeln, mit dem sich Netzwerktests mit der gleichen Arbeitsweise durchführen lassen, wie Unittests in der Softwareentwicklung durchgeführt werden. Dabei müssen folgende Anforderungen an die Tests erfüllt sein:

- Tests müssen planbar sein. Es soll ein Testplan existieren.
- Tests müssen systematisch spezifiziert werden. Es existieren Test-Spezifikationen.
- Testresultate werden Dokumentiert.
- Tests sollen, wo möglich, automatisiert durchgeführt werden.
- Testergebnisse müssen reproduzierbar und nachvollziehbar sein.

Es soll evaluiert werden, welche bereits verfügbaren Tools sich für ein solches Testprogramm eignen. Die Umsetzung soll diese Tools einbinden. Eine Wichtige Anforderung an das zu entwickelnde System ist, dass sich weitere Netzwerktests möglichst einfach hinzufügen lassen, ohne dass dafür der gesamte Code geändert werden muss.

1.3 Herausforderungen

Eine der grössten Herausforderungen an ein automatisiertes Testsystem sind die verschiedenen Protokolle und die Unterschiede der Standards von diversen Herstellern.

Ohne Kenntnisse, welche Protokolle auf einem Netzwerkgerät konfiguriert sind, können diese Netzwerkgeräte nicht effizient getestet werden, da die Netzwerkprotokolle das Verhalten der Geräte zur Laufzeit beeinflussen. Deshalb müssen für die Testdurchführung im vornherein die Konfigurationen und verwendeten Protokolle der Netzwerkgeräte bekannt sein und ein Testsystem muss mit diesen interagieren können.

Unterschiedliche Hersteller haben verschiedene Kommandozeilenbefehle (CLI-Commands) für ihre Geräte, welche für die Konfiguration und Abfrage der Konfiguration verwendet werden. Ausserdem kann es vorkommen, dass ein Hersteller mit der Einführung einer neuen Version des Gerätebetriebssystems neue CLI-Commands einführt oder alte Befehle ändert. Dies setzt eine enorme flexibilität für ein Testprogramm voraus, welches ein beliebiges Netzwerk mit unterschiedlichen Geräten von verschiedenen Herstellern testen soll.

1.4 Vorarbeit

Die Studienarbeit 'Network Unit Testing' aus dem Herbstsemester 2016 von Andreas Stalder und David Meister hat sich bereits mit dem Thema automatisierte Netzwerktests auseinandergesetzt. Der Fokus lag dabei auf dem Ausarbeiten einer Testdefinitionssprache, mit der solche Netzwerktests annotiert werden können. Zudem wurde mit SaltStack ein Konfigurationsmanagement-Tool evaluiert und auf dessen basis eine Software entwickelt, mit der man Netzwerktests ausführen konnte.

Die grössten Herausforderungen der Vorarbeit lagen dabei auf den Unterschieden verschiedener Hersteller und darin, dass die Outputformate der Neztwerktests je nach Hersteller anders zu Parsen sind. Diese Herausforderungen haben dazu geführt, dass beim Parsen der Ergebnisse oftmals auf reguläre Ausdrücke zurückgegriffen wurde, deren Implementation umständlich und schwierig zu lesen ist. Die Autoren haben dabei die Hoffnung angemerkt, dass man die Resultate künftig einfacher verarbeiten kann und die Hersteller ein einheitliches Rückgabeformat verwenden.

Der grösste Nachteil in der Verwendung der Lösung aus der Vorarbeit besteht darin, dass die Implementation neuer Netzwerktests eher umständlich ist. Ausserdem besteht eine starke Abhängigkeit zum Tool SaltStack, bei dem keine Garantie für eine langfristige Verfügbarkeit besteht.

Zukünftige Arbeiten sollen desshalb eine geringere Abhängigkeit zu externen Tools haben und die erweiterung um neue Tests so einfach wie möglich umsetzen.

2 Ergebnisse

2.1 Resultat der Arbeit

Die Studienarbeit hat ein Python-Programm entwickelt, welches basierend auf einer Testdefinition im YAML-Format automatische Tests gegen ein Netzwerk ausführen kann.

Kern der Software ist das Nornir-Modul für Python, ein Framework welches in Python geschrieben ist und die Automation von Netzwerktätigkeiten ermötlicht. Die Auswahl von Nornir ist das Ergebnis der Evaluation möglicher Tools für die Software, wobei schon zu Beginn der Analysephase die Verwendung von Nornir feststand, da das Framework ein breites Spektrum von Funktionen für die Durchführung und Auswertung von Netzwerktests bietet. Nornir erlaubt dem Anwender, automatisiert Konfigurationsänderungen oder -abfragen an ein Netzwerk zu senden.

Eine mögliche paralelle Ausführung der Netzwerktests wurde in der frühen Phase des Projekts angesprochen, konnte aber aufgrund des engen Zeitrahmens der Arbeit nicht umgesetzt werden. Stattdessen wurde die Zeit investiert, damit die Implementierten Tests korrekt funktionieren und die Erweiterung um neue Tests so einfach wie möglich ist. Die Erweiterung um eine paralelle Ausführung wird in die künftigen Arbeiten übernommen.

Die im Kapitel 1.2-Aufgabenstellung genannten Anforderungen an die Tests werden vom Programm vollständig erfüllt:

Planbarkeit von Tests/Test-Spezifikation Die Testdefinition entspricht dem Testplan/ der Test-Spezifikation. Netzwerktests werden in der Testdefinition spezifiziert und in der Defininitionsreihenfolge, sofern nicht über das Grafische Userinterface anders definiert, durchgeführt.

Dokumentation von Testresultaten Die Testresultate werden in einem txt-File mit dem Datum und Zeitpunkt der Durchführung gespeichert. Somit lassen sich sämtliche vergangenen Testdurchführungen nachvollziehen.

Automatische Testdurchführung NUTS2.0 lässt sich vollautomatisch durchführen. Eine Orchestrierungssoftware muss dafür lediglich den Kommandozeilenbefehl 'python -m nuts -r' zu festgelegten Zeitpunkten ausführen und das Programm läuft danach selbstständig durch und dokumentiert die Ergebnisse.

Reproduzierbarkeit/Nachvollziehbarkeit der Ergebnisse Wenn sich die Konfiguration im Netzwerk zwischen der Testdurchführung nicht verändert, wird auch das Testergebniss gleich ausfallen. Testergebnisse werden in einem einheitlichen Format gespeichert und bei nicht bestandenen Tests wird zusätzlich zum Testnamen noch das erwartete und das tatsächliche Ergebnis ausgegeben, um einen Soll-Ist-Vergleich durchführen zu können.

2.2 Programmausführung

Das Programm kann auf einem Beliebigen Betriebssystem, welches Python installiert hat, ausgeführt werden. Mit dem Befehl 'python -m nuts' im Order 'Nuts2.0' wird das Programm gestartet und lädt selbstständig die Informationen aus dem Inventar und die Testdefinitionen. Mit dem Kommandozeilenparameter '-r' kann dabei das Grafische Userinterface übersprungen werden, um Netzwerktests automatisiert ausführen zu lassen. Das GUI wird verwendet, um spezifische Tests zu selektieren und die Reihenfolge der selektierten Tests festzulegen. Wird das GUI übersprungen, werden alle Netzwerktests in der Reihenfolge durchgeführt, in der sie in der Testdefinition angegeben wurden.

Die Testergebnisse werden nach der Testdurchführung in der Konsole angezeigt und in einem Result.txt, unter angabe des Zeitstempels gespeichert. Erfolgreiche Tests werden nur mit dem Testnamen aus der Testdefinition und dem Vermerk 'PASSED' ausgegeben/gespeichert, da hauptsächlich die nicht bestandenen Netzwerktests relevant sind. Nicht bestandene Tests haben zusätzlich zum Testnamen aus der Testdefinition noch das erwartete Ergebniss und das tatsächliche Ergebniss für einen Soll-Ist-Vergleich.

2.3 Geräte-/Herstellerunterstützung

Das Modul Nornir erlaubt mit seinen Plugins eine breite Unterstützung von diversen Herstellergeräten. Verschiedene Plugins haben dabei ein unterschiedliches Rückgabeformat, welches in den konkreten Netzwerktests in eine Normalform gebracht werden muss, damit NUTS2.0 einheitlich damit umgehen kann.

Nachfolgend sind die verschiedenen Plugins für die Verbindungsschnittstellen beschrieben:

Napalm Abkürzung für "Network Automation and Programmability Abstraction Layer with Multivendor support". Napalm ist eine Pyhton Library welche verschiedene Funktionen anbietet, mit denen man mit Netzwerkgeräten über eine einheitliche Schnittstelle kommunizieren kann.

Paramiko Paramiko ist eine Python-Implementation des SSHv2 Protokolls für die sichere Kommunikation zwischen verschiedenen Endgeräten.

Netmiko Netmiko ist eine Library, welche die Paramiko SSH verbindung vereinfacht. Das Ziel von Netmiko ist, für verschiedene Herstellergeräte eine einheitliche Schnittstelle zu bieten und die Kommunikation zwischen Endgeräten und Server zu vereinfachen.

Netconf Das Network Configuration Protocoll ist ein Protokoll für das Netzwerk Management. Netconf wurde als RFC 6241 publiziert und bietet Mechanismen für die Installation, Manipulation und das Löschen von Konfigurationen auf Netzwerkgeräten.

2.4 Implementierte Netzwerktests

Die nachfolgende Auflistung beschreibt die Netzwerktests, die zum Zeitpunkt des Projektabschlusses implementiert sind. In erster Linie wurden Tests implementiert, die auf das gegebene Testnetzwerk ausgeführt werden konnten. Im Kapitel 3.3-Künftige Arbeiten werden weitere Netzwerktests beschrieben, die in das System integriert werden können.

Testname	Testbeschreibung
Napalm Ping	Test der einen Ping auf ein Zielgerät mit dem Napalm-Treiber durchführt.
Netmiko Ping	Test der einen Ping auf ein Zielgerät mit dem Netmiko-Treiber durchführt.
Napalm show Interfaces	Test der mit dem Napalm-Treiber die Interfaces des Hostgeräts anzeigt.
Netmiko show Interfaces	Test der mit dem Netmiko-Treiber die Interfaces des Hostgeräts anzeigt.
Netmiko Traceroute	Test der die Hops zwischen dem Hostgerät und einem Zielgerät anzeigt.
Napalm show ARP Table	Test der die ARP-Tabelle des Hostgeräts mit dem Napalm-Treiber abfragt.
Netmiko show ARP-Table	Test der die ARP-Tabelle des Hostgeräts mit dem Netmiko-Treiber abfragt.
Napalm OSPF Neighbors	Test der die OSPF Nachbarn des Hostgeräts mit dem Napalm-Treiber abfragt.
Netmiko OSPF Neighbors	Test der die OSPF Nachbarn des Hostgeräts mit dem Netmiko-Treiber abfagt.

Tabelle 1: Implementierte Tests

Man könnte nun argumentieren, dass mehrere Tests doppelt Implementiert seien, was duplizierter Code wäre. Allerdings kann man nicht für beliebige Hostbetriebssysteme die gleichen Netzwerktests verwenden. Zudem gibt es bestimmte Tests, z.B. Traceroute, die sich nicht mit Napalm implementieren lassen (zumindest nicht zum Zeitpunkt der Implementierung, ein Pull-Request für die integrierung des Napalm Traceroute steht offen). Das Programm entscheidet, basierend auf dem Betriebssystem des Hostgeräts, welcher konkrete Test instanziert werden soll. Napalm wurde dabei als der default Test für die IOS Geräte des Testnetzwerks implementiert, mit den Netmiko-Tests als Fallback und für Tests, die sich mit dem Napalm-Treiber nicht umsetzen liessen.

3 Schlussfolgerungen

3.1 Endresultat

Das Endergebnis der Studienarbeit hat leichte Abweichungen von dem, was wir zu Beginn des Projekts geplant haben. In der Architektur wurde zuerst eine weitere Strategy-Factory für die Erstellung der Netzwerk-Verbindungen geplant. Diese wurde aber nicht implementiert, da sich herausgestellt hat, dass eine Strategy-Factory nur für weitere Komplexität im Code sorgt. Stattdessen wurde ein Mapping der Hostbetriebssysteme auf die kompatiblen Verbindungsschnittstellen mittels einem Dictionary vorgenommen. Dieses Vorgehen bietet eine ähnliche Funktionalität, ist aber weitaus simpler in der Anwendung und im Verständniss.

Weiterhin wurde ein grosser Teil der Testdurchführungslogik in die konkreten Testimplementationen verschoben. Am Anfang wurde geplant, dass jede Komponente die Logik beinhaltet die für die Testdurchführung, die Evaluation der Ergebnisse und das Mapping der Resultatwerte benötigt werden. Stattdessen wurde die Funktionalität des Mappers, des Evaluators und des Testrunners soweit vereinfacht, dass darin für jeden Test die jeweilige Methode aufgerufen wird, die diese Logik spezifisch für jeden Test implementiert. Durch dieses Vorgehen konnte beispielsweise das Mapping der Testresultate für jeden Netzwerktest innerhalb des Tests vorgenommen werden. Dadurch wurde viel Code im Mapper gespart, da jeder Test eine andere Formatierung der Rückgabewerte hat und dementsprechend für jeden Test ein spezifisches Mapping der Ergebnisse hätte gemach werden müssen.

Das Nornir-Framework erlaubt es dem Benutzer, schnell und einfach Netzwerkbefehle an ein definiertes Netzwerkgerät zu senden und die Rückgabewerte werden dabei in einem Dictionary von Resultatwerten zurückgegeben. Dadurch müssen die Resultate nicht mit verwendung von regulären Ausdrücken mühsam geparst werden, sondern sie können unter Anwendung von einfachen Operationen in eine einheitliche Form gebracht und mit den Erwartungswerten verglichen werden. Der grösste Nachteil in der Verwendung von Nornir ist die Navigierung durch die Dokumentation. Will man die einzelnen Netzwerktreiber (Napalm, Netmiko, etc.) verwenden, muss man oftmals in den Dokumentationen der Treiber selber nachsehen, wie genau man die Implementation nun vornehmen muss, vorausgesetzt, es exisiert eine spezifische Anleitung für die Implementation.

3.2 Vergleich mit der Vorarbeit

Die Studienarbeit aus dem Herbstsemester 2016 hat im vergleich mit dieser Studienarbeit mehr konkrete Netzwerktests umgesetzt. Allerdings hat diese Studienarbeit einen höheren Fokus auf der Erweiterbarkeit um neue Tests und es werden mehr verschiedene Gerätehersteller unterstützt.

Die reine Pyhton-Implementation erlaubt ein Deployment auf eine vielzahl von Geräten mit unterschiedlichen Betriebssystemen unabhängig von externen Frameworks. NUTS2.0 lässt sich in ein bestehendes Configuration Mangement Tool einbinden und kann auf jedem Gerät ausgeführt werden, welches Python3 unterstützt. Im vergleich dazu wurde NUTS1.0 hauptsächlich für die Verwendung auf Linux-Systemen entwickelt.

Die Erweiterung des Systems ist in NUTS2.0 einfacher gestaltet, da sämtlicher Code in Python geschrieben ist und die verwendeten Module ebenfalls in Python entwickelt wurden.

3.3 Künftige Arbeiten

Automatische Geräteerkennung

Im aktuellen Programm müssen sämtliche Netzwerkgeräte von Hand in das Inventar aufgenommen werden, was genaue Kenntnisse des Netzwerks voraussetzt und zeitintensiv ist. Eine mögliche Erweiterung von NUTS2.0 wäre eine automatische Erkennung aller Geräte im Netzwerk und das automatische Erstellen des Inventars. Dieses Vorgehen würde dem Anwender einiges an Arbeit einsparen und den Prozess für die Erstellung von Netzwerktests beschleunigen.

Anbindung einer Datenbank

Die Anbindung einer Datenbank würde viele Vorteile mit sich bringen. Die Verwaltung des Inventars oder der Testdefinitionen müsste nicht mehr in YAML-Files gemacht werden, sondern könnte mit einer Datenbank vereinfacht werden. Das Speichern von Testresultaten in einer Datenbank würde es ermöglichen, spezifische Queries für individuelle Testresultate oder Testdurchführungen zu verwenden. Dadurch könnte man wirkungsvolle Abfragen durchführen, um beispielsweise sämtliche Tests vor einer geplanten Änderung und danach abzurufen und auszuwerten.

Komplette GUI geführte Durchführung

Die aktuelle Software hat nur ein minimalistisches GUI für die Auswahl und Reihenfolge der Testdurchführung. Eine mögliche Erweiterung ist das Einbinden eines GUI für die gesammte Testdurchführung. Ein Benutzer könnte z.B. die Testdefinitionen über ein Interface gesteuert erfassen, statt diese in einem YAML selber zu erstellen. Man könnte für sämtliche Testcommands beschreibungen anzeigen, was diese Tests machen und wie sie verwendet werden. Es könnte eine Grafische Abbildung des Netzwerks angezeigt werden (Digitaler Zwilling), in dem man sämtliche Neztwerkgeräte mit ihren Parametern anzeigt.

Erweiterung um Tests

Zum Zeitpunkt der Studienarbeit wurden nur einige Netzwerktests implementiert, die sich auf das Testnetzwerk auch ausführen liessen. Das System liesse sich um sämtliche Netzwerktests erweitern, die in allen möglichen Netzwerken auch tatsächlich ausgeführt werden könnten. Ausserdem gibt es Netzwerktests, die sich mit Nornir zum jetzigen Zeitpunkt gar nicht ausführen lassen, z.B. die Grafische darstellung des OSPF spanning Tree. Solche Tests müssten manuell implementiert werden und benötigen allenfalls andere Module als Nornir.

Mögliche Tests TODO

Asynchrone Durchführung der Tests

Momentan werden alle Netzwerktests nacheinander, d.H. Synchron durchgeführt. Dadurch benötigt das Programm für die Durchführung von umfangreichen Testdefinitionen viel Zeit. Man könnte in einer zukünftigen Erweiterung Tests mit der gleichen Kategorie, z.B. Ping-Tests asynchron (paralell) ausführen, was zu einer kürzeren Durchführungszeit führen würde.

4 Glossar

Begriffserklärung

TODO

Abbildungsverzeichnis

*

Tabellenverzeichnis

*

Literaturverzeichnis

TODO

II. Anhang

1 Projektplanung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2 Anforderungen

Im folgenden Abschnitt werden die Anforderungen an ein Netzwerk-Test-System formuliert. Es werden die Kern-Akteuere identifiziert und deren Funktion und Abhängigkeiten und Anforderungen formuliert, um auf dieser Basis die Software zu entwerfen.

2.1 Akteure in einem Netzwerksystem

In der Praxis gibt es für die verschiedenen Akteure in einem Netzwerksystem unterschiedliche Bezeichnungen. Beispielsweise ist oft nicht klar, was der Unterschied zwischen einem Netzwerk-Architekten und einem Netzwerk-Engineer ist und welche Verantwortungen diese nun genau haben. Wir haben eine eigene Unterscheidung der Akteure formuliert und diese in den kommenden Sektionen dokumentiert, um eine einheitliche Basis für die Leser zu schaffen.

Netzwerk-Architekt

Ein Netzwerk-Architekt plant und erstellt Kommunikationsnetzwerke. Im Zuge dieser Arbeit wurde zwischen dem Architekten als verantwortlichen Senior-Network-Engineer und einem Network Engineer (Junior oder Senior) als operativen Mitarbeiter unterschieden. Der Architekt nimmt dabei eher die Rolle des Managers oder Teamleiters ein. Er führt dabei normalerweise keine Konfigurationen am Netzwerk durch.

Netzwerk-Engineer

Ein Netzwerk-Engineer ist für die Installation und Instandhaltung eines Netzwerks zuständig. Er ist dem Netzwerk-Architekten unterstellt und setzt mit Ihm zusammen die geplanten Arbeiten um.

Netzwerk-Administrator

er Netzwerk Administrator hat üblicherweise eine abgeschlossene Berufslehre in der Informatik und arbeitet zusammen mit dem Netzwerk-Engineer am Netzwerk. Es wird davon ausgegangen, dass ein Netzwerk Administrator keine bis wenige Programmierkenntnisse hat. Ein Netzwerk Administrator hat, je nach Grösse des Netzwerks, nur Kenntnisse über einen Teil der Netzwerkumgebung. Er führt dabei ihm vom Architekten oder Engineer vorgegebene Arbeiten aus und muss dazu nicht den vollen Überblick über das Netzwerk und die darin verwendeten Technologien haben.

Netzwerk-User

Benutzer der Netzwerkumgebung. User können das Netzwerk verwenden, aber nicht dessen Konfigurationen anpassen.

Netzwerk-Gerät

Ein Netzwerkgerät kann aus Hardware wie Switch, Router oder Server bestehen oder Virtuell als Software implementiert sein. Im Zuge der Arbeit werden Netzwerkgeräte auch als Netzwerk-Devices oder einfach Device bezeichnet. Typischerweise haben Devices eine Statische Konfiguration und einen dynamischen Zustand zur Laufzeit. In den kommenden Kapiteln wird genauer auf Netzwerkgeräte eingegangen.

Netzwerk-Verbindung

Die Netzwerkverbindung ist der Kommunikationskanal zwischen den einzelnen Netzwerkgeräten. Sie kann in physischer Form als Kabel, oder mit kabellosen Mitteln z.B. Funk umgesetzt sein. Die Wahl des Übertragungsmediums hat grossen Einfluss über die verfügbare Bandbreite und mögliche Störfaktoren.

Repository/Inventar

Im Inventar werden die Unterschiedlichen Devices mit den für den Betrieb wichtigsten Parametern gespeichert. Das Inventar kann in digitaler Form als Repository, als File auf einem Ordner/Computer, oder analogin einem Dokumentenorder abgelegt sein. Das Inventar wird benötigt, um die aktuellen Konfigurationen, die physische Position des Geräts oder sonstige für den Betrieb relevanten Informationen zu dokumentieren.

2.2 Akteure in der zu entwickelnden Software

Testprogramm

Das Testprogramm ist der Kern des zu entwickelnden Systems dieser Arbeit. Es Interagiert mit den anderen Akteuren und hat, vom Akteur und Kontext abhängig, unterschiedliche Anforderungen. Es soll so aufgebaut sein, dass ein Benutzer der Software diese mit möglichst geringem Aufwand bedienen kann.

Testdefinitionssprache

Wird im Rahmen dieser Arbeit auch als Testbeschreibungssprache oder Definitionssprache bezeichnet. Eine Testdefinition beschreibt die einzelnen Testfälle, die von einem System durchgeführt werden sollen. Die Definitionssprache soll dabei in einem Format gehalten werden, das von allen Benutzern der Software verstanden wird und von diesen erweitert werden kann.

Testreport

Ein Testreport soll, möglichst einfach und genau, die Ergebnisse eines Netzwerktests aufzeigen. Fehlgeschlagene Tests sollen dabei möglichst einfach und schnell zu erkennen sein und alle Informationen beinhalten, die ein Betrachter benötigt, um den Fehler im System zu lokalisieren und beheben. Ausserdem muss mindestens noch ein Zeitstempel vorhanden sein, um die Historie vergangener Netzwerktests nachvolziehen zu können. Testreporte können auf dem System, welches die Tests ausführt, oder in einem zentralen Repository abgelegt werden, damit mehrere Mitglieder eines Netzwerkteams gleichzeitig darauf zugreifen können.

Kommunikationskanal

Der Kommunikationskanal, nicht zu verwechseln mit der Netzwerkverbindung zwischen zwei Netzwerkgeräten, verbindet ein zu testendes Netzwerk mit dem Testprogramm. Möglichkeiten für einen Kanal sind beispielsweise das SSH (secure shell) Protokoll oder der Restconf Standard. Dies kann über eine Kabelverbindung oder Kabellos geschehen. Die Wahl des Kommunikationskanals beeinflusst dabei, in welcher Form Netzwerktests durchgeführt werden können und in welchem Format die Ergebnisse zurückgegeben werden.

Netzwerktest

Werden heute meist manuell oder mit Hilfe eines Skripts durchgeführt. Ein automatisierter Netzwerktest sollte hypothetisch ad-hoc nach jeder Konfigurationsänderung vom Testprogramm durchgeführt werden um die Funktionsweise des Netzwerks zu validieren. Ein Benutzer des zu entwickelnden Systems soll in der Lage sein, mit nur geringer Einarbeitungszeit, Netzwerktests zu spezifizieren und durchzuführen.

2.3 Use Cases

2.3.1 Use Case Diagramm

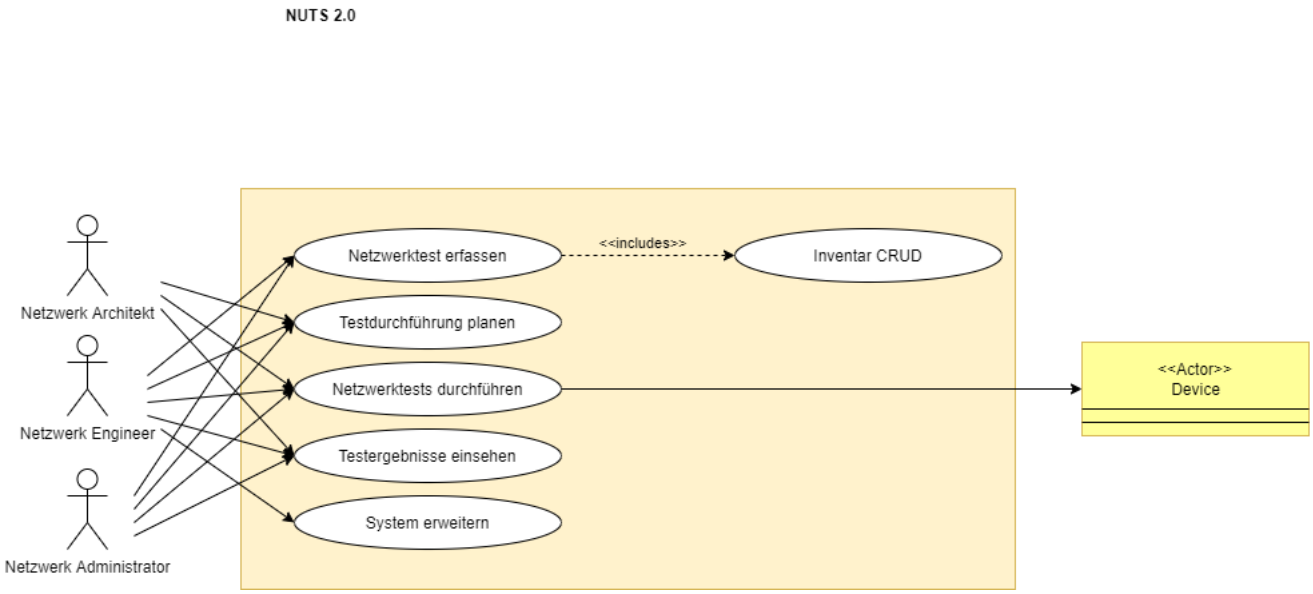

Abbildung 1: Use Case Diagramm

2.3.2 Aktoren

Die Primären Akteure sind der Netzwerk Architekt, -Administrator und -Engineer. Der Architekt will primär die Ergebnisse einsehen können, um zu sehen, dass das Netzwerk korrekt funktioniert. Ausserdem möchte er, beispielsweise um eine Erweiterung des Netzwerks zu Planen, eine Ausführung von Netzwerktests konfigurieren und durchführen. Der Administrator will Netzwerktests erfassen, deren Durchführung planen, die Tests durchführen und die Ergebnisse einsehen. Der Engineer möchte neben den Tätigkeiten, die der Administrator ausführt, zudem das Testsystem um weitere Netzwerktests erweitern können.

2.4 Beschreibung Usecases (Brief)

Netzwerktest erfassen Ein Netzwerktest setzt sich zusammen aus der Testdefinition mit den zu testenden Devices, Befehle, die auf den Devicesv ausgeführt werden sollen, einem oder mehreren Kommunikationskanälen, über den die Devices angesprochen werden und einem Erwartungswert für das Ergebnis. Diese Informationen werden in einer Testdefinitionssprache gespeichert, welche so strukturiert sein muss, dass sie ein Softwaresystem einfach laden kann und trotzdem von Menschen interpretiert werden kann. Die Erfassung eines Netzwerktests soll so einfach wie möglich gehalten werden, damit Administratoren und Engineers effizient neue Tests spezifizieren können.

Inventar CRUD Der Netzwerk Engineer oder -Administrator möchte die physischen und virtuellen Netzwerkgeräte und deren statische Konfiguration in einem Inventar verwalten. Das Inventar wird von dem zu entwickelnden System verwendet, um die Gerätekonfigurationen wie z.B. Zugangsdaten oder Herstellerinformationen abzurufen. Es soll möglich sein, das Inventar automatisiert zu erstellen und Geräte in distinkte Gruppen zu kategorisieren.

Testdurchführung planen Themen, die in der Testausführung relevant sind, sind die Auswahl, welche Tests überhaupt ausgeführt werden sollen, die Reihenfolge der Tests, auf welchen Teil des Systems sie angewandt werden sollen und ob sie synchron oder asynchron durchgeführt werden. Weitere Punkte wären das automatische durchführen von Tests zu spezifischen Zeiten oder Wochentagen und dass die Testkonfiguration gespeichert wird, um sie später anzupassen. Auch hier ist darauf zu achten, dass das zu entwickelnde System so aufgebaut ist, dass Engineers, Architekten und Administratoren effizient arbeiten können.

Netzwerktests durchführen Der Anwender möchte die in der Testdurchführung geplanten Netwerktests auf das angegebene System ausführen. Dazu wird in einer Benutzeroberfläche die Testausführung gestartet oder auf einem Gerät automatisch die zu entwickelnde Software ausgeführt.

Testergebnisse einsehen Nachdem ein Test durchgeführt wurde, soll ein Testresultat angezeigt werden. In den Resultaten soll ersichtlich sein, welche Tests durchgeführt wurde, was der Test genau gemacht hat, welche Befehle auf welchen Devices ausgeführt wurde und wie das Ergebnis ist. Die Testergebnisse sollen auf der Konsole/Benutzeroberfläche ersichtlich sein und zusätzlich in einem Testreport mit Datum und Uhrzeit gespeichert werden, damit die Historie des Netzwerks ermittelt werden kann. Wenn Tests durch einen Fehler im zu entwickelnden System nicht durchgeführt werden kann, sollen alle anderen Tests nicht davon beeinflusst werden und das Ergebnis soll einen Vermerk für das Versagen des Systems beinhalten, mit dem der Anwender in der lage ist, die Ursache zu ermitteln und zu beheben.

System erweitern Engineers sollen in der Lage sein, das System bei Bedarf zu erweitern, z.B. um weitere Tests oder Netzwerkschnittstellen hinzuzufügen oder Fehler zu verbessern. Die Erweiterungen beschränken sich aber auf rein funktionale Bereiche des Systems. Für Änderungen an der Benutzeroberfläche sollen Softwareengineers mit Erfahrung auf dem Gebiet hinzugezogen werden.

2.5 Nichtfunktionale Anforderungen

In diesem Kapitel werden die nichtfunktionalen Anforderungen an das Projekt behandelt. Es werden Aspekte und Anforderungen aus den Bereichen Änderbarkeit, Benutzbarkeit, Effizienz, Zuverlässigkeit, Betreibbarkeit und Sicherheit gemäss ISO/IEC 9126 betrachtet. Die jeweiligen Aspekte werden in ihren Unterkapiteln genauer beschrieben. Es wurden mögliche Szenarien erarbeitet, die in der Erstellung oder dem Betrieb der Software auftreten können und beim Architekturdesign in betracht gezogen wurden.

2.5.1 Änderbarkeit

Aufwand, der zur Durchführung von vorgegebenen Änderungsarbeiten benötigt wird. Unter Änderungen gehen Korrekturen, Anpassungen oder Veränderungen der Umgebung, Anforderungen oder funktionalen Spezifikation. Gemäss ISO 9126 gehören zur Änderbarkeit folgende Teilmerkmale:

Analysierbarkeit Aufwand, der benötigt wird, um das System zu verstehen, z.B. um Ursachen von Versagen oder Mängel zu diagnostizieren oder Änderungen zu planen.

Modifizierbarkeit Wie leicht lässt sich das System anpassen, um Verbesserungen oder Fehlerbeseitigungen durchzuführen.

Stabilität Wahrscheinlichkeit, dass mit Änderungen unerwartete Nebenwirkungen auftreten.

Testbarkeit Wie gross wird der Aufwand, bei Änderungen die Software zu prüfen.

Szenario: Neue Netzwerkschnittstelle Wenn zum bestehenden System eine neue Netzwerkschnittstelle definiert werden soll, so muss die dafür notwendige Software innerhalb von einer Arbeitswoche entwickelt, integriert und in Betrieg genommen werden können.

Qualitätsziele	Flexibilität, Erweiterbarkeit, Anpassbarkeit, Austauschbarkeit
Geschäftsziel(e)	Software kann mit geringem Aufwand an geänderte Anforderungen angepasst werden
Auslöser	Ein Engineer möchte weitere Tests einbinden oder Schnittstellen, die nicht im System integriert sind.
Reaktion	Die Software lässt sich von einem Entwickler in weniger als einer Woche um benötigte Komponenten erweitern.
Zielwert	Erweiterungen der Netzwerkschnittstellen oder Anpassungen von Tests sind innerhalb von 40 Personenstunden umsetzbar.

Tabelle 2: Szenario: Neue Schnittstelle

Scenario: Schnelle Fehlerlokalisierung Die Ursache von fehlgeschlagenen Tests (Software-Unittests) lässt sich in kurzer Zeit lokalisieren.

Qualitätsziele	Schnelle Fehlerbehebung, Änderbarkeit, Anpassbarkeit, geringes Risiko bei Erweiterungen
Geschäftsziel(e)	Entwickler können das Programm einfach anpassen und erkennen im Fehlerfall schnell, was nicht funktioniert hat.
Auslöser	Eine Änderung im Code führt zu Fehlnern in der Ausführung.
Reaktion	Wenn ein Fehler dazu führt, dass die Softwareausführung fehlschlägt, kann ein Entwickler aufgrund von Fehler- und/oder Log-Nachrichten die Ursache in kurzer Zeit lokalisieren.
Zielwert	Fehlerlokalisierung findet durchschnittlich in weniger als 10 Minuten statt.

Tabelle 3: Szenario: Schnelle Fehlerlokalisierung

2.5.2 Benutzbarkeit

Zeitlicher Aufwand, der für die Erlernung der Benutzung des Programms benötigt wird. Die User werden hierfür in spezifische Nutzergruppen mit festgelegten Fähigkeiten unterteilt.

Verständlichkeit Aufwand für den Nutzer, die Konzepte und Menüführung der Anwendung zu verstehen.

Erlernbarkeit Aufwand für den User, sich ohne Vorwissen in das System einzuarbeiten.

Bedienbarkeit Aufwand für den Benutzer, die Anwendung zu bedienen.

Szenario: Einfachheit der Definitionssprache Die Definitionen von Inventar und Tests sind so aufgebaut, dass ein User in kurzer Zeit die Struktur und den Aufbau versteht und eigene Tests implementieren kann.

Qualitätsziele	Produktivität, Einfachheit, Verständlichkeit
Geschäftsziel(e)	Einarbeitung in die Testdefinition erfolg möglichst einfach und benötigt nur geringes Vorwissen.
Auslöser	Ein Nutzer, welcher keine Erfahrung im Umgang mit der Software hat, möchte eigene Tests definieren.
Reaktion	Benutzer können sich schnell in die Testdefinitionen einlesen und rasch eigene Tests definieren, vorausgesetzt, sie haben Kenntnisse des Netzwerkes.
Zielwert	Ungeschulte Nutzer verstehen innerhalb von durchschnittlich 30 Minuten die Struktur und den Aufbau der Testdefinitionen und sind in der Lage, eigene Tests zu erstellen.

Tabelle 4: Szenario: Einfachheit der Definitionssprache

Szenario: Hinweis auf Fehleingaben Fehlerhafte Eingaben werden vom System ignoriert und der Benutzer wird auf die falsche Eingabe hingewiesen. Das Programm führt fehlerfreie Programmteile unabhängig von den Fehlern durch.

Qualitätsziele	Robustheit, Verständlichkeit, Fehlertoleranz.
Geschäftsziel(e)	Fehleingaben führen nicht dazu, dass die Tests nicht mehr durchgeführt werden können.
Auslöser	Ein Benutzer macht einen Fehler bei der Testdefinition und startet das Programm.
Reaktion	Das Programm führt alle korrekten Tests durch und informiert den Benutzer, dass es fehlerhafte Tests gibt, die nicht durchgeführt werden können. Die Hinweise werden im Report und auf der Konsolenausgabe geschrieben.
Zielwert	Tests sind einzeln gekapselt und werden unabhängig voneinander durchgeführt. Falscheingaben werden vom Programm detektiert und im Testreport sowie auf der Konsolenausgabe erwähnt.

Tabelle 5: Szenario: Hinweis auf Fehleingaben

2.5.3 Effizienz

Mit Effizienz ist die 'performance efficiency' gemeint, d.h. das Verhältnis zwischen dem Leistungsniveau der Software und den eingesetzten Hardwarekomponenten. Andere Beschreibungen umfassen: Skalierbarkeit, Speicherbedarf, Verarbeitungsgeschwindigkeit, Antwortzeit etc. Teilmerkmale nach ISO 9126:

Zeitverhalten Dauer für Verarbeitung und Antwortzeit sowie Durchsatz bei der Ausführung des Programms

Verbrauchsverhalten Wie viel Speicherbedarf hat das Programm, wie lange werden Betriebsmittel in Anspruch genommen und welche Hardwarekomponenten werden benötigt.

2.5.4 Zuverlässigkeit

Unter Zuverlässigkeit versteht man die Fähigkeit der Software, unter festgelegten Bedingungen die Funktionalität über einen definierten Zeitraum zu gewährleisten

Reife Geringe Ausfallhäufigkeit durch Fehlzustände.

Fehlertoleranz Die Software ist in der Lage, trotz Fehlern ihr spezifiziertes Leistungsniveau beizubehalten.

Wiederherstellbarkeit Im Fehlerfall können betroffene Daten wiederhergestellt und die Funktionalität wieder aufgenommen werden.

Szenario: Tests lassen sich auf der Netzwerkseite nicht ausführen Falls ein Test auf dem jeweiligen Netzwerkgerät nicht erfolgreich durchgeführt werden kann, läuft das Programm weiter und definiert den dazugehörigen Netzwerktest als nicht bestanden.

Qualitätsziele	Robustheit, Behandlung Infrastrukturbedingter Fehler.
Geschäftsziel(e)	Das System führt alle Tests unabhängig voneinander durch. Wenn ein Test zu einem Fehler führt, weil z.B. ein falsches Netzwerkgerät angegeben wurde, wird dieser Test unabhängig von allen anderen Tests fehlschlagen.
Auslöser	Test lässt sich auf spezifizierter Infrastruktur nicht ausführen.
Reaktion	Test schlägt fehl und mögliche Ursachen werden im Report und in der Konsole angezeigt. Alle anderen Tests laufen durch.
Zielwert	Das Fehlschlagen eines Tests fürht nicht zum Programmabbruch.

Tabelle 6: Szenario: Testausführung Netzwerkseitig nicht ausführbar

2.5.5 Betreibbarkeit

Die Betriebbarkeit wird in der ISO 9126 nicht definiert. Die ISO spezifiziert aber mehrere Teilmerkmale, die unter dem Begriff Betreibbarkeit zusammengefasst werden können:

Analysierbarkeit Aufwand, der benötigt wird, um den Code zu analysieren, um im falle eines Versagens dessen Ursachen zu diagnostizieren oder um Änderungen zu planen und durchzuführen.

Installierbarkeit Aufwand, das Programm auf einem frisch aufgesetzten Gerät laufen zu lassen.

Übertragbarkeit Kann die Software von einer Umgebung auf eine andere übertragen werden. Als Umgebung zählen Hardwarekomponenten, Softwarekomponenten, Organisatorische Umgebungen oder Betriebssysteme.

Austauschbarkeit Aufwand und Möglichkeit, die Software anstelle einer anderen in deren spezifizierten Umgebung laufen zu lassen.

Koexistenz Fähigkeit der Software, neben anderen Programmen mit ähnlichen oder übereinstimmenden Funktionen zu arbeiten.

Szenario: Einfache Installation auf einem neuen Gerät Das Programm lässt sich auf einem neuen Gerät ohne grossen Mehraufwand installieren, ohne dass die Funktionalität des Geräts beeinflusst wird.

Qualitätsziele	Einfachheit, Portierbarkeit, Benutzbarkeit
Geschäftsziel(e)	Die Installation der Software ist so einfach, dass sie innert kurzer Zeit und/oder automatisiert durchgeführt werden kann.
Auslöser	Die Testsoftware soll auf einem frisch aufgesetzten Gerät installiert werden.
Reaktion	Installationszeiten sind gering, benötigen wenige bis keine weiteren Softwarekomponenten oder lässt sich mit einigen Kommandozeilenbefehlen automatisch installieren.
Zielwert	Die Software wird mit einer Installationsanleitung ausgeliefert, die einfach und verständlich die Inbetriebnahme des Programms erklärt. Abhängigkeiten zu anderen Softwarekomponenten werden bewusst gering gehalten um eine einfache Installation mit weniger als 30 Minuten Zeitaufwand zu gewährleisten.

Tabelle 7: Szenario: Einfache Installation auf einem anderen Gerät

2.5.6 Sicherheit

In dieser Sektion werden Sicherheitsanforderungen beschrieben. Verschlüsselung, Privacy und der Umgang mit Passwörtern.

Verschlüsselung von Datenübertragungen Die Netzwerktest werden über eine Verschlüsselte Verbindung durchgeführt, die dem aktuellen Stand der Technik entspricht.

Umgang mit Passwörtern Zugangsdaten der Devices werden im Inventar in Unverschlüsselter Form abgelegt. Es liegt in der Verantortung der Betreiber des Netzwerks, dass die Zugangsdaten nicht von dritten eingesehen werden.

3 Design

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

4 Realisierung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Paramiko-Treiber Paramiko unterstützt sämtliche Geräte, zu denen eine SSH Verbindung aufgebaut werden kann. Es werden in der Dokumentation von Paramiko keine Spezifischen Hersteller oder Betriebssysteme genannt, die nicht von Paramiko unterstützt werden.

Netmiko-Treiber Die Geräteunterstützung von Netmiko lässt sich in drei Kategorien unterteilen: Regelmässig getestet, Limitiert getestet und Experimentell.

Bei regelmässig getesteten Geräten wird die komplette Test-Suite vor dem aktuellen Release von Netmiko durchgeführt. Die unterstützten Betriebssysteme sind:

- Arista vEOS
- Cisco ASA
- Cisco IOS
- Cisco IOS-XE
- Cisco IOS-XR
- Cisco NX-OS
- Cisco SG300
- HP ProCurve
- Juniper Junos
- Linux

Bei limitiert getestete Geräten werden die show- und config-Befehle getestet. Die unterstützten Betriebssysteme sind:

- Alcatel AOS6/AOS8
- Apresia Systems AEOS
- Calix B6
- Cisco AireOS (Wireless LAN Controllers)
- CloudGenix ION
- Dell OS9 (Force10)
- Dell OS10
- Dell PowerConnect
- Extreme ERS (Avaya)
- Extreme VSP (Avaya)
- Extreme VDX (Brocade)
- Extreme MLX/NetIron (Brocade/Foundry)
- HPE Comware7
- Huawei
- Huawei OLT
- Huawei SmartAX
- IP Infusion OcNOS
- Juniper ScreenOS
- Mellanox
- MikroTik RouterOS
- MikroTik SwitchOS
- NetApp cDOT
- Nokia/Alcatel SR OS
- OneAccess
- Palo Alto PAN-OS

- Pluribus
- Ruckus ICX/FastIron
- Ruijie Networks
- Ubiquiti EdgeSwitch
- Vyatta VyOS

Experimentelle Geräteunterstützung bedeutet, dass für diese spezifischen Geräte keine eigenen Unittests existieren und die Unterstützung somit nicht getestet ist.

Die aktuelle Netmiko-Dokumentation kann auf https://github.com/ktbyers/netmiko angesehen werden.

Napalm-Treiber Generell werden folgende Betriebssysteme von Napalm unterstützt:

- EOS
- Junos
- IOS-XR
- NX-OS
- NX-OS SSH
- IOS

5 Zeitauswertung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

6 Be(nuts)eranleitung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

7 Persönliche Berichte

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8 Eigenständigkeitserklärung

Die Authoren erklären hiermit,

- dass die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt wurde, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit den Betreuerm vereinbart wurde.
- dass sämtliche verwendeten Quellen erwähnt und gemäss den gängigen wissenschaftlichen Zitierregeln korrekt angegeben wurden.
- dass keine durch Uhrheberrecht geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter weise verwendet wurden.

Ort, Datum:	Ort, Datum:
Name, Unterschrift:	Name, Unterschrift:

9 Vereinbarung Urheber-/ Nutzungsrechte

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit NUTS2.0 von Janik Schlatter und Mike Schmid unter der betreuung von Beat Stettler und Urs Baumann geregelt

2. Urheberrecht

Die Urheberrechte stehen den Authoren zu.

3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von den Authoren, von der OST (ehemals HSR), sowie vom INS Institut for Networked Solutions nach Abschluss der Arbeit verwendet und weiter entwickelt werden.

4. Softwarelizenz

Die in der Arbeit entstandene Software untersteht der Apache Licence 2.0. Somit darf die Software frei verwendet, verändert und verteilt werden, insofern dies nicht die in der Lizenz genannten Regelungen verletzt.

Rapperswil, den..... .....
Die Studentin/ der Student

Rapperswil, den..... .....
Die Studentin/ der Student

Rapperswil, den..... .....
Der Betreuer/ die Betreuerin der Studienarbeit

Rapperswil, den..... .....
Der Betreuer/ die Betreuerin der Studienarbeit