HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Digital Twin

STUDIENARBEIT

ABTEILUNG INFORMATIK
HOCHSCHULE FÜR TECHNIK RAPPERSWIL

Herbstsemester 2019

Autoren: Dario Caluzi, Patrick Lehmann

Betreuer: Beat Stettler, Marc Sommerhalder

20. Dezember 2019

Abstract

Aufgabenstellung

Durch die zunehmende Komplexität und Grösse der Netzwerke ist es für Netzwerk-Ingenieure schwieriger auftretende Fehler im Netzwerk zu finden und zu beheben. Hilfreich könnte ein digitales Abbild, ein sogenannter "Digital Twin", des Netzwerks sein, welches den einwandfreien Soll Zustand mit korrekter Konfiguration abbildet und in einer Datenbank abgespeichert wird. Die Informationen über die Komponenten des Netzwerk und die verwendeten Protokolle sollen nach OSI Layer strukturiert abgebildet und miteinander verknüpft werden. Dadurch kann der reale Zustand jeweils mit dem Abbild verglichen werden und bei allfälligen Problemen zu einer schnelleren Eingrenzung führen.

Vorgehen / Technologien

Zu Beginn der Arbeit wurde eingehend die Graphentheorie studiert. Die daraus gewonnen Erkenntnisse wurden auf typische Netzwerk Problematiken angewendet. Anschliessend wurde eine Schema für die benötigten Daten auf Basis der ersten drei Schichten des OSI-Modell erstellt. Diese Abstrahierung wurde anhand von einem realen Netzwerks als Webapplikation mit Backend realisiert.

Für die Entwicklung des Frontends wurde das JavaScript-Framework Vue.js mit der Webkomponente 3D Force Graph, welches für das Zeichnen des Graphen verantwortlich ist, verwendet. Im Backend bildet die in Python geschriebene Flask-API die zentrale Komponente des Digital Twin für die Datenabfrage des Frontends. Die Daten des Digital Twin werden über die RESTCONF-Schnittstelle der Netzwerkgeräte abgefragt, im Backend verarbeitet und in einer Neo4j Graphendatenbank abgespeichert. Die dazugehörigen Zugangsdaten der Netzwerkgeräte sind in der NoSQL-Datenbank MongoDB abgelegt.

Ergebnis

Das Ergebnis ist eine Webanwendung, welche Daten der ersten drei OSI-Modells von Cisco Netzwerkgeräten abfragen und als Graph repräsentiert. Zusätzlich kann der Digital Twin mit neuen Nodes und Connections erweitert werden. Zudem kann die Konnektivität von zwei Nodes überprüft und den kompletten Pfad der Verbindung angezeigt werden. Die Architektur ist so aufgebaut, dass weitere OSI-Layer sowie weitere Netzwerkgeräte Schnittstellen einfach erweitert werden können. Zusätzlich können weitere Graph Operationen auf dem Digital Twin zur Analyse des Netzwerks hinzugefügt werden.

Inhaltsverzeichnis

Aufgabenstellung

Aufgabenstellung Digital Twin

Netzwerke wurden in den letzten Jahren immer grösser und neue Protokolle auf verschiedenen Layern haben die Komplexität des Gesamtsystems massiv erhöht. So ist denn auch für einen Netzwerk- Ingenieur zunehmend schwierig, auftretende Fehler im Netzwerk zu finden und zu beheben. Eine grosse Hilfe könnte ein digitaler Zwilling des Soll Zustandes sein, welcher das Netz in einwandfreiem Zustand und mit korrekter Konfiguration abbildet und in einer DB abspeichert. Informationen über Komponenten, über Verbindungen zwischen Komponenten und über die Zustände der eingesetzten Protokolle müssen nach OSI Layer strukturiert abgebildet und miteinander verknüpft werden. Dazu kommen noch aufbauende Konzepte wie Segmentierung, Overlays, Firewall Regeln usw.

Der Vorteil eines solchen Digitalen Zwillings sind vielfältig:

- Der reale Zustand eines Netzes kann laufend mit dem Digitalen Zwilling verglichen werden. Treten Fehler auf, können diese durch kurzen Vergleich mit dem "Soll Zustand" im Digitalen Zwilling rasch eingegrenzt werden.
- Werden später Änderungen am Netzwerk gewünscht, können diese zuerst im Digitalen Zwilling vorgenommen und getestet werden, bevor sie auf das produktive Netz geschalten werden.

In einer Vorgängerarbeit wurden bereits die OSI Layer 1 und 2 abstrahiert. In dieser Arbeit geht es nun darum, höhere Layer hinzuzufügen. Dabei können je nach Vorliebe der Studierenden:

- Netzwerk-Segmentierungen (VLANs, VRF, VPN usw.) sowie die dazugehörenden Sicherheitsregeln (Firewall, Policies, QoS) etc. abstrahiert werden
- der neue Ansatz von Campus Fabrics mit Overlays abstrahiert werden

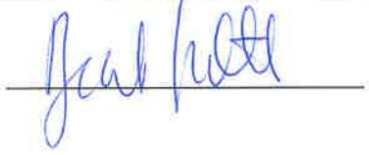
Neben diesem theoretischen Teil soll der so entwickelte Ansatz danach in Software umgesetzt und an einem konkreten Netzwerk überprüft werden. Dabei kann der Prototyp aus der ersten Arbeit übernommen werden (sofern hilfreich).

Voraussetzung:

- Freude an Netzwerktechnologien und Protokollen.
- Gute Fähigkeit, komplexe Zusammenhänge zu abstrahieren

Ort, Datum: 18.12.2019

Unterschrift Betreuer:



Management Summary

Ausgangslage

Obwohl sich die Netzwerktechnologien stetig weiterentwickeln, basieren grosse Teile des Internets und firmenbetriebene Netzwerke nach wie vor auf Technologien, welche in den 80er Jahren entwickelt wurden. Wenn in Betracht gezogen wird, dass immer mehr Systeme den Zugang zum Netzwerk respektive Internet voraussetzen, um ihr volles Potenzial auszuüben, ist es wenig erstaunlich, dass Netzwerke immer grösser und umfangreicher werden. All diese Faktoren zusammengenommen, ist es nachvollzierbar, dass ein Netzwerk stetig an Komplexität zunimmt.

Um in einer Firma ein Netzwerk möglichst ohne Probleme zu betreiben und im Fall eines Ausfalls effizientes Troubleshooting durchzuführen, sind sehr gut ausgebildete Netzwerkadministratoren gefordert. Da ein Netzwerk unterbruchsfrei funktionieren muss, haben auch die Netzwerkadministratoren keine gute Möglichkeit geplante Änderungen vorgängig im Netzwerk zu testen, ohne ein sehr kostenintensives Testnetzwerk mit Hardwaregeräten aufzubauen.

In diesem Fall bräuchte man einen Digital Twin, welcher ein exaktes Abbild des produktiven Netzwerks ist. Der Digital Twin erlaubt es den Netzwerkadministratoren, ein Netzwerk präventiv auf Schwachstellen und Fehler zu überprüfen, sowie manuell angedachte Netzwerkerweiterung digital zu testen, ohne dass Kosten für zusätzliche Hardware anfallen oder ein Netzwerkunterbruch entsteht.

Vorgehen / Technologien

Um diesen Digital Twin zu realisieren, muss das Netzwerk abstrahiert werden. Damit dies optimal gelöst werden kann, werden Studien zu Multilayer Netzwerken mit Graphentheory in Betracht gezogen. In unserem Fall werden verschiedene Netzwerkkomponente durch Graphen Elemente repräsentiert und den Netzwerkfunktionalitäten entsprechende Graphenopreationen zugewiesen.

Ergebnisse

Als Ergebnis der Arbeit wurde eine Architektur erstellt, welche das gewünschte Mapping in die Graphentheorie für die Basisfunktionalitäten des Netzwerks ermöglicht. Damit diese Architektur einfach zugänglich ist, wurde eine klassische Webanwendung, in Kombination mit einem Backend, erstellt. Aus Sicherheitsgründen sind diese Teile voneinander getrennt. Denn während die Webanwendung die Interaktion mit dem System vereinfacht, soll das Backend System komplett von der Aussenwelt getrennt sein. Es braucht jedoch Zugang zu den Netzwerkgeräten, um die Informationen über das Netzwerk abzufragen.

Im Digital Twin können Zugangsdaten zu gewünschten Netzwerkgeräten einfach verwaltet werden. Anschliessend werden diese genutzt, um ein Abbild des Netzwerks zu erstellen. Zur Analyse gibt es eine graphische Darstellung des Netzwerks, welche dynamisch erstellt wird, und anschliessend nach Benutzerwunsch auf Problemdomänen eingegrenzt werden kann. Damit eine Verbindung im Netzwerk

getestet werden kann, gibt es eine Ansicht, wo frei wählbare Endpunkte auf ihre Konnektivität überprüft werden können. Erweiterungen im Netzwerk können so getestet werden, indem manuell über die Weboberfläche neue Endpunkte oder Verbindungen zum Digital Twin hinzugefügt werden.

Ausblick

Die Architektur des Digital Twin ist so aufgebaut, dass die Software einfach auf gewünschte Funktionalität erweitert werden kann. Dadurch können sowohl Schnittstellen für unterstützte Netzwerkgeräte ausgeweitet, wie auch neue Technologien implementiert werden. Der Digital Twin ist demnach mit neuen Netzwerktechnologien erweiterbar und wird nicht durch Neuerungen im Netzwerkbereich überflüssig.

1 Technischer Bericht

1.1 Ausgangslage

In Arbeiten wie beispielsweise Multilayer Networks [1] von Mikko Kivelä wurde gezeigt, wie es möglich ist, diverse Netzwerke durch Graphen zu repräsentieren. Auch ein Computer Netzwerk kann daher durch einen Graphen dargestellt werden. In dieser Studienarbeit wollen wir eine geeignete Repräsentation eines Graphen für ein solches Netzwerk finden, bei welcher auch Algorithmen der Graphentheorie eingesetzt werden können, um Schwachstellen und weiteres in dem gegebenen Netzwerk zu identifizieren.

Diese Analyse bildet die Grundlage für ein Tool, welches Netzwerkingenieure dabei unterstützt, ein Netzwerk zu verwalten.

Diese Applikation wird anhand der definierten Zuordnung von Graphentheorie auf das Netzwerk gebaut, welche den Zustand eines gegebenen Netzwerks in einem Graphen abspeichert. Diese Kopie des Netzwerks wird von uns als Digital Twin bezeichnet. Des Weiteren soll die Applikation ein Interface bieten, welches einem Netzwerkingenieur die Möglichkeit gibt, den Digitalen Twin manuell zu erweitern, um zukünftige Änderungen am Netzwerk zu testen, ohne den produktiven Betrieb zu beeinträchtigen. Auch vordefinierte Funktionen zur Fehleranalyse des Netzwerks sollen geboten werden.

1.2 Problembeschreibung

1.2.1 Motivation

Netzwerke wurden in den letzten Jahren immer grösser und durch das Einsetzen von neuen Protokollen auf unterschiedlichen OSI Layern zunehmend komplexer. Aus diesen Gründen ist es auch für gut ausgebildete Fachleute wie NetzwerkingenieurInnen schwieriger geworden Schwachstellen sowie auftretende Fehler zu identifizieren und zu beheben.

Ein Netzwerk ist absolut kritisch für ein funktionierendes Unternehmen und jeder Ausfall hat schwerwiegende Konsequenzen zur Folge. Dies bedeutet, dass am Netzwerk nicht experimentiert werden darf. Es dürfen nur Änderungen vorgenommen werden, bei welchen man sicher ist, dass sie nicht zu Problemen führen.

Durch den Einsatz eines Digital Twin ergibt sich die Möglichkeit, dass ein Netzwerk mit sehr wenig Aufwand und Kosten auf Schwachstellen und Fehler überprüft sowie zu Testzwecken erweitert werden kann.

1.2.2 Bestehende Lösungen

Es gibt bereits bestehende Lösungen, welche es NetzwerkingenieurInnen erleichtert, ein Netzwerk zu verwalten. Darunter sind beispielsweise Tools wie Network Performance Monitor von SolarWinds [2] und PRTG Network Monitor [3]. Diese Tools sind darauf spezialisiert, um das Netzwerk zu monitoren und das operationelle Betreiben des Netzwerkes sicherzustellen. Damit vorgängig eine Änderung an einem Netzwerk getestet werden kann, muss trotzdem noch ein separates exakt kopiertes Testnetzwerk aufgebaut werden, was ein enormer Kosten- und Zeitaufwand mit sich bringt.

1.3 Lösungskonzept

Der Digital Twin ist eine klassische 3 Tier Applikation, bestehend aus Frontend, Backend und Persistenz Layer.

1.3.1 Frontend

Im Frontend haben wir mit HTML und Vue [4] die Darstellung realisiert, und für die Logik und Verbindung zum Backend mit JavaScript gearbeitet. Damit der Graph aus der Datenbank schön dargestellt werden kann und man auch als User intuitiv im Graph navigieren kann, beispielsweise durch Drehen und Verschieben des Graphen sowie das Hineinzoomen und Beschriften der Elemente haben wir die Library 3d Force Graph [5] verwendet. Die Library bietet eine gute Implementation und unterstützt für die Daten des Graphen das weit verbreitete Format JSON. Das Verhalten des Graphen sowie die stilistischen Attribute wie Farben etc. konnten von uns beliebig angepasst werden.

1.3.2 Backend

Das Backend besteht aus einer API, welche in Python geschrieben ist, und die Library Flask [6] verwendet, um die API zur Verfügung zu stellen. Über die API können Funktionen aufgerufen werden, wie das Erstellen des Digital Twins, welches die Netzwerkgeräte direkt abfragt. Dazu muss das Backend im selben Netzwerk sein wie die Netzwerkkomponenten, welche man wünscht, im Digital Twin abzubilden. Auch die Aufbereitung der Daten für das Darstellen des Graphen im Frontend wird im Backend gemacht und anschliessend über die API ans Frontend geschickt. Zusätzlich ist das Backend auch das Verbindungsglied zur Persistenz.

1.3.3 Persistenz

Für die Persistenz brauchen wir zwei unterschiedliche Datenbanken. Einerseits setzen wir eine MongoDB [7] ein, um die Zugänge zu den Netzwerkgeräten zu verwalten, welche man in den Digital Twin integrieren möchte. Um mit der MongoDB kommunizieren zu können, verwenden wir im Backend die Library pymongo [8]. Um den Digital Twin zu sichern, benutzen wir eine neo4j Datenbank [9], welche das Netzwerk als Graph abgespeichert hat. Manuell ergänzte Geräte und Verbindungen im Digital Twin werden direkt in der Datenbank hinzugefügt. Im Backend wird die neo4j Datenbank über den Neo4j Bolt Driver für Python [10] angesteuert, welcher es erlaubt im Backend Cypher Querys auf die Datenbank abzusetzen.

1.3.4 Begründung

Diese Architektur hat es uns ermöglicht, die Aufgaben der Applikation sauber in unterschiedliche Tiers aufzuteilen. Durch die Unabhängigkeit der einzelnen Tiers konnten wir jeweils die Technologien einsetzen, welche für die Aufgabe am besten geeignet war. So konnten wir im Frontend mit Vue und 3d Force Graph eine saubere Darstellung erreichen, welche auch eine gute User Experience bietet. Im Backend haben wir mit Python eine Sprache gewählt, welche gut geeignet ist, um Tätigkeiten wie das Abfragen Netzwerkgeräte oder die Implementation der Logik umzusetzen. Auch bei den Datenbanken haben wir für die jeweiligen Benutzung optimale Lösungen gefunden. MongoDB als leichte Nosql Datenbank, welche gut geeignet ist kleine Datenmengen mit wenig Ressourcen zu verwalten und die Daten gleich im JSON Format zur Verfügung stellt. Mit Neo4j haben wir eine Graphdatenbank, was Sinn ergibt, da wir auch das Netzwerk auf einen Graphen abbilden. Zusätzlich konnten wir sogar einige bereits eingebaute Funktionalitäten wie den Shortest Path Algorithmus auf dem Graphen nutzen, was uns das Implementieren von zusätzlicher Logik oder einer weiteren Library erspart hat.

1.4 Umsetzung

Wir haben eine Webapplikation erstellt, die eine Liste von Zugangsdaten der Netzwerkgeräten verwalten kann. Anschliessend, auf Basis dieser Daten, alle Netzwerkgeräte abfragt und die benötigten Informationen in einer Graphdatenbank abspeichert, die dann visuell als Graph dargestellt werden.

1.5 Implementierungsdetails

1.5.1 Zugangsdaten Speichern

Die Zugangsdaten für die Netzwerkgeräte sind in einer MongoDB abgelegt. Damit Passwörter oder ssh keys nicht unverschlüsselt in der Datenbank liegen, haben wir die Python Library Cryptography Fernet [11], welche anhand eines Keys eine symmetrische Verschlüsslung und beim Abfragen der Geräte wieder eine Entschlüsselung durchführt.

Damit bei der Abfrage des Netzwerkgerätes der Typ des Gerätes bekannt ist, haben wir ein Feld hinzugefügt, bei welchen man dies spezifizieren kann.

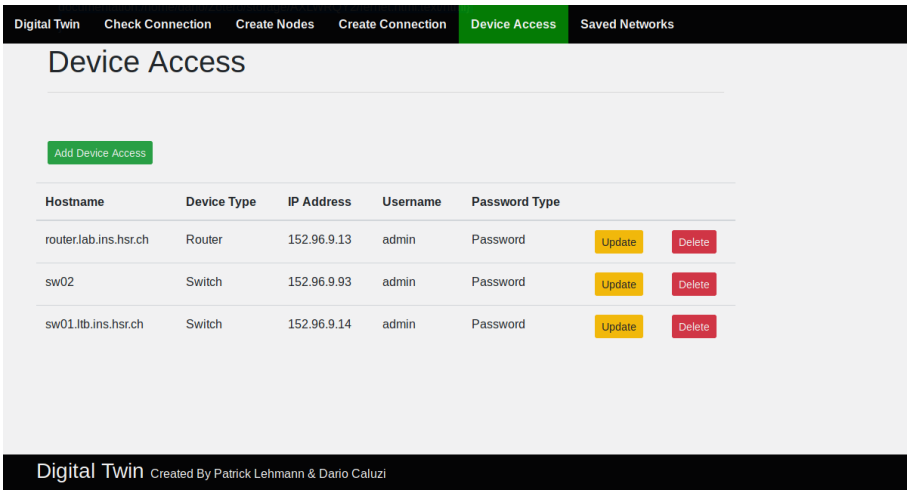


Abbildung 1.1: Ansicht der Benutzeroberfläche um Netzwerkgerätezugänge zu verwalten

1.5.2 Erstellung Digital Twin

Beim Abfragen der Netzwerkgerät Informationen haben wir festgestellt, dass die benutzten RESTCONF Schnittstellen der Switches und Router, welche wir vom Institute for Networked Solutions (INS) zu Testzwecken zur Verfügung gestellt bekommen haben, von Softwareversion und Gerätetyp sehr stark abhängig waren. Bei den Router beispielsweise haben wir zuerst die Cisco IOS XE Software Version 16.06.01 benutzt, womit es uns leider nicht möglich war, operative Daten abzufragen, welche zum Aufbau des Netzwerks für die Verbindungen relevant sind. Nach dem Update auf die Version 16.09.03 konnten alle benötigten Informationen abgefragt werden. Bei den Switches war die NXOS Version 9.2.3 notwendig. Diese Problematik hat uns sehr viel Zeit gekostet, was zu grosse Verzögerungen der Meilensteine 3 *Anforderungsspezifikation* und 4 *End of Elaboration* zur Folge hatte, da wir nicht mit Sicherheit alle Anforderungen mit RESTCONF erfüllen konnten.

Da die Abfragen auch mit den funktionierenden Versionen noch von Router zu Switch sich unterscheiden, mussten wir die Abfragen dynamisch zum Typ separieren. Dazu wird das entsprechende Feld aus der Datenbank zur Verwaltung der Netzwerkgeräte verwendet.

Die Antworten der RESTCONF Abfragen an die Router haben eine schöne strukturelle Separierung der Informationen und konnten so spezifisch abgefragt werden. Bei den Switches jedoch gab es teilweise Inkonsistenz in den Daten, sodass beispielsweise die Beschreibung der Interface nicht einheitlich

war, einmal mit *eth1/1* und auf einer anderen Abfrage mit *Ethernet1/1*. Dies hat dazu geführt, dass wir alle Daten zuerst ins Memory des Backends zwischenspeichern und einheitlich konvertieren mussten, bevor sie in der Datenbank abgespeichert werden konnten.

Damit wir die Verbindungen zwischen den einzelnen IP Adressen auf den Network Layer erstellen konnten, benötigten wir Funktionalität, welche bestimmen konnte, welche IP's im selben Subnetz sind und welche nicht. Da auch hier die Daten von RESTCONF teilweise mit Prefix Netzmaske (/24) oder mit Netzmaske (255.255.255.0) angegeben waren, hätte hier eine eigene Implementation viel Zeit und Aufwand benötigt. Deshalb haben wir uns dafür entschieden, die Python library ipaddress [12] zu verwenden. Sie bietet die Logik, um aus verschiedenen IP Adressen die Netzwerkadresse und Subnetze zu berechnen, sowie zu überprüfen, ob eine IP in einem Subnetz ist.

1.5.3 Darstellung

Nachdem der Digital Twin erstellt wurde, kann der Graph auf der Webapp graphisch, nach OSI-Layer und Netzwerkgeräten getrennt, angezeigt werden. Der dargestellte Graph kann auch auf die gewünschten Teilbereiche eingeschränkt werden. In der Abbildung 1.2 ist ein Screenshot ersichtlich, wie der komplette Digital Twin in der Webapplikation dargestellt wird. Im selben Screenshot kann ist oben links unter der Navigation zu sehen, dass es zwei Dropdown Menus gibt, wo man die gewünschten Layer oder die einzelnen Hostnamen auswählen kann.

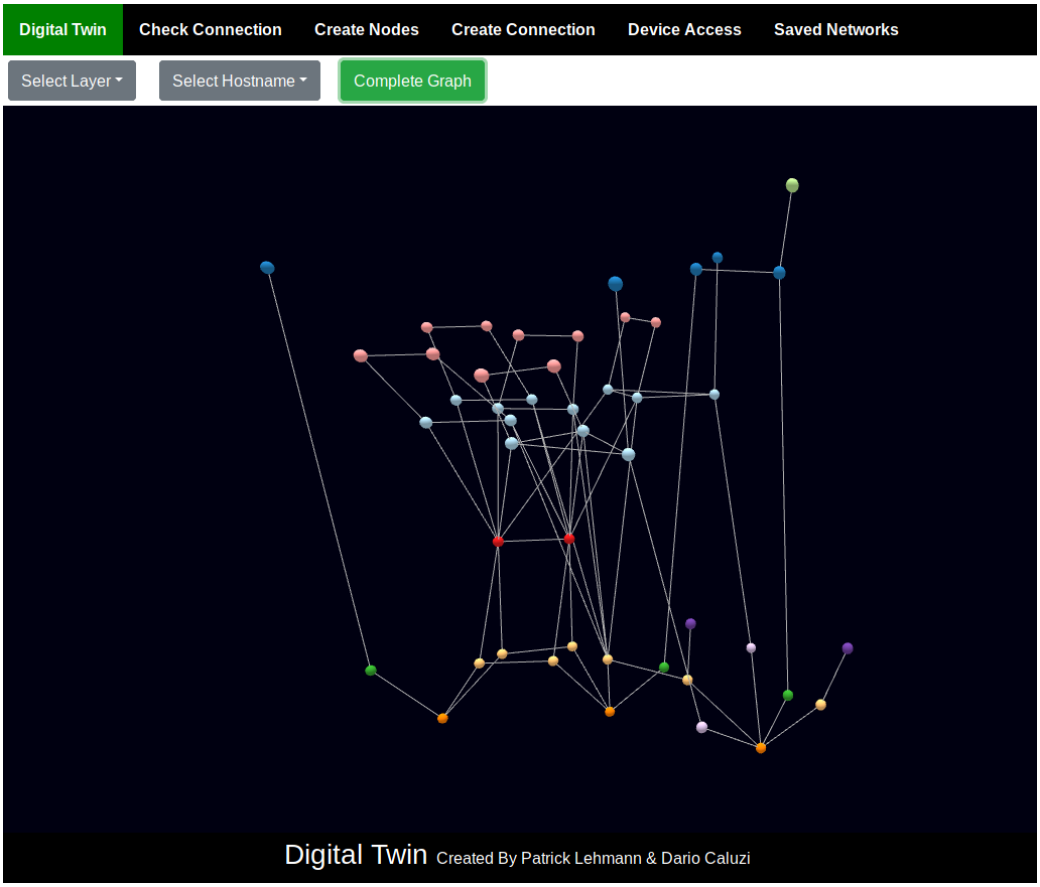


Abbildung 1.2: Graphische Darstellung des kompletten Digital Twin

1.5.4 Verbindung überprüfen

Um eine Verbindung von zwei Endpunkten zu überprüfen, verwenden wir intern den Shortest Path Algorithmus. Genauer gesagt, wird die in Cypher von Neo4j eingebaute Implementation des Shortest Path Algorithmus dafür verwendet.

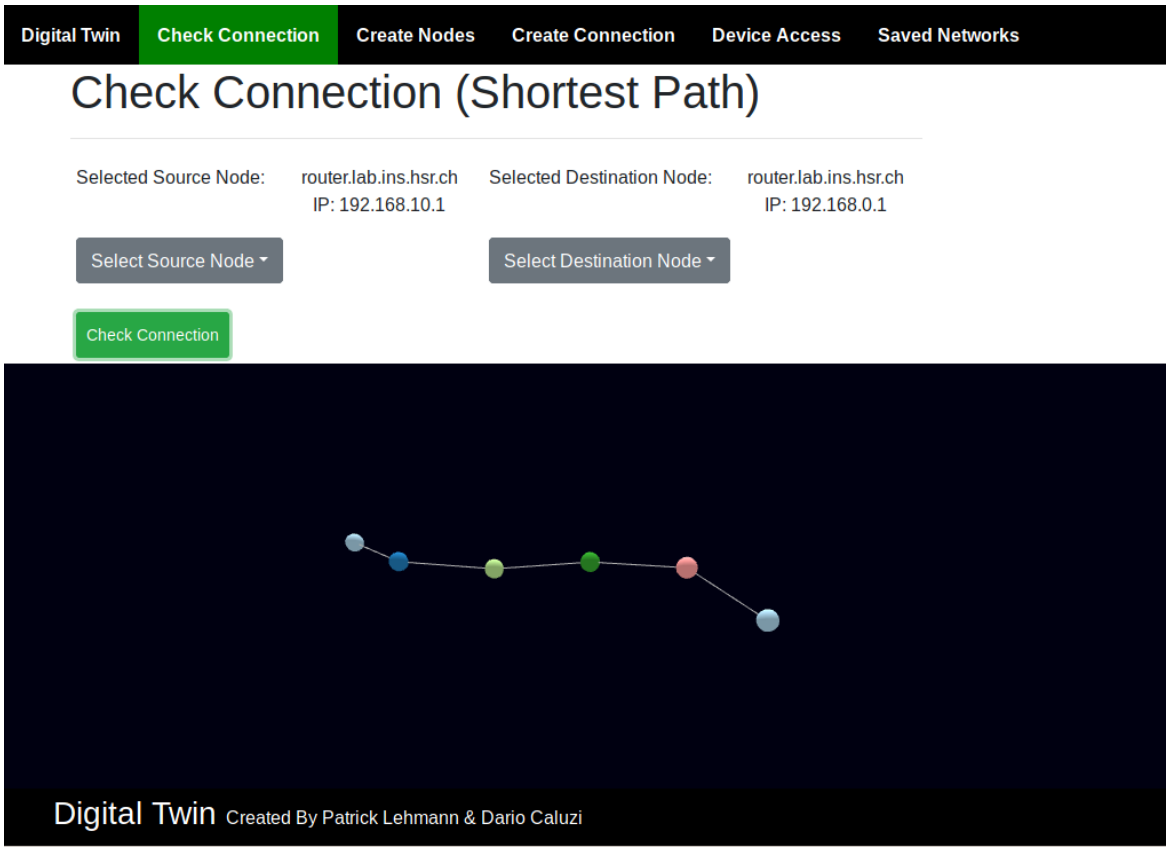


Abbildung 1.3: Bentzerinterface um die Konnektivität von zwei Endpunkten zu überprüfen

1.5.5 Implementierter Umfang

Der in der Architektur definierte minimal Umfang, ist auch der, welcher wir bei der Umsetzung erreicht haben. Die enthaltenen Funktionalitäten wurden gemäss Use Cases implementiert, ausser bei Use Case 4 gibt es eine Abweichung, welche nachfolgend diskutiert wird.

Der Einfachheit halber sind hier die implementierten Use Cases aufgelistet:

- Use Case 1 Netzwerk abspeichern
- Use Case 2 Digital Twin manuell erweitern
- Use Case 3 Netzwerkgeräte Zugang abspeichern
- Use Case 4 Digital Twin oder Netzwerk darstellen (nur eingeschränkt)
- Use Case 5 Verbindung zwischen zwei Endpunkten überprüfen

Einschränkung in Use Case 4

Die nicht erreichte Diskrepanz im Use Case *Digital Twin oder Netzwerk darstellen* bezieht sich darauf, dass es nicht möglich ist, ein Netzwerk darzustellen, ohne dass vorher ein Digital Twin erstellt wurde.

Aufgrund der hohen Komplexität beim Abfragen der Netzwerkgeräte, insbesondere bei den Switches, war es uns nicht möglich, alle Informationen ohne Zwischenverarbeitung im Backend abzugreifen. Im Backend werden einige Teile der Logik wie beispielsweise das Erstellen von Verbindungen durch das Erstellen der Datenbank auf Basis der abgefragten Daten abgebildet. Damit also auch ein Netzwerk ohne das Abspeichern der Datenbank dargestellt werden könnte, hätte dies eine separate Implementierung dieser Logik nur für diesen einen Fall vorausgesetzt. Weil dies viel Aufwand zur Folge gehabt hätte, haben wir uns aus zeitlichen Gründen dazu entschieden, dass immer zuerst ein Digital Twin des Netzwerks erstellt und dann dargestellt wird.

Offene Use Cases

Aus zeitlichen Gründen konnten die weiteren definierten Use Cases leider nicht umgesetzt werden. Die Webapplikation sowie das Backend der Applikation wurden so umgesetzt, dass diese Use Cases, welche wir nicht implementieren konnten, noch in der Applikation ergänzt werden können.

Layer und Technologien

Wie auch bei den Use Cases haben wir die Layers und Technologien implementiert, welche in den Architektur definiert wurden. Natürlich ist auch hier die Architektur so ausgelegt, damit weitere Technologien im Digital Twin integriert werden können.

1.6 Ergebnisdiskussion

Das Ziel der Arbeit war es, ein Mapping zu erstellen, welches es ermöglicht, ein Netzwerk durch einen Graphen zu repräsentieren und anschliessend ein Tool zu erstellen, welches dieses Mapping verwendet, um einen Digital Twin zu erstellen.

Dies ist uns gelungen.

1.6.1 Analyse

Zur Analyse ist zu sagen, dass wir den Aufwand und die Komplexität, um ein Netzwerk auf einen Graphen zu abzubilden, unterschätzt haben. Die reine Anzahl an Netzwerktechnologien, welche in einem Router nach heutigen Industriestandard eingesetzt werden, ist sehr gross. Es gibt unterschiedliche Protokolle mit teilweise sogar unterschiedlichen Versionen für die selben Aufgaben. Deshalb haben wir uns auf Technologien bis und mit Layer 3 im OSI Modell eingeschränkt. Selbst dann gab es noch enorm viele Möglichkeiten, ein Netzwerk auf ein Graphen zu mappen. Das Problem dabei ist auch, dass es nicht eine richtige und falsche Lösung gibt. Je nachdem auf was für einen Teil man sich fokussiert, haben wir jeweils unterschiedliche Arten gefunden, das Mapping zu erstellen. Die einzige Art, wie das Konzept getestet werden konnte, ist anhand der Dummy Datenbank und einer Überprüfung gewisser Operationen im Netzwerk. Die Resultate mussten dann mit den analogen Operation im Graphen verglichen werden.

Schlussendlich muss die Applikation in einem richtigen Netz eingesetzt werden, um zu sehen, ob das Konzept wirklich gut funktioniert. Wir sind aber sehr zufrieden mit dem Endprodukt und glauben, dass wir einen guten Kompromiss zwischen allen Technologien gefunden haben.

1.6.2 Erstellung Digital Twin

Wie im Kapitel Implementationsdetails 1.5 beschrieben, konnten wir mit RESTCONF alle Daten, die wir benötigten, von den Netzwerkgeräten abfragen. Aufgrund der unterschiedlichen YANG Models auf den verschiedenen Gerätetypen, mussten wir dies unterscheiden können. Diese Unterscheidung hatte zur Folge, dass wir das Abfragen der Geräte pro Typ separat machen mussten. In unserem Fall hatten wir im Testnetzwerk nur zwei unterschiedliche Typen, was zwei Implementationen bedeutet. Wenn man nun aber ein grösseres Netzwerk hat und womöglich noch weitere Gerätetypen betreibt, hätte dies zur Folge, dass man für jeden weiteren Typ, der sich in der RESTCONF unterscheidet, eine weitere Implementation erstellen müsste. Das ist nicht sehr flexibel und suboptimal.

Möglicherweise wäre hier eine andere Lösung für das Abfragen der Information von den Netzwerkgeräten besser geeignet gewesen und für mehrere Systeme einsetzbar. Würde aber natürlich auch da voraussetzen, dass auch alle Informationen, die benötigt werden, abgefragt werden können, was hier weitere Analyse bedeuten würde.

Das Erstellen des Digital Twin hat jedoch relativ gut funktioniert. Um die Funktionalität zu testen, haben wir in unserem kleinen Testnetzwerk Anpassungen gemacht, wie z.B. neue Virtual Local Area Netwerk (VLAN)'s eingerichtet, und anschliessend den Digital Twin erstellt. So haben wir manuell kontrolliert, ob alle Informationen jeweils im Digital Twin, wie geplant, vorhanden sind. Optimal könnte man noch weitere Tests mit umfangreicheren Netzwerken durchführen. Leider hat es uns wegen der limitierten Zeit der Studienarbeit nicht gereicht, dies ausführlicher testen.

Im Graphen in der Datenbank haben wir noch mehr Informationen über das Netzwerk abgespeichert als momentan im Frontend für den Digital Twin angezeigt und verarbeitet wird. Beispielsweise für OSPF haben wir bis jetzt nur die ID und Area verwendet, haben aber auch noch Hellointervall, Deadintervall sowie Authentication abgespeichert. Für PVST haben wir noch Infos wie Rootbridge, Bridge ID und Priority, welche noch nicht weiter verwendet werden. Diese Daten sind bereits in der Datenbank, da wir bei der Analyse erarbeitet haben, dass sie für die Funktionsfähigkeit der jeweiligen Protokolle wichtig sind. Eine entsprechende Implementation dazu müsste allerdings noch erweitert werden.

1.6.3 Darstellung

Die Darstellung des Graphen im Frontend, wie es in Abbildung 1.2 zu sehen ist, finden wir ist gut gelungen. Die farbliche Hervorhebung der einzelnen Endpunkttypen in Kombination mit der visuellen Auftrennung, dass alle Nodes vom gleichen Typ auf einer Ebene sind, macht den Graphen viel übersichtlicher. Auch die dynamische Generierung ist optimal, dass es unabhängig der Datenbasis immer nach dem gleichen Stil aufgebaut ist.

Da der Graph jedoch mit einer steigenden Anzahl Netzwerkgeräte im Digital Twin auch unübersichtlicher werden kann, ist es eine nützliche Option, dass man den Graph auf spezifische Layer oder Netzwerkgeräte einschränken kann. So findet man sich auch schneller zurecht, wenn nach einem ganz bestimmten Node oder einer Verbindung gesucht wird.

Natürlich darf nicht aussen vor lassen werden, dass wir das Konzept für die Darstellung entwickelt haben, und es daher für uns logisch und nachvollziehbar ist. Es wäre also empfehlenswert mit ausgewählten Personen im richtigen Betrieb, ein Usability Test zu tätigen. Damit könnte herausgefunden werden, ob die Darstellung intuitiv für neue Benutzer zu bedienen ist.

1.7 Zusammenfassung und Ausblick

In diesem Kapitel werden wir nochmals auf alle Aspekte der Studienarbeit zurückblicken und äusserst bemerkenswerte Aspekte erneut aufgreifen und darstellen, welchen Einfluss diese auf das Endresultat der Arbeit hatten. Anschliessend werden wir auch einige Punkte erwähnen, wie es mit dem Projekt weitergehen könnte.

1.7.1 Projektplanung

Weil dies das erste Projekt für uns beide war, bei welchem man zuerst eine genaue Analyse durchführen musste, ob es möglich ist ein Netzwerk auf einen Graphen abzubilden, haben wir bei der initialen Planung viel zu wenig Zeit dafür eingeplant. Durch die Betreuer wurden wir jedoch darauf aufmerksam gemacht und konnten dies nochmals etwas umplanen. Schlussendlich hatten wir jedoch durch unerwartete Verzögerungen der RESTCONF Abfragen etwas an Zeit verloren, welche wir schon für andere Entwicklungsarbeiten eingeplant hatten. Da die Studienarbeit in Aufwand und Enddatum fix ist, konnten wir daher die Zeit nur durch Mehraufwand wieder aufholen. Trotzdem hatten wir leider keine Zeit, um wie geplant, die Applikation ausführlich durch Unit Tests zu testen.

Da das Problem, so wie es in der Aufgabenstellung beschrieben ist, noch eine viel grössere Problemdomäne aufweist, als das, was wir umgesetzt haben, war es zu Beginn schwierig, einen guten Start in das Projekt zu finden. Dank der Definition des Umfangs der Studienarbeit und der Festlegung der Prioritäten durch die Betreuer konnten wir gezielt vorwärts arbeiten.

Im Nachhinein hätten wir beim Planen der Arbeit zu Beginn bereits jeweils ein kleines Zeitfenster für ungeplante Verzögerungen mit eingeplant, damit man danach nicht sofort in Verzug ist.

1.7.2 Analyse

Die Analyse hat sehr viel Denkarbeit von uns verlangt. Das Konzept. wie ein Netzwerk auf einen Graphen abgebildet werden soll, ist der Schlüsselpunkt der Arbeit gewesen.

Aus unserer Sicht war es auch absolut notwendig, dass in der Analyse alle Aspekte der Abbildung beachtet wurden, auch wenn bereits feststand, dass die entsprechende Implementation nicht im Umfang der Studienarbeit getätigt werden kann. Hinzu kommt auch das Ziel, das die Applikation nach dem Abschluss unserer Arbeit von uns oder anderen Entwicklern einfach mit weiteren Funktionalitäten ausgebaut werden kann. Natürlich gehört dazu auch, das Ausbauen auf den kompletten OSI Layer Stack. Des Weiteren war uns das Unterstützen von weiteren Netzwerkgeräten und Abfragetechnologien wichtig.

Für das abschliessende Konzept der Abbildung vom Netzwerk auf den Graph wären noch ausführlichere Test sinnvoll gewesen. Aus dem Grund, dass das Endprodukt komplett von dem Konzept abhängt, müsste man bereits vor Beginn der Implementation sicherstellen können, dass das Konzept funktionsfähig ist. Am besten geeignet dafür wäre ein Proof of Concept, wo man mit ausgewählten Spezialisten, unterschiedliche Netzwerke und reale Probleme abbildet und versucht den Graphen zu verifizieren. Leider ist dies mit einem grossen Aufwand verbunden, welchen wir im Umfang der Studienarbeit leider nicht so einplanen konnten. Wir gehen aber davon aus, dass wir mit dem Aufwand den wir hatten, das Konzept auf einen funktionierenden Stand gebracht haben.

1.7.3 Umsetzung

Bei der Umsetzung der Arbeit respektive der Entwicklung von Frontend und Backend konnten wir gute Fortschritte machen. Die Aufteilung in Frontend Lead und Backend Lead hat uns sehr geholfen, den Überblick zu behalten und gezielt zu arbeiten. Die Arbeit konnte natürlich auch sehr gut aufgeteilt werden wie die Abfragen der Netzwerkgeräte, das Erstellen des Digital Twin, das Aufbereiten der Daten im Backend, das Darstellen des Graphen und das manuelle Erweitern des Digital Twins im Zusammenhang mit der User Experience im Frontend. Bei Problemen konnten wir uns jederzeit gegenseitig unterstützen.

Auch bei der Umsetzung haben wir sehr stark darauf geachtet, dass die Lösung einfach erweiterbar ist. Dies konnten wir mit der gewählten Architektur sehr gut machen. Das Backend als API anzubieten hat sich gelohnt.

1.7.4 Ausblick

Der nächste optimale Schritt wäre, den Digital Twin in einer realen Umgebung ausführliche zu prüfen. So könnte rasch eventuelle Schwachstellen sowie Optimierungsmöglichkeiten festgestellt werden. Wenn das Produkt diese Prüfungsphase hinter sich hat und die Funktionsweise bestätigt wurde, kann der Digital Twin mit den bisher implementierten Funktionalität bereits eingesetzt werden.

Parallel dazu kann man die Applikation wie gewünscht weiter ausbauen. Einige Use Cases dazu sind in der Analyse unserer Studienarbeit bereits ausgearbeitet worden. Diese Use Cases können als fully dressed erweitert und anschliessend die Funktionalitäten implementiert werden.

Da bisher im OSI Modell nur bis Layer 3 im Digital Twin miteinbezogen wurden, könnte man das Konzept für den kompletten Stack ausweiten, um ein Netzwerk bis zum Präsentations Layer abbilden zu können.

Eine weitere Möglichkeit ist es neue Schnittstellen für unterschiedliche neue Netzwerkgerätetypen anzubieten, um eine grössere Reichweite zu erreichen. Dazu ist natürlich auch nicht ausgeschlossen, eine weiter Technologie für die Abfrage der Geräte anzubieten.

2 Inhalt

2.1 Analyse Spezifikation

In diesem Kapitel werden die einzelnen Komponenten analysiert, die für einen digitalen Zwilling gebraucht werden.

2.1.1 Darstellung von einem Netzwerk durch Graphen

In vielen Bereichen wie Biologie, Physik, Information und Engineering werden Netzwerke verwendet, um komplexe Systeme zu repräsentieren. Meistens werden solche Netzwerke in Graphen dargestellt. Dabei werden Nodes eingesetzt, um Beziehungen gewisser Entitäten und Edges zu repräsentieren. In diesem Kapitel wird ausgearbeitet, wie ein Computernetzwerk durch einen solchen Graphen dargestellt werden kann.

Endpunkt

Die Endpunkte in Computernetzen unterscheiden sich je nach OSI-Layer, der abgebildet werden soll. Beim Data Link Layer werden zum Beispiel die MAC-Adressen der Ports und die der Netzwerkkarten verwendet und in einem Graph miteinander verbunden. Dadurch kann ein Gerät auf den verschiedenen Layern mehrere Endpunkte besitzen. Jeder Endpunkt bildet ein Node.

Verbindung

Eine Verbindung zwischen zwei Endpunkten wird durch ein Edge im Graph dargestellt. Eine Verbindung kann also nur zwischen zwei Nodes existieren. Beispielsweise könnte dies als Kabel zwischen zwei Geräten vorgestellt werden.

Es wäre technisch möglich mehrere Verbindungen zwischen zwei Endpunkten zu realisieren, aber dies ist nicht vorgesehen für den Digital Twin. Falls mehrere Verbindungen zwischen zwei Geräten bestehen, beispielsweise bei Link Aggregation, kann dies über Verbindungen der beteiligten Ports geschehen. Hingegen sollen mehrere Verbindungen zu unterschiedlichen Nodes möglich sein. Dies wird benötigt, um OSI-Layer übergreifende Verbindungen zu erstellen.

Verbindungsunterbruch
Bei einem Verbindungsunterbruch durch einen Link Ausfall entweder durch einen Hardware Fehler oder eine Fehlkonfiguration wird die Verbindung nicht erstellt und dargestellt.

Wenn ein Verbindungsunterbruch gegeben ist, wird es schwierig herauszufinden, ob es sich um eine Fehlkonfiguration handelt. Man müsste viel Zeit für die Interpretation der Konfiguration oder des nicht aktiven Status des Ports investieren, damit dies umgesetzt werden kann.

Bei einem Verbindungsunterbruch ist es auch möglich, den Graphen mit einer vorherigen Version des Digitalen Twins zu vergleichen. Die Differenz kann dann dem User gezeigt werden, dieser kann dann entscheiden, ob der Wegfall der Verbindung gewollt ist oder ein Ausfall bedeutet.

Verbindungsgeschwindigkeit
Da es unterschiedliche Verbindungen gibt, wie die Geschwindigkeit Links, kann man diese Verbindungen mit Gewichtungen versehen. Die Durchsatzrate z.B. 10Gbit\kann als Gewicht für den Edge verwendet werden. Hier werden alle Verbindungen in Mbit\s angegeben, da Einheiten nicht vermischt werden dürfen. Sonst geben die Algorithmen falsche Resultate zurück.

Layer

Ein Netzwerk wird in unterschiedliche Technologien innerhalb eines OSI Layers aufgeteilt. Deshalb werden wir auch im Graphen in die zum OSI Modell korrespondierenden Layern aufteilen. Für ein Layer wird im Graphen ein entsprechendes Attribut für Nodes und Edges abgespeichert. So können alle Komponenten eines Layers assoziiert werden.

Intra-Layer Verbindung
Eine Intra-Layer Verbindung ist eine Verbindung von zwei Nodes innerhalb desselben Layers. Beispielsweise eine Verbindung von zwei Ports zwischen zwei verschiedenen Layer 2 Switches.

Diese Verbindung wird in Section 2.1.1 beschrieben.

Inter-Layer Verbindung
Eine Inter-Layer Verbindung ist zwischen zwei Endpunkten auf unterschiedlichen Layern anzutreffen.

Dies kann anhand eines Beispiels mit einem Layer 3 Switch erläutert werden. Ein Layer 3 Switch arbeitet sowohl auf dem Data-Link-Layer (Layer 2) wie auch auf dem Network-Layer (Layer 3). Die Geräte können auf Layer 2 in einem VLAN miteinander kommunizieren. Falls Pakete das Subnetz verlassen müssen, wird der Verkehr zum Default Gateway des VLAN geschickt, was bei einem Layer 3 Switch ein VLAN Interface darstellt. Zusammengefasst bedeutet dies, dass pro Switch ein Node für das VLAN gebraucht wird und auf Layer 3 Ebene das VLAN-Interface ein Node darstellt.

Die Inter-Layer Verbindung wäre in diesem Fall zwischen der MAC-Adresse und der IP-Adresse des VLAN Interface. Diese beiden Nodes unterscheiden sich graphisch in der Z-Achse und werden deshalb durch einen orthogonal zum Layer stehenden, gestrichelten Edge dargestellt.

Sublayer

Zwischen den herkömmlichen OSI Layer gibt es noch weitere Sublayer mit verschiedenen Technologien. Auch in einem Sublayer erhalten alle betroffenen Komponenten ein Attribut mit dem entsprechenden Layer. Damit kann ein Sublayer extrahiert oder separat dargestellt werden.

Link Aggregation
Ein Link Aggregation ist ein Zusammenschluss von mehreren Links zu einer logischen Verbindung. Dabei werden die zu sendende und empfangenen Daten auf die beiden physikalischen Links aufgeteilt, wobei die Frame Reihenfolge innerhalb eines Datenflusses nicht verändert wird [13].

Eine Link Aggregation soll im Graphen in einem Sublayer dargestellt werden.

Die beiden Nodes der Links, die aggregiert werden, sollen im Sublayer zu einem Node zusammengeführt werden. Die beiden original Nodes werden durch eine Linien zum neuen Node im Sublayer verbunden. So entstehen aus vier Nodes im Data Link Layer zwei Nodes im Sublayer für Link Aggregation.

VLAN
VLAN erleichtern es Netzwerkadministratoren, ein einzelnes Netzwerk zu partitionieren, um den Funktions- und Sicherheitsanforderungen ihrer Systeme gerecht zu werden, ohne neue Kabel verlegen oder grössere Änderungen an ihrer aktuellen Netzwerkinfrastruktur vornehmen zu müssen. VLAN werden oft von grösseren Unternehmen eingerichtet, um Geräte für ein besseres Traffic-Management neu zu unterteilen.

Für die Identifizierung von VLAN's werden auch Attribute auf den betroffenen Geräten hinzugefügt. Auf diese Weise kann jedes VLAN genau extrahiert oder behandelt werden.

Jeder Node, der im entsprechenden VLAN verfügbar ist, bekommt auch einen Node im Sublayer, welcher mit einer inter-Layer Verbindung 2.1.1 zum ursprünglichen Node im Data Link Layer. Falls ein Endpunkt im VLAN nicht vorhanden ist, wird auch kein entsprechender Node im Sublayer erzeugt.

Spanning Tree Protocol (STP)
STP ist ein Protokoll, um Zyklen in einer Broadcast Domain zu verhindern. Bei einer Anzahl n Switches und e Verbindungen, wobei gilt, dass $\mathrm{e} \geq \mathrm{n}$ ist, können endlose Loops entstehen, was Broadcast Storms verursacht. Um dies zu verhindern, darf es genau $\mathrm{n} - 1$ Verbindungen geben.

STP ermittelt alle Verbindungen und blockiert die Ports von redundanten Verbindungen. Fällt ein Link aus, wird der Spanning Tree neu erstellt, so dass alle Switches erreichbar sind, jedoch keine Zyklen entstehen können.

Graphisch werden wir das in einem Sublayer darstellen. Der Sublayer enthält alle Nodes des Data Link Layers jedoch nur $\mathrm{n} - 1$ Edges. Blockierte Ports sollen nicht dargestellt werden. Alle anderen besitzen den Status Forwarding und werden entsprechend im Graph angezeigt.

Tunneling

Unter Tunneling Protokolle versteht man ein Kommunikationsprotokoll, dass das Transferieren von Daten von einem Netzwerk zu einem anderen Netzwerk ermöglicht. So können Daten von einem privaten Netzwerk über ein öffentliches Netz, wie das Internet, gesendet werden. Dabei werden die Daten in ein anderes Kommunikationsprotokoll gekapselt.

Weil wir Graphen nicht der Technologie entsprechend anpassen, bleibt ein Tunnel eine Node zu Node Verbindung auf dem Netzwerklayer. Da bei einer Tunnel-Verbindung die darunter liegende Architektur unbekannt ist, kann dies in den unteren Layern nicht genau dargestellt werden. Deshalb werden die unteren Layers der Tunnel-Verbindungen nicht im Graphen abgebildet. Anwendungsbeispiele von Tunnels sind VPN-, SSH-Tunnel oder ein Tunnel, welches IPv6 über IPv4-Netzwerke transportiert.

2.1.2 Netwerktypen

In diesem Kapitel werden verschiedene Netzwerktypen näher betrachtet. Das Ziel ist es zu entscheiden, für welchen Typ Netzwerk der Digital Twin eingesetzt werden soll.

Campus Area Network (CAN)

Ein CAN ist ein Netzwerk, das einen Bildungs- oder Firmencampus abdeckt. Beispiele dafür sind Grundschulen, Universitätsgelände und Firmengebäude.

Ein Campus-Netzwerk ist größer als ein Local Area Network (LAN), da es mehrere Gebäude innerhalb eines bestimmten Bereichs umfassen kann. Die meisten CANs bestehen aus mehreren LANs, die über Switches und Router verbunden sind, die sich zu einem einzigen Netzwerk verbinden. Sie funktionieren ähnlich wie ein LAN, indem Benutzer mit Zugriff auf das Netzwerk direkt mit anderen Systemen innerhalb des Netzwerks kommunizieren können.

Ein CAN wird von einer einzigen Einheit, wie beispielsweise dem Campus-IT-Team, verwaltet und gepflegt. Die Netzwerkadministratoren können den Zugriff auf das Netzwerk überwachen, zulassen und einschränken. Firewalls werden typischerweise zwischen CAN und Internet platziert, um das Netzwerk vor unbefugtem Zugriff zu schützen. Eine Firewall oder ein Proxy-Server kann auch verwendet werden, um die Websites oder Internet-Ports, auf die Benutzer zugreifen können, einzuschränken.

Wide Area Network (WAN)

Das WAN ist ein geografisch verteiltes Netzwerk, das mehrere LANs miteinander verbindet. In einem Unternehmen kann ein WAN als Verbindungen zu einer Niederlassung eines Unternehmens, zu Colocation-Einrichtungen, Cloud-Services und anderen Einrichtungen bestehen.

WAN-Verbindungen können über Technologien wie Virtual Private Network (VPN), Multiprotocol Label Switching (MPLS), kommerzielle Breitband Internetverbindungen, aber auch 4G Long-Term Evolution (LTE), öffentliche WiFi oder Satellitennetze bereitgestellt werden. Ein Kunde kann beispielsweise die WAN Infrastruktur auch von einem Internet Service Provider (ISP) als Service mieten.

Internet Area Network (IAN)

Ein IAN verbindet Endpunkte sicher über das öffentliche Internet, so dass sie kommunizieren wie auch Informationen und Daten austauschen können, ohne an einen physischen Standort gebunden zu sein.

Weil ein IAN auf dem öffentlichen Internet aufbaut, werden wir uns nicht auf diesen Netzwerktyp fokussieren, da ein Digitaler Twin nicht möglich ist, wenn mehrere Parteien in dem Netz involviert sind.

Fokus

In dieser Studienarbeit werden wir uns auf Netzwerke vom Typ CAN fokussieren, da die Verwaltung immer bei einer Partei liegt. Wenn mehrere Parteien involviert sind, ist es administrativ schwieriger an alle Zugangsdaten der Geräte zu gelangen, um einen kompletten Digital Twin des Netzwerks zu

erstellen. Sonst ist es schwierig, aus einem nicht vollständigen Digital Twin Aussagen über das derzeitige Netz zu machen, und allenfalls zu testen oder zu erweitern. Zusätzlich haben wir beide auch ein breiteres Wissen und mehr Erfahrung mit CANs als mit WANs.

2.1.3 Graphen Theorie

In diesem Kapitel werden die einzelnen Komponenten eines Graphen näher beleuchtet, die allenfalls für den Digital Twin von Nöten sein könnten.

Graphen

Ein Graph in einem single-layer Netzwerk ist ein Tupel $\mathrm{G} = (\mathrm{V}, \mathrm{E})$. Bei dem V ein Set von Nodes und $\mathrm{E} \sqsubseteq \mathrm{V} * \mathrm{V}$ ein Set von Edges ist, welche die Nodes miteinander verbindet [1].

Der Graph, welcher das Netzwerk darstellt, wird unabhängig von der Netzwerktechnologie sein. Es werden keine UML Symbole für die spezifischen Netzwerkgeräte der diversen Hersteller in dieser Studienarbeit verwendet.

ungerichtete Graphen
Ein ungerichteter Graph (undirected graph) ist ein Graph, bei welchem alle Edges bidirektional sind. Die ungerichtete Verbindung (Edge) wird mit einer Linie repräsentiert.

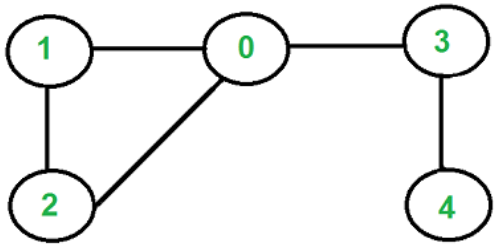


Abbildung 2.1: ungerichteter Graph [14]

gerichtete Graphen
Bei einem gerichteten Graphen (directed graph) zeigen alle Edges von einem Node zum anderen. Eine gerichtete Verbindung wird durch einen Pfeil dargestellt.

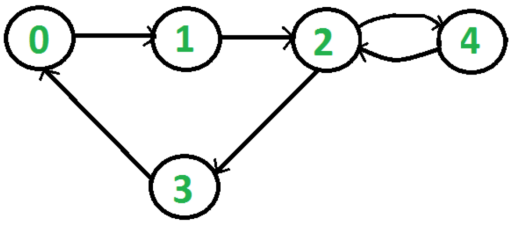


Abbildung 2.2: gerichteter Graph [15]

gewichtete Graphen
Gewichtete Graphen (weighted graph) sind Graphen, bei welchen jeder Edge eine Gewichtung besitzt. Diese Gewichtung kann verschiedene Attribute bedeuten z.B. bei Flügen könnte es die Flugzeit sein oder auch die Distanz zwischen den Destinationen.

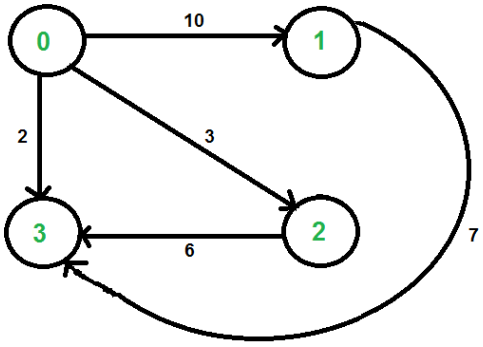


Abbildung 2.3: gewichteter Graph [16]

Subgraphen und Supergraphen

Ein Graph H ist ein Subgraph von G, wenn $\mathrm{V(H)} \sqsubseteq \mathrm{V(G)}$. Dementsprechend ist G ein Supergraph von H.
Durch das Löschen von allen Zyklen und mehrfach Verbindungen zwischen Node Paaren erhalten wir den underlying simple graph.

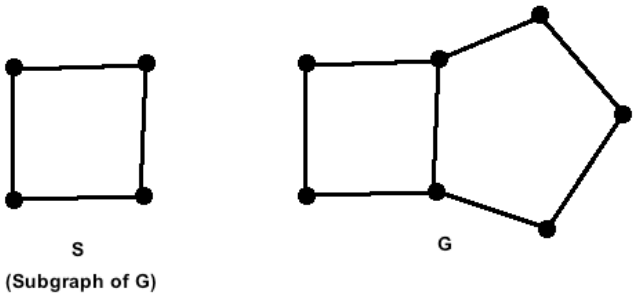


Abbildung 2.4: Graph S ist der Subgraph von Graph G [17]

Walk

Ein Walk in einem Graphen G ist eine endliche Sequenz von Nodes und Edges. Die Nodes und Edges dürfen mehrfach vorkommen. Es ist der Weg von einem Ausgangs Node $\mathrm{V1}$ zu einem End Node Vn

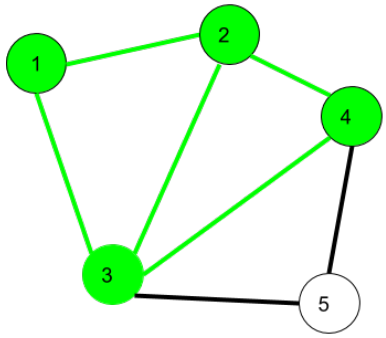


Abbildung 2.5: Walk: 1 -> 2 -> 3 -> 4 -> 2 -> 1 -> 3 [18]

Zyklus (Cycle)

Ein Walk ist zyklisch, wenn der Ausgangs Node und der End Node die gleiche sind und eine positive Länge haben. Ein Graph ohne einen Zyklus wird ein acyclic Graph genannt.

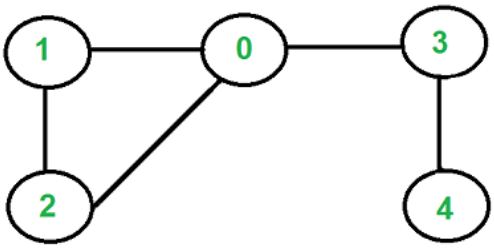


Abbildung 2.6: Ein Zyklus zwischen Node 0, 1 und 2 [14]

Tree

Ein Tree ist ein verbundener acyclic Graph. Das heisst, dass alle Nodes eine Verbindung haben und es keine Zyklen im Graphen gibt. Zudem besitzt der Tree einen Anfangs Node auch Root genannt.

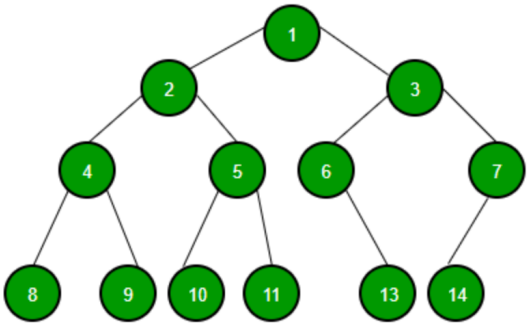


Abbildung 2.7: Tree mit dem Node 1 als Root [19]

Spanning Tree

Bei einem undirected Graph ist ein Spanning Tree ein Subgraph, welcher alle Nodes miteinander verbindet und keine Zyklen hat. Ein Graph kann mehrere Spanning Trees besitzen.

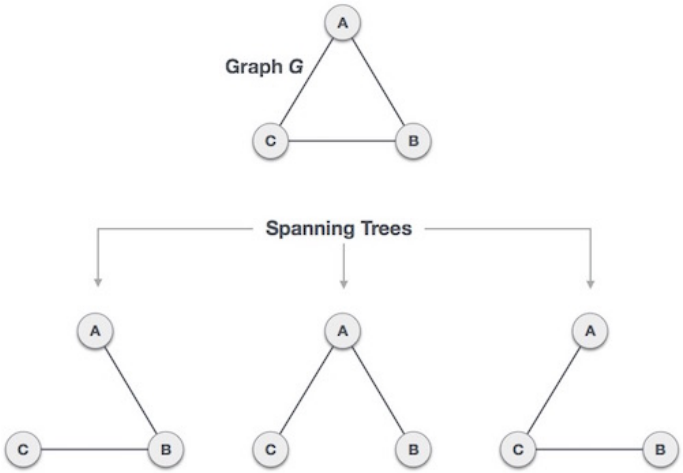


Abbildung 2.8: mehrere Spanning Trees aus einem Graph [20]

2.1.4 Graphen Probleme

In diesem Kapitel werden mehrere Graphen Probleme erläutert, die für ein Netzwerk von Bedeutung sein können. Zudem wird beschrieben, mit welchen Algorithmen die Probleme gelöst werden können. Das Ziel ist es herauszufinden, was für Operationen oder Anwendungsfälle dank der Graphentheorie auf dem Digital Twin möglich sind.

Shortest Path

Beim Shortest Path Problem geht es darum, von einem beliebigen Node x aus den kürzesten Weg zum Zielnode y herauszufinden. Bei gewichteten Graphen ist der Shortest Path der Weg mit den niedrigsten Pfad Kosten. Bei ungewichteten Graphen wird der Weg mit der niedrigsten Anzahl von Verbindungen zwischen den beiden Nodes gewählt.

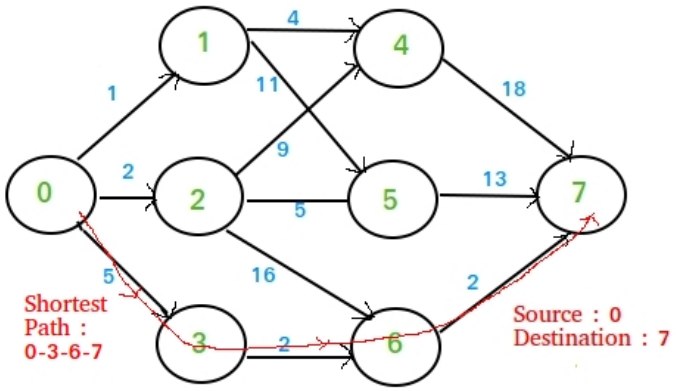


Abbildung 2.9: Shortest Path von Node 0 zu Node 7 [21]

Lösungsalgorithmen: BFS 2.1.5 (bei ungewichteten Graphen), Dijkstra Algorithmus 2.1.5, Bellman-Ford, Floyd-Warshall 2.1.5 und weitere.

Negative Cycles

Bei gewichteten Edges ist es möglich, dass auch negative Kosten vorhanden sind. In diesem Fall muss der Graph darauf geprüft werden, dass keine Negative Cycles vorhanden sind. Ein Negative Cycle ist ein Zyklus, bei dem die aufsummierten Kosten eine negative Zahl ergeben. Ein solcher Zyklus kann immer wieder durchlaufen werden, was die Zykluskosten immer weiter nach unten treibt. Dies gefährdet die mathematischen Grundlagen des Graphes, was beispielsweise zu falschen Berechnungen beim Shortest Path und anderen Algorithmen führen würde.

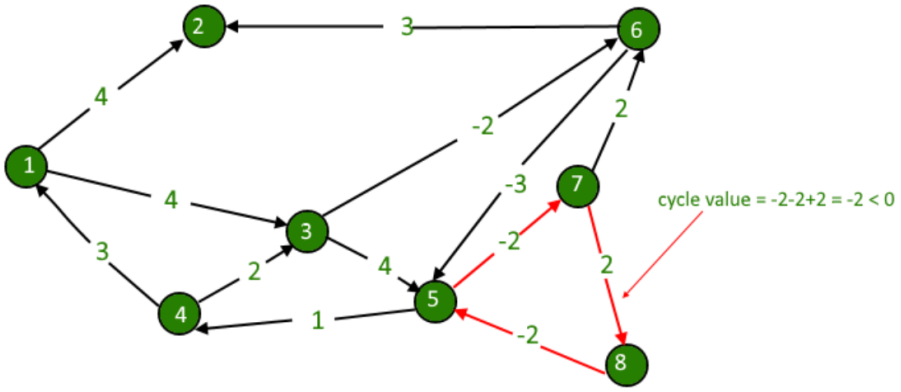


Abbildung 2.10: Negative Cycle zwischen Node 5, 7 und 8 [22]

Lösungsalgorithmen: Bellman-Ford 2.1.5, Floyd-Warshall 2.1.5

Strongly Connected Components

Ein gerichteter Graph ist strongly connected, wenn von jedem Node aus alle anderen Nodes über zyklische gerichtete Verbindungen erreicht werden können. Komponenten, die diese Eigenschaften besitzen nennt man Strongly Connected Components (SCC).
In der nachfolgenden Abbildung sind die Nodes A, B, C strongly connected. Das gleiche gilt für die Nodes D, E, F. Dadurch gibt es zwei SCCs.

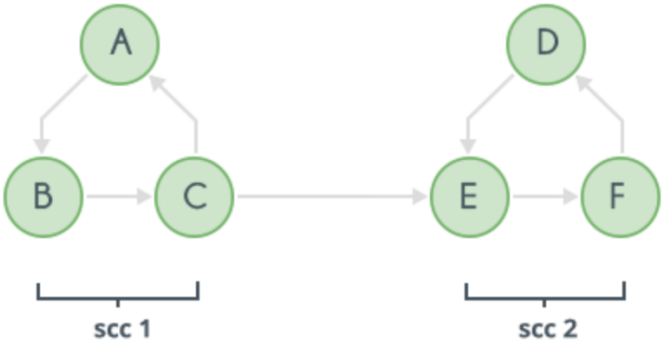


Abbildung 2.11: zwei Strongly Connected Components [23]

Lösungsalgorithmen: Tarjan Algorithmus 2.1.5, Kosaraju Algorithmus

Travelling Salesman

Bei Travelling Salesman geht es um einen gewichteten Graphen, bei dem man versucht herauszufinden, welches der kürzeste Weg ist, um jeden Node genau einmal zu besuchen und zum Ursprungsnode zurückzukehren.

Ein detailliertes Beispiel kann auf dieser Webseite entnommen werden [1]

Lösungsalgorithmen: Held-Karp Algorithmus

Bridges

Bridges sind Edges in einem Graphen, welche bei einem Ausfall, den ursprünglichen Graphen in genau zwei Graphen aufteilt.

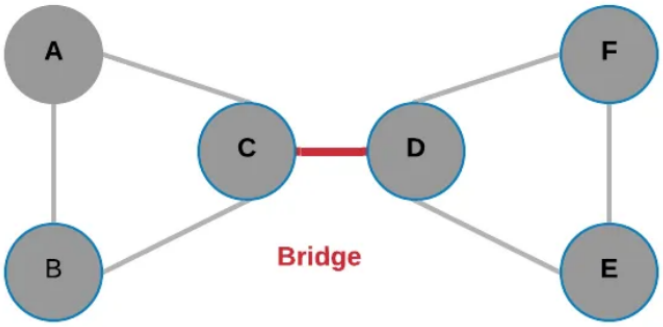


Abbildung 2.12: Bridge zwischen Node C und D [25]

Lösungsalgorithmen: Bridges Algorithmus 2.1.5

Articulation Points

Ein Articulation Point ist die Node Analogie zu einer Bridge. Ein Articulation Point ist ein Node welcher bei einem Ausfall den Graphen in zwei eigenständige Graphen teilt.

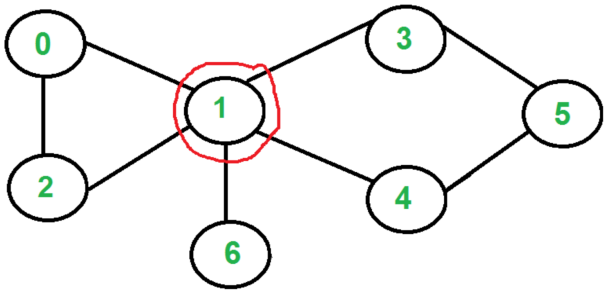


Abbildung 2.13: Node 1 ist der Articulation Point [26]

Lösungsalgorithmen: Articulation Algorithmus 2.1.5

[1] https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/[24]

Minimum Spanning Tree

Ein Minimum Spanning Tree (MST) oder Minimum Weight Spanning Tree für ein gewichteten, verbundenen und ungerichteten Graphen ist ein Spanning Tree mit einem Gewicht kleiner oder gleich dem jedes anderen Spanning Trees. Diesen zu finden ist das Problem.

Der Minimum Spanning Tree ist ein Graph, in welchem alle Nodes eine Verbindung zu einem anderen Node haben, ohne dass Zyklen vorhanden sind. Dabei werden die Edges so gewählt, dass sie einen möglichst geringen Gewicht aufweisen.

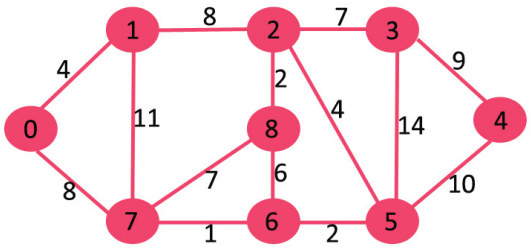


Abbildung 2.14: ungerichteter, gewichteter Graph [27]

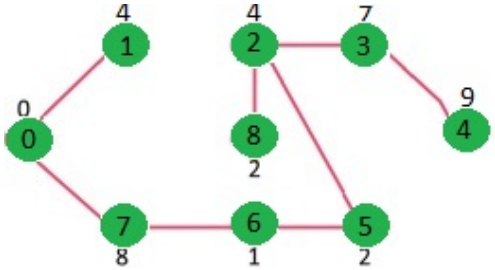


Abbildung 2.15: Minimum Spanning Tree mit der kleinsten Gewichtung [28]

Lösungsalgorithmen: Prim's Algorithmus 2.1.5, Krukal's Algorithmus, Borukva's Algorithmus

Network Flow: max Flow

In der Graphen Theorie ist ein Flow Netzwerk (oder auch Transport Netzwerk) ein gerichteter Graph, bei dem jeder Edge einen Wert für die Kapazität und einen für den Flow erhält. Der Flow ist die derzeitige Auslastung des Edge. Der Wert des Flows einer Edge darf demnach die Kapazität nicht überschreiten.

Der max Flow ist der maximale Durchsatz, welcher von einem Node zu einem anderen existieren kann. Dadurch ist man in der Lage, Engpässe in einem Netzwerk zu identifizieren.

Lösungsalgorithmen: Ford-Fulkerson Algorithmus 2.1.5, Edmonds-Karp Algorithmus und Dinic's Algorithmus

2.1.5 Graphen Algorithmen

Nachfolgend werden mehrere Algorithmen vorgestellt, die an einem Graphen angewendet werden können. Diese stellen meist die Basis für Lösungen der Graphen Probleme dar, welche im vorherigen Kapitel näher gebracht wurden.

Breadth First Search (BFS)

Ein BFS ist ein fundamentaler Algorithmus, um Nodes und Edges eines Graphen zu erforschen bzw. zu traversieren. Er hat eine Komplexität von $\mathcal{O}(V + E)$ und wird oft als Grundlage für andere Algorithmen verwendet. Seine Spezialität ist den kürzesten Pfad von einem Node zu einem anderen bei ungewichteten Graphen zu finden.

Er startet an einem beliebigen Node und besucht zuerst alle Nachbarnodes, bevor mit der nächsten Ebene an Nachbarnodes fortfährt. Alle besuchten Nodes werden einer Liste hinzugefügt. Der Vorgang wird wiederholt, bis alle möglichen Nodes einmal besucht wurden.

Depth First Search (DFS)

DFS ist wie auch BFS ein Traversierungsalgorithmus und besitzt auch eine Laufzeit von $\mathcal{O}(V + E)$. Er wird zudem auch oft als Basis für andere Algorithmen verwendet. Die Differenz zu BFS liegt darin, dass anstelle der Nachbarnodes bei DFS zuerst in die Tiefe des Graphen traversiert wird, bis er am Ende angelangt ist. Danach überprüft er die zuvor besuchten Nodes, ob diese einen weiteren Weg in die Tiefe des Graphen enthalten.

Topological Sort

Ein Topological Sort oder kurz Top Sort ist ein Sortierungsalgorithmus für gerichtete Graphen, die keine Zyklen aufweisen. Daraus entsteht eine Reihenfolge von Nodes unter Berücksichtigung ihrer Abhängigkeiten zueinander. In der Praxis kommt der Algorithmus in Package Managern oder beim Erstellen eines Programm Builds zum Einsatz.

Der Algorithmus ordnet einen Graphen so an, dass für jede gerichtete Edge ab der Node A vor Node B Node kommt. Anders ausgedrückt ist Node B von Node A abhängig. Der Top Sort kann eine topologische Sortierung in einer Laufzeit von $\mathcal{O}(V + E)$ finden. Allerdings kann es sein, dass ein Graph keine topologische Sortierung hat, z.B. wenn der Graph einen Zyklus besitzt.

Nachfolgend ein Beispiel einer topologischen Sortierung

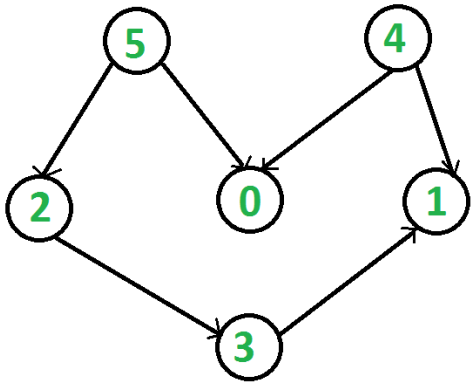


Abbildung 2.16: Graph mit der topologischen Reihenfolge 5 4 2 3 1 0 oder 4 5 2 3 1 0 [29]

Der Algorithmus fängt in diesem Beispiel bei der Nummer 0 an und traversiert so weit, bis keine ausgehende Edges gefunden werden kann, was hier der Fall ist. Anschliessend wird mit dem nächsten noch nicht bereits besuchten Node fortgefahren. Für das Traversieren wird glsDFS 2.1.5 verwendet, wobei vor jedem Rücksprung zum vorherigen besuchten Node der aktuelle Node in umgekehrter Reihenfolge in einer Liste hinzugefügt wird. Die Liste in normaler Reihenfolge ist danach die Sortierung. In diesem Fall 5 4 2 3 1 0 oder 4 5 2 3 1 0. Somit ist die Reihenfolge nicht eindeutig und es kann verschiedene geben.

Dijkstra Algorithmus

Der Dijkstra Algorithmus ist ein Single Source Shortest Path Algorithmus für Graphen ohne negativ Gewichtung. Seine Laufzeit ist typischerweise $\mathcal{O}(E * log(V))$. Dies hängt aber auch von der jeweiligen Implementation ab. Wenn der Graph Negative Cycles 2.1.4 enthält, schlägt er jedoch fehl. Dieser Algorithmus hat sich bei Graphen mit gewichteten Edges durchgesetzt.

Die Funktionsweise des Dijkstra Algorithmus ist auf dieser Webseite ersichtlich [2]

Bellman-Ford Algorithmus

Der Bellman-Ford Algorithmus ist wie auch der Dijkstra Algorithmus ein Single Source Shortest Path Algorithmus. Er hat allerdings eine langsamere Laufzeit von $\mathcal{O}(E * V)$ als Dijkstra. Der Vorteil gegenüber dem Dijkstra ist jedoch, dass er auch funktioniert, wenn der Graph einen Negative Cycle 2.1.4 enthält. Er kann sogar zur identifizierung von Negative Cycles und wo sie auftreten genutzt werden.

So funktioniert der Bellmann-Ford Algorithmus [3].

Floyd-Warshall Algorithmus

Der Floyd-Warshall Algorithmus ist ein All Pairs Shortest Path Algorithmus. Dies bedeutet, er findet den jeweils kürzesten Weg zwischen jeder möglichen Node Paarung in einem Graphen. Er hat eine

[2] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/[30]
[3] https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/[31]

Laufzeit von $\mathcal{O}(V^3)$, was bedeutet, dass er nicht gut skaliert.

Hier ist die Funktionsweise des Floyd-Warshall Algorithums [4].

Tarjans Algorithmus

Der Tarjan Algorithmus findet Strongly Connected Components 2.1.4 in einem Graphen. Dazu nutzt werden sogenannte Low-Link Values verwendet. Der Low-Link Value eines Nodes ist die tiefste Node ID, wenn ein DFS 2.1.5 ausgeführt wird. Er hat eine Laufzeit von $\mathcal{O}(V + E)$.

So funktioniert der Tarjan Algorithmus [5].

Bridges und Articulation Points Algorithmus

Für die Identifizierung von Bridges und Articulation Points kann eine leicht modifizierte Version des selben Algorithmus verwendet werden. Eine Eigenschaft von einer Bridge ist auch, dass beide Nodes, welche die Edge verbindet, Articulation Points sind. Zusätzlich ist ein Articulation Point vorhanden, wenn von einem Node aus die DFS ein Zyklus identifiziert, ausser der Node hat keine oder nur einen ausgehenden Edge. Der Algorithmus hat eine Laufzeit von $\mathcal{O}(V + E)$.

So funktioniert der Algorithmus mit dem Fokus auf Bridges [6] und so mit Articulation Points [7].

Prim's Minimum Spanning Tree Algorithmus

Prim ist ein greedy Minimum Spanning Tree Alogrithmus. Prim ist vor allem gut bei dichten Graphen. Wenn ein Graph jedoch nicht ganz verbunden ist, funktioniert der Prim Algorithmus nicht, dann müsste man ihn auf jedem Teilgraphen separat ausführen. Deshalb ist es sinnvoll zuerst zu überprüfen, ob alle Nodes miteinander verbunden sind. Es gibt eine lazy Version des Prim Algorithmus mit einer Laufzeit von $\mathcal{O}(E * log(E))$ und eine eager Implementation mit der Laufzeit $\mathcal{O}(E * log(V))$.

Die Funktionsweise von Prim's Algorithmus ist hier beschrieben [8].

Ford-Fulkerson Algorithmus

Der Ford-Fuklerson Algorithmus findet den maximalen Flow in einem Graphen. Er basiert beispielsweise auf dem DFS, um alternative Pfade von der Source zur Senke zu finden. Welcher Algorithmus dazu verwendet wird, ist jedoch nicht spezifiziert. Die Laufzeit hängt auch davon ab, was für ein Suchalgorithmus für die Pfade eingesetzt wird.

Eine Funktionsweise des Ford-Fulkerson Algorithmus ist hier [9].

[4] https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/[32]
[5] https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/[33]
[6] https://www.geeksforgeeks.org/bridge-in-a-graph/[34]
[7] thttps://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/ext[35]
[8] https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/[36]
[9] tehttps://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/xt[37]

2.1.6 Typische Netzwerkprobleme

Nachfolgend werden typische Netzwerkprobleme beschrieben, die oft in der Praxis anzutreffen sind. Solche Probleme könnten mit einem Digital Twin eines Netzwerks gelöst werden.

Netzwerkverbindung zwischen zwei Geräten vorhanden

Ein Netzwerkgerät ist nicht pingpar und man möchte herausfinden, ob es eine Netzwerkverbindung auf dem Physical Layer zwischen diesem und einem anderen Gerät gibt. Der Grund für die fehlende Verbindung kann vielfältig sein. Das Netzwerkkabel könnte an einem falschen Port angeschlossen sein oder wurde erst gar nicht eingesteckt. Ein weiterer Grund könnte ein defektes Netzwerkkabel sein.

Speicherung der Konfiguration

Die Netzwerkadministratoren verbessern und warten das Netzwerk im täglichen Betrieb. Dabei werden verschiedene Anpassungen auf dem Command Line Interface der Netzwerkgeräten getätigt. Wenn nun die Konfiguration nicht manuell gespeichert wird, hat dies noch keinen Einfluss auf das Netzwerk. Erst wenn die Geräte neu gestartet werden, verwenden diese die alte Start-Up Konfiguration und somit sind die zuvor gemachten Anpassungen verloren. Dies kann dann zu Einschränkungen des Netzwerks führen.

Spanning-Tree

Das Spanning-Tree Protokoll verhindert Zyklen in der Broadcast-Domäne.2.1.1 Anhand der Bridge-Priorität wird eine Root-Bridge bestimmt, über welchen der Netzwerkverkehr läuft. Die redundante Links, welche nicht den kürzesten Weg zur Root-Bridge führen, werden dabei blockiert. Ein häufiger Netzwerkfehler ist die falsche Auswahl der Root-Bridge, welches meist durch die fehlerhafte Konfiguration der Bridge-Priorität verursacht wird. Die Auswirkung ist dann, dass der Netzwerkverkehr Umwege macht und dadurch die Performance der Applikationen beeinträchtigen kann.

VLANs

Das VLAN wird oft für das Separieren des Netzwerks verwendet.2.1.1 Damit die Geräte innerhalb eines VLAN miteinander kommunizieren können, muss auf jedem Switch das entsprechende VLAN konfiguriert sein. Häufig werden einzelne Switches vergessen gegangen oder falsche VLANs definiert. Die Geräte können somit nicht am Netzwerk teilnehmen. Darüber hinaus kann der Uplink-Port falsch konfiguriert sein und die VLANs können nicht an das nächste Netzwerkgerät weitergegeben werden.

Switch-Port Konfiguration

Port Konfiguration wird oft von den Netzwerkadministratoren von Hand eingestellt. Dabei können einige Änderungen vergessen werden. Beispielsweise falls die Duplex-Einstellung nicht auf beiden Seiten des Kabels gleich sind, entsteht ein Duplex Mismatch. Dies führt dazu, dass der Netzwerkverkehr nicht gleichzeitig gesendet und empfangen werden kann und so zu massiven Performance-Einbussen

kommt.
Des Weiteren spielt die Bandbreiteneinstellungen auch eine wichtige Rolle. Falls auf der einen Seite 100Mbit/s und auf der anderen 1Gbit/s eingestellt ist, dann schickt die eine Seite zu viel Netzwerkverkehr über den Link, was die Gegenstelle nicht verarbeiten kann. Dies führt zu Fehler auf dem Switchport. Je nach Hersteller kommt der Link gar nicht erst hoch.

doppelte IP-Adressen

Wenn die gleiche IP-Adresse von einem anderen Gerät zugeordnet wird, erhalten beide Geräte nur die Hälfte des Netzwerkverkehrs, was zu massiven Performance-Problemen führt. Hingegen falls die IP-Adresse des Standard-Gateways ein zweites Mal vergeben wird, geht der ganze Verkehr, welcher das Netz verlassen soll, an zwei verschiedenen Geräten, was das ganze Subnetz lahm legt.

Linküberlastung

Ein weiteres häufiges Netzwerkproblem ist die Überlastung einer Verbindung. Grund kann eine schlechte Verteilung der Last sein oder dass das Gerät keine höhere Port-Bandbreite unterstützt und die Pakete dadurch weggeworfen werden. Das führt zu langsameren Antwortzeiten, da die weggeworfenen Pakete nochmals gesendet werden müssen je nach Transport-Protokoll.

weitere Themen

- Nutzung unbekannter Switches oder Hubs im Netzwerk
- ungleiche Belastung der Etherchannel-Links
- fehlerhafte Netzwerkkonfiguration (IP-Adresse, Gateway, Subnetzmaske, Subnetz, DHCP-/DNS-Server)
- OSPF-Protokoll nicht richtig konfiguriert: keine OSPF Route (nicht per Ping erreichbar bzw. kein OSPF Neighborship) z.B. falsche Area, falsches Netz propagiert

Troubleshooting von Netzwerkproblemen

Es gibt hilfreiche Werkzeuge, um ein Netzwerkproblem einzuschränken und zu finden. Nachfolgend werden ein paar Möglichkeiten erwähnt.

Ping
Ping ist ein Netzwerk Diagnose Werkzeug, um die Erreichbarkeit von Geräten über das Internet Protocol (IP) Netzwerk zu testen. Dies wird oft als erster Schritt bei der Analyse eines Netzwerkproblems benutzt.

nslookup
Das nslookup Tool erlaubt Abfragen an DNS Server, um die IP-Adresse oder der Namen eines Geräts zu erhalten.

ipconfig / ifconfig
Diese beiden Befehle sind auf den meisten Windows oder Linux Distributionen anzutreffen. Sie zeigen die Netzwerkwerkeinstellungen des Endgeräts an.

Command Line Interface (CLI) Commands
Auf den managebaren Netzwerkgeräten können auf dem CLI Befehle eingegeben werden, welche z.B. VLAN-, STP-Konfiguration oder auch den Status von einzelnen Ports überprüft werden kann. Dies ist sehr hilfreich, um Fehlkonfigurationen ausfindig zu machen.

2.1.7 Netzwerkprobleme durch Graphen abbilden

Durch das Abbilden eines Computernetzwerks in einem Graphen können die im Kapitel 2.1.5 behandelten Operationen und Algorithmen auf die Probleme des vorherigen Kapitels 2.1.6 angewendet werden. Damit kann das Netzwerk analysiert und Voraussetzungen überprüft werden.

Ping Verfügbarkeit

Möchte man feststellen, ob zwei IP Adressen per Ping voneinander erreichbar sind, kann man dies mit dem Shortest Path Algorithmus von Dijkstra 2.1.5 im entsprechenden Netzwerk Graphen herausfinden.

Die Beiden IP Nodes werden durch Source und Destination repräsentiert als Input in die Funktion mitgegeben. Diese gibt dann einen Pfad vom Source zum Destination Node zurück. In diesem Fall ist eine Ping Verbindung vorhanden. Wenn die Funktion keinen Pfad zurückgibt ist auch keine Ping Verbindung möglich.

Zusätzlich zur Ping Erreichbarkeit kann mit dem Shortest Path Algorithmus auch die Round Trip Time der Verbindung mitberechnet werden. Dazu müssen die Latenzen der einzelnen Edges als Attribute abgespeichert werden. Auch die Time to live kann mit dem Ausgangswert (Default meist 64) minus der Anzahl Hops, welche durch Nodes im Shortest Path bekannt sind, berechnet werden.

Spanning Tree

Mit Graphen Operationen kann einerseits im Spanning Tree verifiziert werden, dass keine Zyklen vorhanden sind und ob, es sich um einen Minimum Spanning Tree handelt.

Verifizierung
Durch einen Algorithmus kann überprüft werden, ob der Spanning Tree wirklich keine Zyklen mehr enthält. Beispielsweise durch eine Depth-First Traversal 2.1.5 können Zyklen aus dem Graphen identifiziert werden. Dazu muss der Sublayer des Spanning Tree's analysiert werden. Der ganze Subgraph des Spanning Tree's wird der Funktion übergeben. Gibt die Funktion einen Zyklus zurück, ist klar, dass der Spanning Tree des Netzwerks nicht korrekt ist.

Minimum Spanning Tree
Wurde durch die Funktion bestätigt, dass der Subgraph des Spanning Tree keinen Zyklus enthält, kann noch überprüft werden, ob der vorhandene Spanning Tree auch ein Minimum Spanning Tree 2.1.4 ist. Dazu gibt man den kompletten Graphen des Data-Link Layer als Input in eine Funktion mit. Diese führt beispielsweise den Prim Algorithmus 2.1.5 aus, welcher einen Minimal Spanning Tree zurückgibt. Wenn der zurückgegebene Graph deckungsgleich mit dem Spanning Tree des Netzwerks ist oder die gleiche Gewichtung hat, dann ist auch der Spanning Tree des Netzwerks minimal.

Netzwerkausfall Reduktion

Wenn man wissen möchte, wo im Netzwerk man potenzielle Schwachstellen hat, sind die folgenden Varianten hilfreich. Die potenziellen Schwachstellen werden auf einem Layer identifiziert. Wenn eine gefunden wurde, kann man auf dem darunterliegenden Layer überprüfen, ob beispielsweise ein zusätzlicher Link für Ausfallsicherheit vorhanden ist, welche im Fall eines Unterbruchs zum Einsatz kommen würde.

Link Ausfall
Wenn durch ein Link Ausfall ein ganzer Bereich eines Netzwerks nicht mehr erreichbar ist, ergibt das einen grösseren Schaden für den Netzwerkbetreiber. Um dies zu verhindern, werden im Normalfall wichtige Verbindungen redundant geführt. Durch den Algorithmus zur Erkennung von Bridges 2.1.5 können Verbindungen gefunden werden, welche nicht redundant ausgelegt sind. Zu beachten ist, dass normalerweise Arbeitsstationen nicht mehrfach an ein Netzwerk angeschlossen sind, jedoch Server schon.

Man übergibt den Graphen des Physical Layer als Input in die Funktion, welche anschliessend alle Bridges dieses Graphen zurückgibt. Die returnierten Bridges hätten das Bedürfnis für eine redundante Auslegung.

Netzwerkgerät Ausfall
Auch beim Ausfall von einem Netzwerkgerät kann es dazu kommen, dass ein grosser Teil eines Netzwerks nicht mehr erreichbar ist. Mit dem auf Articulation Points angepassten Algorithmus, kann man auch hier Netzwerk Geräte finden, welche nicht redundant ausgelegt sind.

Netzwerkfluss

Durch Flow Graphen können auch die Duschsatzgeschwindigkeiten von einem Netzwerk analysiert werden.

Maximale Durchsatzrate
Wenn alle Verbindungen im Netzwerk gewichtet sind, können mit dem Ford-Fulkerson Algorithmus 2.1.5 die maximale Bitrate von einem Endpunkt zum anderen berechnet werden. Man gibt der Funktion den Graphen sowie Source und Destination Node an und diese gibt dann den maximal mögliche Durchsatzrate zurück.

Engpässe
Der Ford-Fulkerson Algorithmus kann auch den Minimum Cut des Graphen bestimmen. Der Minimum Cut ist die Verbindung mit der kleinsten Durchsatzrate. Diese ist also der Engpass im Netzwerk. So ist ersichtlich, welche Verbindung am ehesten Ausbaupotenzial hat, um die Netzwerkgeschwindigkeit zu verbessern.

Man übergibt wieder den Graphen mit einem Source und Destination Node als Input der Funktion mit und diese gibt die Verbindung, welche die tiefste Kapazität aufweist, zurück.

VLAN

Durch einen Algorithmus wie Tarjan 2.1.5 können alle Nodes, welche zu einem VLAN gehören, Überprüft werden. Man übergibt der Funktion den Graphen des VLAN's. Als Output werden alle verbundenen Nodes zurückgegeben. Wenn die Anzahl an returnierten Nodes mit der Anzahl an übergebenen Nodes übereinstimmt, sind alle VLAN Endpunkte miteinander Verbunden. Falls diese nicht gleich sind, gibt es ein Problem bei der VLAN Konfiguration.

Doppelte IP Adressen

Beim Aufbau das Graphen werden alle IP Nodes nacheinander erstellt. Wenn man beim Hinzufügen eines neuen Nodes zuerst überprüft, ob eine Node mit der nächsten IP bereits vorhanden ist, kann

man doppelt vergebene IP Adressen finden.

Switch-Port Konfiguration

Auf dem Physical Layer ist es möglich zu überprüfen, ob die Übertragungseinstellungen der Verbindung mit der Eigenschaft des Kabels übereinstimmt. Die Eigenschaft der Edge und die Konfiguration beim Port (Node) werden jeweils als Attribut gespeichert. Anschliessend kann überprüft werden, ob das Konfigurationsattribut beim Node mit dem Bandbreitenattribut des Edges übereinstimmt. Wenn beide Nodes am jeweiligen Ende mit dem Edge Attribut übereinstimmen, haben auch beide Nodes die selbe Konfiguration und eine korrekte Verbindung kann zustande kommen.

2.1.8 Graphen Beispiele

Das unterteilen von einem Netzwerk in Layers und Sublayers ist sehr abstrakt. Aus diesem Grund ist es wichtig, dass durch die Visualisierung eines Netzwerks die logische Trennung klar und einfach ersichtlich ist. Dazu folgen hier einige Beispiele wie ein Graph dargestellt werden könnte. Diese Exemplare sind jedoch nur Ausschnitte eines Graphen und gehen auf unterschiedliche Problemstellungen und Sonderfälle ein. Es werden nicht alle relevanten Daten dargestellt. Zusätzliche Informationen, welche als Attribute zu den Verbindungen und einzelnen Nodes abgespeichert werden, sind in den Graphen nicht ersichtlich.

Auch im Endprodukt soll es möglich sein, dass der Umfang auf Basis der Layer wählbar ist.

Layer 1

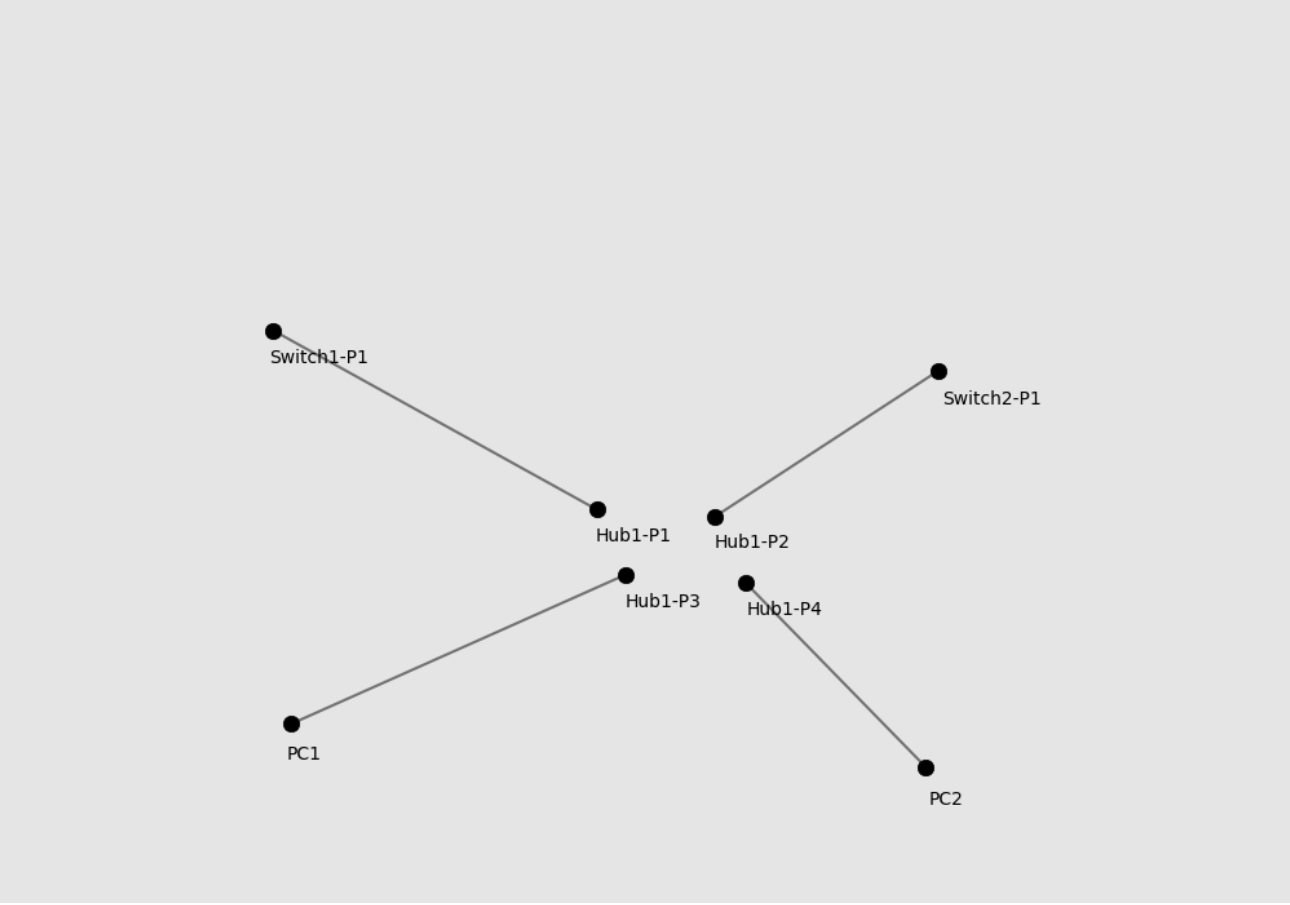


Abbildung 2.17: Layer 1 mit Hub

In dieser Abbildung sehen wir Graphen für ein Netzwerk mit je zwei Switches und PC's, welche über ein Hub miteinander verbunden sind. Hier wird ausschliesslich der Physical Layer dargestellt. Die Verbindungen sind jeweils zwischen den Ports des Hubs und PC respektive des Switch.

Link Aggregation

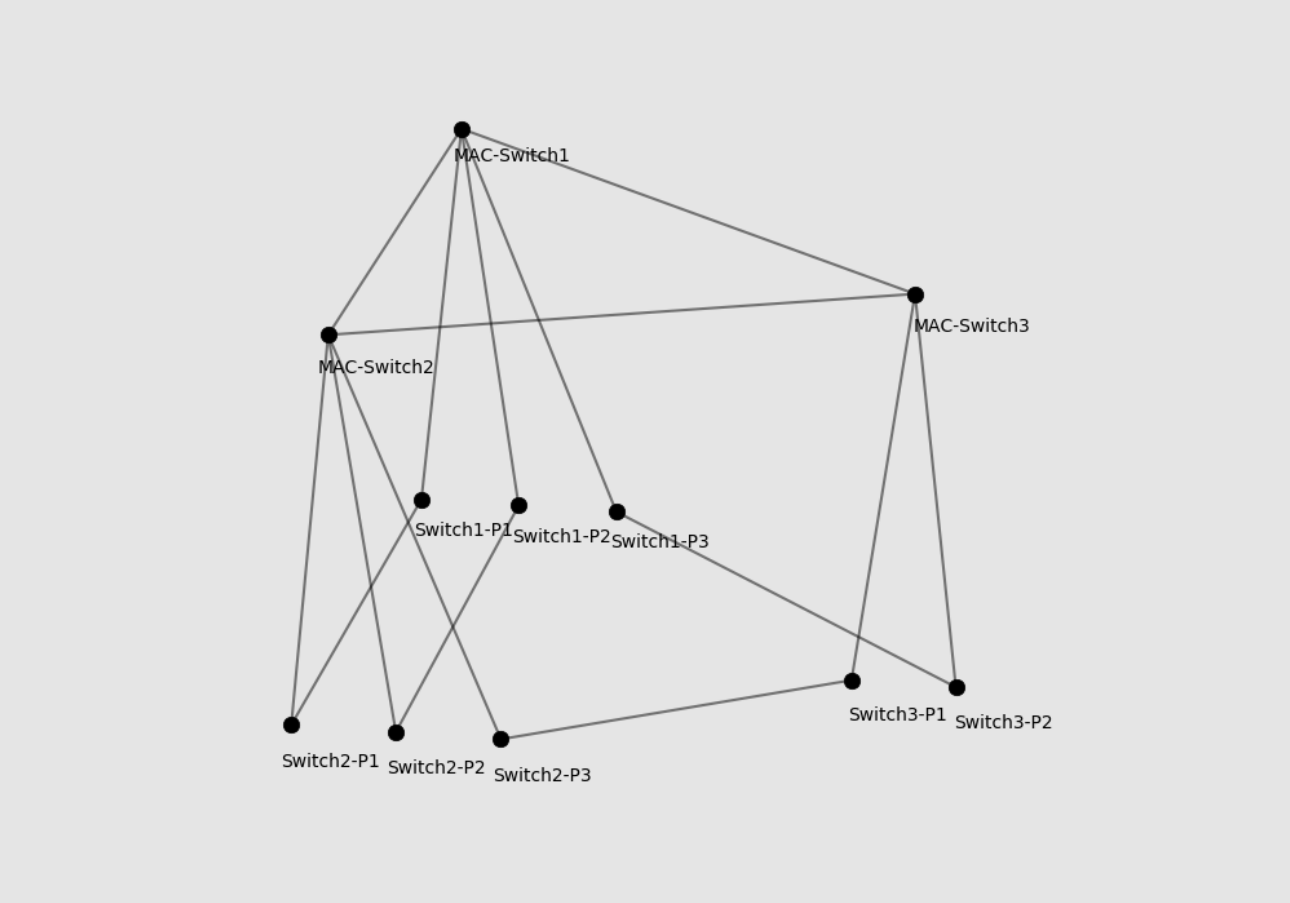


Abbildung 2.18: Link Agregation

In diesem Graphen 2.18 sehen wir ein Netzwerk aus drei Switches. Die Abbildung fokussiert hier auf den Physical sowie auf den Data-Link Layer. Alle Ports eines Switches aus dem Layer 1 sind auch mit der MAC Adresse des Switches verbunden. Zwischen $\mathrm{Switch1}$ und $\mathrm{Switch2}$ ist der aggregierte Link ersichtlich. Auf Layer 1 gibt es zwei Verbindungen zwischen den beiden Switches, einmal von $\mathrm{Switch1}$ Port 1 zu $\mathrm{Switch2}$ Port 1 und parallel dazu von $\mathrm{Switch1}$ Port 2 zu $\mathrm{Switch2}$ Port 2. Auf dem Layer 2 besteht dann jedoch nur noch eine Verbindung zwischen den beiden MAC Adressen der Switches. Zwischen $\mathrm{Switch2}$ und $\mathrm{Switch3}$ sowie zwischen $\mathrm{Switch1}$ und $\mathrm{Switch3}$ gibt es sowohl auf dem Layer 1 wie auch auf dem Layer 2 jeweils nur eine Verbindung zwischen den Geräten.

Spanning Tree

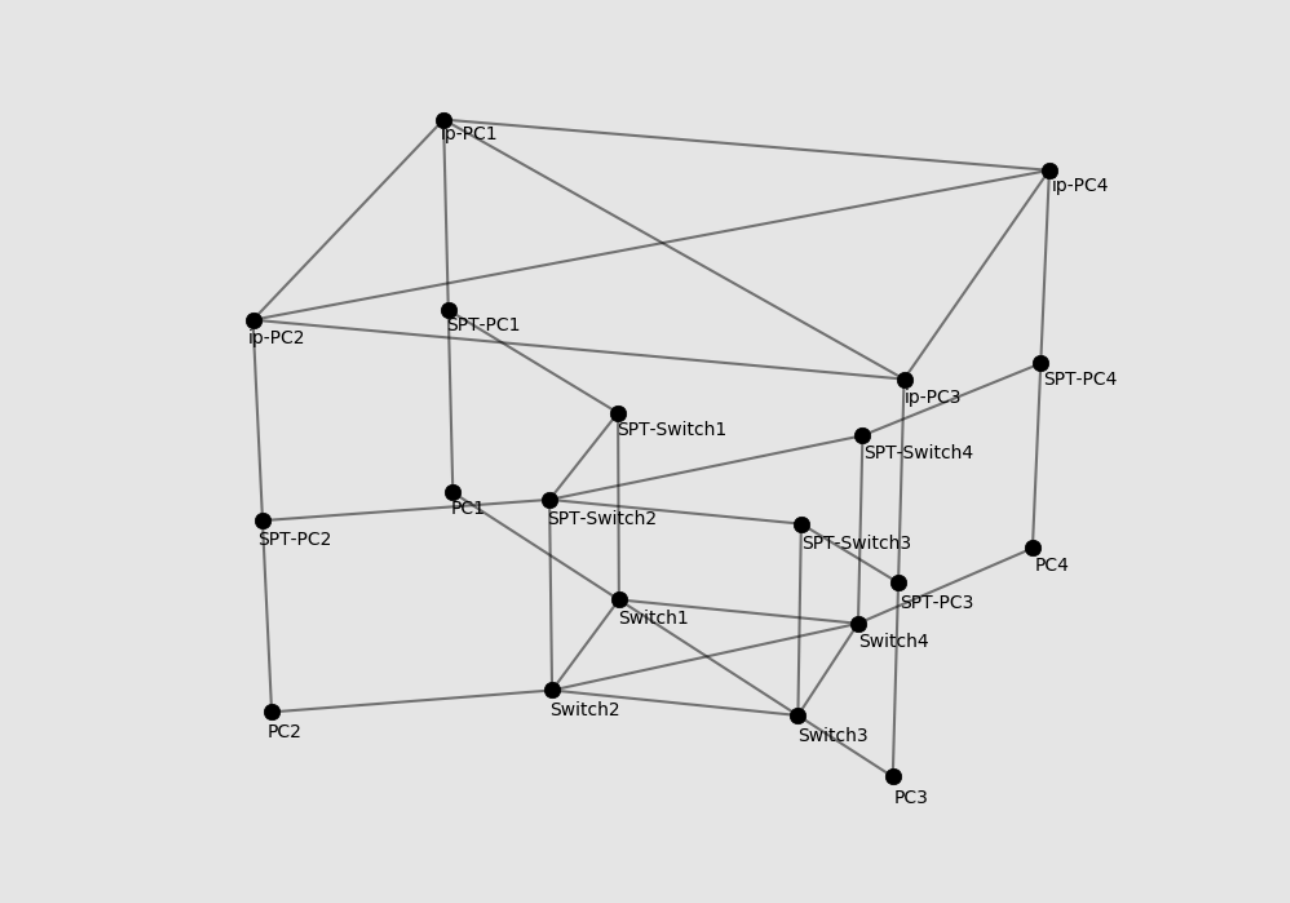


Abbildung 2.19: Spanning Tree

In dieser Abbildung 2.19 ist ein Graph mit vier Switches, vier PC's und einem Spanning Tree Sublayer eines Netzwerks zu sehen. Zusätzlich zum STP Layer ist der darunter liegende Data Link Layer und der darauf aufbauende Network Layer dargestellt. Da auf Layer 2 zwischen den vier Switches Loops vorhanden sind, wird STP benötigt. Im mittleren Sublayer für STP werden dadurch Links auf dem Data Link Layer blockiert, damit keine Broadcast Storms entstehen. Diese Verbindungen fallen also beim STP Sublayer wieder weg. Auf Layer 3 werden die IP Verbindungen der PC's dargestellt, welche auf dem STP Sublayer aufbauen. Aus Sicht von Layer 3 sind die unteren Verbindungen abstrahiert. Wenn es im STP Layer eine Verbindung von PC2 zu PC3 gibt, erscheint diese auch auf Layer 3.

VLAN

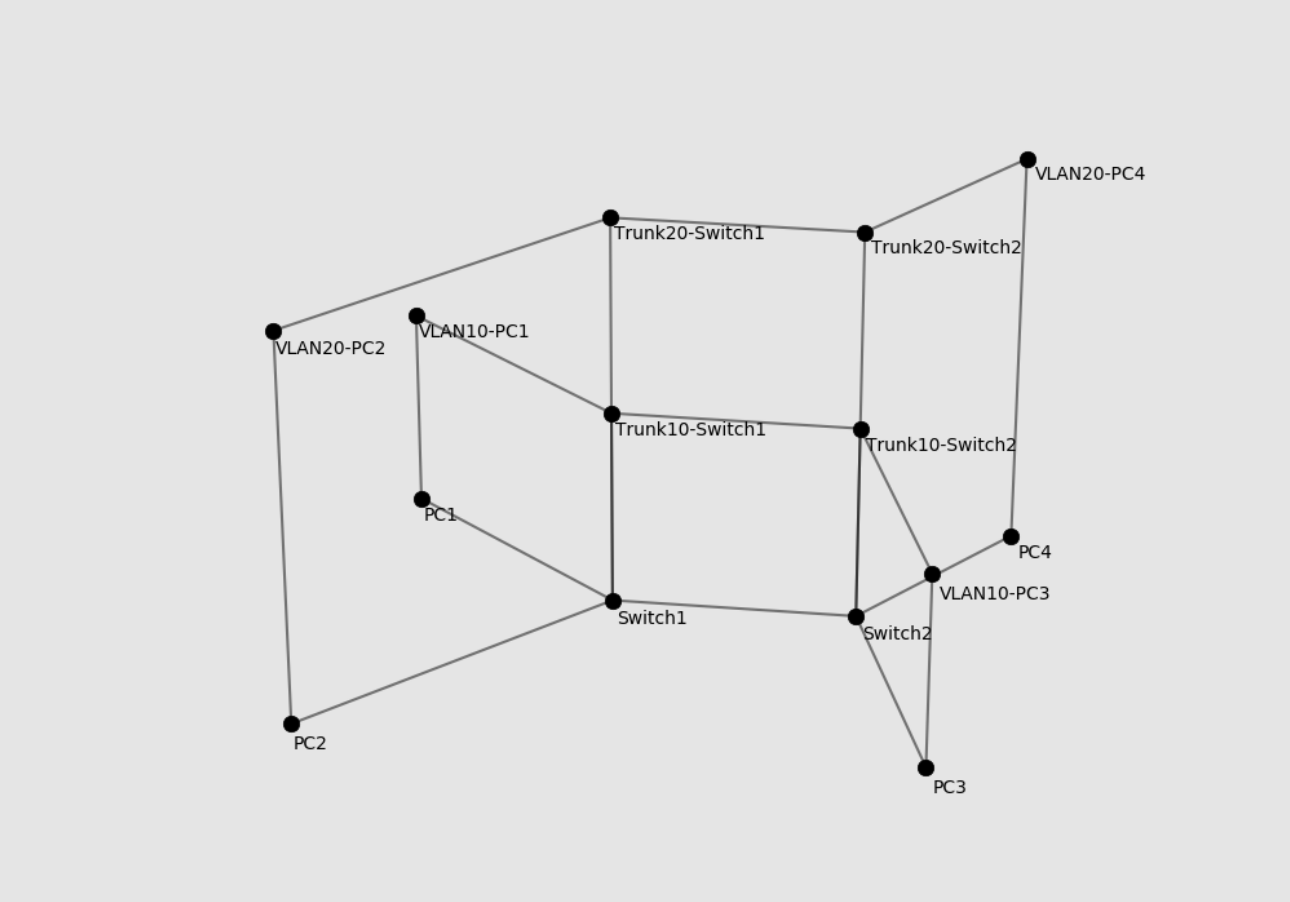


Abbildung 2.20: Layer 2 mit zwei aufbauenden VLAN

Der Graph in Abbildung 2.20 wurde aus einem Netzwerk bestehend aus vier PC's und zwei Switches generiert. Zusätzlich sind zwei VLAN's definiert. $\mathrm{PC1}$ und $\mathrm{PC3}$ sind im $\mathrm{VLAN10}$ und $\mathrm{PC2}$ und $\mathrm{PC4}$ sind im $\mathrm{VLAN20}$. Die beiden Switches haben jeweils beide VLAN's konfiguriert und die Verbindung zwischen ihnen fungiert als VLAN Trunk. Der Graph zeigt den Layer 2 als Grundlage für die beiden Sublayer der VLAN's, die darauf aufbauen. Es ist schön ersichtlich, in welchem VLAN ein PC ist, da er in dem anderen VLAN Sublayer keinen Node besitzt. Nur die Switches, welche beide VLAN's Konfiguriert haben, sind auf allen drei dargestellten Layern durch einen Node vertreten.

2.1.9 Informationsbeschaffung von Netzwerkgeräten

In Kapitel 2.1.1 wurde analysiert, wie man das Netzwerk in einen Graph abbildet. Nun müssen die dazu benötigten Daten von den jeweiligen Netzwerkgeräten abgefragt werden. Nachfolgend werden verschiedenen Varianten zur Informationsbeschaffung näher betrachtet.

Simple Network Management Protocol (SNMP)

SNMP[10] ist ein Protokoll auf der Applikationsschicht im OSI-Layer Modell[11] und ist für den Datenaustausch von Verwaltungsinformationen zwischen Netzwerkgeräten zuständig. SNMP ist weit verbreitet für das Verwalten und Überwachen von Netzwerk-Elementen. Dabei gibt es zwei Akteure. Einerseits der SNMP Agent, welcher auf dem Gerät installiert ist, und andererseits der SNMP Manager, welcher Informationen und Status vom SNMP Agent abfragen oder ändern kann. Auf einem SNMP Agent können spezifische Rules erstellt werden, welche bei Verletzung eine Nachricht an die Management Software schickt z.B. Temperatur der CPU ist zu hoch. SNMP wird vor allem präventiv für eventuelle Wartungsarbeiten verwendet.

Vorteile:

- Open Standard: wird von vielen Herstellern unterstützt
- einfach, es gibt wenige Befehle
- Geräteeigenschaften durch weitere Management Information Base (MIB)s erweiterbar

Nachteile:

- die MIBs sind hardwareabhängig und in Binary gespeichert, was das Lesen davon erschwert
- SNMP Manager Software benötigt
- keine Transaktionen, die beispielsweise einen Rollback von Änderungen ermöglichen
- auf allen Geräten muss der SNMP Agent konfiguriert sein
- je nach Hersteller besitzt der Agent nicht die gleichen OID's
- SNMP wird meist unverschlüsselt eingesetzt

CLI over SSH

Mit SSH ist es möglich, CLI Befehle auf dem Netzwerkgerät ausführen zu lassen. Der Vorteil dabei ist, man kann die gewohnten CLI Befehle des Herstellers für die Konfiguration oder Abfragen der Netzwerkgeräte verwenden. Hingegen müssen die unstrukturierten Daten, die als Antwort zurückgegeben, wieder von Hand geparsed werden.

Netminko
Netminko ist eine Python Library basierend auf Paramiko. Die Ziele dieser Library sind es, eine erfolgreiche SSH-Verbindung herzustellen, vereinfachte Ausführung von show und configuration Befehle und diese Funktionalitäten für eine breite Masse an Hersteller und Plattform zu unterstützen.

[10] https://www.ietf.org/rfc/rfc1157.txt
[11] https://www.iso.org/obp/ui/iso:std:iso-iec:7498:-1:ed-1:v2:en

NAPALM
Napalm steht für Network Automation and Programmability Abstraction Layer with Mulitvendor support und ist ein herstellerunabhängiges, plattformübergreifendes Open-Source-Projekt, das eine einheitliche API für Netzwerkgeräte bereitstellt. Es ist in Pyhton geschrieben und unterstützt die bekanntesten Automation Frameworks wie Ansible, Salt, StackStorm. Es bietet mehrere Methoden an, um sich bei den Netzwerkgeräten anzumelden, Konfiguration zu manipulieren oder Daten abzufragen.

Vendor specific API

Viele Hersteller wie Cisco, Palo Alto, Juniper etc. bieten eigene APIs für ihre Netzwerkgeräte an.

Vorteile:

- gute Unterstützung der Abfragen vom Hersteller

Nachteile:

- vom Hersteller abhängig
- für jeden Hersteller muss eine eigene API und somit andere Syntax verwendet werden

NETCONF

Das Network Configuration Protocol kurz NETCONF ist ein Netzwerkverwaltungsprotokoll und wurde von der IETF standardisiert. Es bietet Funktionen wie die Installation, Manipulation und Löschen der Konfiguration von Netzwerkgeräten. Diese Operationen bauen auf den simplen Remote Procedure Calls auf. Die Konfigurationsdaten und Protokollnachrichten werden mit XML kodiert.

Vorteile:

- weit verbreiteter Standard
- basiert auf ACID: Transaktion mit Rollback-Funktion vorhanden
- spezifische Daten durch Filter abfragen

Nachteile:

- Man muss sich mit RPC befassen. Dies macht das Programmieren aufwändiger und komplizierter.

RESTCONF

RESTCONF (Representational State Transfer Configuration Protocol)[12] ist eine Protokoll, das den REST-Prinzipien aus der Webentwicklung folgt, mit welchem man über HTTP Daten abfragen, manipulieren und löschen kann. RESTCONF wurde nicht konzipiert um NETCONF zu ersetzen, sondern um ein zusätzliches vereinfachtes Interface anzubieten. Es bietet die gleichen RPC-Operationen und Modellen (Yang) wie NETCONF und handelt jede Operation als eigene Transaktion ab.

Vorteile:

[12] https://tools.ietf.org/html/rfc8040

- auf REST bzw. HTTP basierendes Protokoll mit bekannten Error und Success Status
- bekannte HTTP Methoden verfügbar wie GET, PUT, POST etc.
- RESTCONF wird von vielen Herstellern unterstützt
- API antwortet mit JSON oder XML, das einfach zu parsen ist

Nachteile:

- es gibt noch nicht so viele Geräte, die RESTCONF unterstützen
- RESTCONF auf dem Gerät aktiviert und konfiguriert werden.

2.2 Anforderungsspezifikation

2.2.1 Graphen Anforderungen und deren Visualisierung

Bei der Analyse haben wir gesehen, dass wir verschiedene Algorithmen benötigen, um Netzwerkprobleme im Graphen abzubilden. In diesem Kapitel gehen wir darauf ein, was für Anforderungen an Graphen bestehen, um die erwähnten Algorithmen korrekt ausführen zu können.

Layer Unterteilung

Da einige Problemdomänen nicht den kompletten Graphen (das komplette Netzwerk), sondern nur ein Teil davon betreffen, wie z.B. Spanning Tree, muss es möglich sein, aus dem ganzen Graphen nur einen Teilgraph zu extrahieren und abzubilden. In unserem Anwendungsfall müssen also Nodes und Edges entsprechend dem Layer bzw. Sublayer selektiert werden können. Aus diesem Grund braucht es für jeden Node und Edge ein Attribut für den jeweiligen Layer.

Netzwerkverbindung

Eine Netzwerkverbindung ist normalerweise bidirektional. In einem Graphen sind ungerichtete Edges die Repräsentation von bidirektionalen Verbindungen. Eine Edge muss also als solche abgespeichert werden können.

Shortest Path

Um ein Shortest Path Algorithmus auszuführen zu können, braucht es grundsätzlich nur eine Edge zwischen zwei Nodes. Wir wollen jedoch mit dem Shortest Path auch direkt alternative Werte zum Pfad wie die Länge oder Latenzzeit ausrechnen. Dazu brauchen wir die entsprechenden Attribute auf den Edges, damit diese als Input für die Funktion verwendet werden kann. Das Attribut für en Pfad soll für unterschiedliche Anwendungen anders gewählt werden können.

Spanning Tree

Bei einem Spanning Tree kann es dazu kommen, dass ein ausgehender Port gesperrt wird, was in einer unidirektionalen Verbindung resultiert. Verbindungen, welche nur in ausgehende oder eingehende Richtung zeigen, werden durch gerichtete Edges repräsentiert. Deshalb muss der Graph zusätzlich zu ungerichteten auch gerichtete Edges unterstützen.

Zusätzlich werden gewichtete Edges für die Berechnung des Minimal Spanning Tree benötigt. Da Spanning Tree eine Technologie auf dem Datalinnk Layer ist, muss dieser als Sublayer dargestellt werde können, welcher separat ausgewählt werden kann.

Netzwerk Schwachstelle

Für die Identifikation einer potenziellen Schwachstelle im Netzwerk braucht es den Bridges- oder den Articulation Point Algorithmus. Diese suchen auch auf einem bestimmten Layer. Dies setzt voraus, das einzelne Layer ausgewählt werden können.

Um die potenzielle Schwachstelle zu verifizieren, muss für den entsprechenden Pfad auf dem tieferen Layer den Algorithmus erneut ausgeführt werden. Dazu muss es möglich sein, die beiden Nodes der Bridge im unteren Layer wieder identifizieren zu können. Dies kann mit einem Attribut wie beispielsweise der Hostname des Netzwerkgerätes umgesetzt werden.

Geräte Abbildung

Da es aus einem Netzwerkgerät mehrere Nodes ergibt, nämlich für die unterschiedlichen Endpunkte wie z.B. physischer Port, MAC Adresse und IP Adresse, ist es wichtig, dass jedem Node das physische Gerät, zu welchem der Endpunkt gehört, abgespeichert werden kann. Dies kann durch einen Node für das physische Gerät realisiert werden, welcher mit seinen Endpunkte verbunden wird. Da beim Zeichnen des Graphen zusätzliche Nodes für das physische Gerät einfügt werden, wird jeweils der Hostname beim Node als Attribut hinzufügt.

VLAN

Damit man die VLAN identifizieren kann, braucht man im Graphen auf den Ports sowie auf den Edges die Identifikation des VLAN's. Da über eine Verbindung mehrere VLANs (Trunk) transportiert werden können, braucht es für dessen Abbildung mehrere Attribute.

IP Identifizierung

Um beim Hinzufügen von einem Node überprüfen zu können, ob bereits eine IP vorhanden ist, muss es möglich sein, den Graphen nach einem bestehenden Node mit dem Attribut oder Namen der entsprechenden IP zu suchen.

Switch Konfigurationen

Damit eine Konfiguration eines Switch-Ports in einem Graphen abgespeichert werden kann, muss der Node mehrere frei wählbare Attribute abspeichern können. Um bei einem allfälligen Troubleshooting einen einfacheren Vergleich der Konfiguration zwischen dem Digital Twin und dem realen Netz zu gewährleisten, müssen nach den relevanten Attributen für ein Teilproblem gefiltert werde können.

Flow

Dass ein Flow und Algorithmen wie der Max Flow ausgeführt werden können, müssen gerichtete Edges im Graphen abgebildet werden können. Zusätzlich brauchen die Edges mehrere Attribute wie z.B. Kapazität und Fluss. Hier brauchen die Edges zwischen den Nodes eines Gerätes eine Kapazität, z.B. Verbindung zwischen MAC Adresse und IP Adresse. Diese wird nicht durch eine Kabelverbindung gesetzt, sondern durch das Abarbeiten der Queue im Prozessor des Netzwerkgerätes.

Zusammenfassung der Anforderungen

Hier ist eine Liste der zusammengefassten Anforderungen an den Graphen. Diese wurde anhand von den vorherig bestimmten Anforderungen ausgearbeitet.

- **Mehrere frei wählbare Attribute für Nodes und Edges (z.B. für Layer, Hostname)**
- **Aus dem kompletten Graphen (Layer) muss ein Teilgraph (Sublayer) anhand von Attributen selektiert werden können.**
- **Die Zuordnung von einem Attribut zu einem Eingabeparameter einer Graphenfunktion soll frei wählbar sein.**
- **Unterstützung von ungerichteten sowie gerichteten Edges.**
- **Unterschiedliche Attribute für verschiedene Layers.**
- **Nodes und Edges müssen anhand von Attributen auffindbar sein.**
- **Filterung nach Attributen.**
- **Endpunkte aus verschiedenen Layern sollen miteinander verknüpft werden, um deren Beziehung zueinander abbilden zu können.**

2.2.2 Anforderungen Netzwerkinformationsbeschaffung

In diesem Kapitel werden alle Anforderungen erläutert, die es für die Informationsbeschaffung der Netzwerkgeräte benötigt werden.

Abfragen der Gerätekonfiguration

Einerseits sollen spezifische Netzwerkinformationen wie MAC-, IP-Adressen, Port-Bandbreite des Geräts abgefragt werden können, anderseits auch die komplette Konfiguration, um diese mit einem vorherigen Zustand des Netzwerks zu vergleichen. Beim Troubleshooting werden zum Teil eigenartige Konfigurationen festgestellt. Ohne einen digitalen Zwilling mit der kompletten Netzwerkkonfiguration ist es nicht ersichtlich, ob diese bereits vor dem Problem vorhanden war oder nicht und dies der Grund der Störung sein könnte.

Abfragen von Status-Informationen

Der aktuelle Zustand, ob der Switch-Port, Link, Interface oder das Protokoll gerade aktiv ist, müssen abgefragt werden können. Ein Switch-Port könnte richtig konfiguriert sein, aber die Gegenstelle nicht. Dann kann kein Verkehr über den Link gesendet werden. Damit eine korrekte Abbildung des Zustands gemacht werden kann, müssen Status-Informationen abgerufen und gespeichert werden können.

Abfragen von Protokoll-Tabellen und -Neighbours

Die MAC-Address-, ARP- oder des Routing-Tabelle beinhalten wichtige Informationen über das Netzwerk und ihren angeschlossenen Geräten. Zusätzlich geben Routing-Protokolle spezifische Informationen zu deren Verbindungen untereinander. Diese Informationen müssen abgefragt werden können, da sie essentiell für die Speicherung des Netzwerks sind.

- **MAC-Address Table**: MAC-Adresse der Endgeräte herausfinden, welche am Switch angeschlossen sind.
- **ARP-Table**: Zuordnung von IP- zu MAC-Adresse der Endgeräte.
- **Routing Table**: Welche Routen vorhanden sind und von welchen Protokollen sie gelernt wurden.
- **OSPF-, BGP-Neighbours**: Die Nachbarn der jeweiligen Routing-Protokolle speichern

Abspeicherung Zugangsdaten

Da bei einem Netzwerk viele Netzwerkgeräte ausgelesen werden müssen und nicht alle die gleichen Zugangsdaten besitzen, müssen unterschiedliche Zugangsdaten hinterlegt werden können. Damit der User diese nicht alle von Hand für jedes einzelne Gerät eingeben muss, müssen diese Daten im System abgespeichert werden können. Zugangsdaten wie Passwörter sollen entsprechend verschlüsselt werden. Es soll auch möglich sein, dass mehrere Verbindungsmöglichkeiten abgespeichert werden können, z.B. Username mit Passwort oder ein SSH Key.

Wenn bereits eine zentrale Ablagestelle der Zugangsdaten existiert, ist es auch eine Möglichkeit diese in die Applikation heranzuziehen.

2.2.3 Anforderungen an die Netzwerk-Layer

Für jeden der sieben OSI-Schichten soll ein Layer erstellt werden können. Diese bauen aufeinander auf und sind voneinander abhängig. Zum Beispiel kann ohne eine physische Verbindung auf Layer 1 kein Routing auf dem übergeordneten Layer 3 stattfinden. Innerhalb dieser Layer werden die einzelnen Technologien als Sublayer abgebildet, welche vom übergeordneten Layer abhängig sind. Beispielsweise kann der Sublayer ÖSPFnur mit dem Layer 3 verwendet werden. Die Sublayer können auch voneinander abhängig sein, müssen aber nicht. Der Spanning-Tree kann beispielsweise auf einzelnen VLANs angewendet werden, dadurch hängt der Spanning-Tree vom jeweiligen VLAN ab, sofern VLAN im Netz verwendet wird. In diesem Kapitel werden die Anforderungen der ersten drei OSI-Schichten beleuchtet.

Layer 1

Auf dem untersten Layer erfolgt die Kommunikation über das physische Medium. Diese bildet die Grundlage für die restlichen sechs Layern.

Folgende Informationen sollen auf Layer 1 gespeichert werden:

- Hostname des Geräts
- Duplexmodus
- Verbindung und dessen Geschwindigkeit
- Drop Count
- Error Count
- Name des Endpunkts (z.B. Portname, Interface)
- Access-Port oder Trunk-Port
- Portstatus: On, Off, Disabled
- (Portfast, BPDU-Guard)

Link Aggregation Group (LAG) Link Aggregation beschreibt verschiedene Methoden zur Bündelung von mehreren Netzwerkverbindungen, um den Durchsatz und Redundanz einer Leitung zu erhöhen. Ein LAG kombiniert mehrere physische Ports zu einer logischen Gruppe. Dadurch wird der Netzwerkverkehr auf die Ports verteilt.

Es stellen sich folgende Fragen zur Link Aggregation:

- Welcher Typ von LAG z.B LACP oder Etherchannel soll benutzt werden?
- Welche Ports werden für die LAG verwendet?
- Was ist der aktuelle Status der LAG?
- Welche Geschwindigkeit haben die Links vor und nach der Bündelung?
- Welche Duplex-Einstellungen sind auf den Ports eingestellt?
- Handelt es sich um Trunk oder Access-Ports, die gebündelt werden sollen?

Link Aggregation Control Protocol (LACP) LACP ist ein IEEE Standard für eine Methode von LACP, welche von diversen Netzwerkgerätehersteller unterstützt wird.

Folgende Eigenschaften sind bei LACP zusätzlich zu beachten:

- Modi: active, passive, on, not configured
- Name und Nr. des Channel-Group

Demnach ergeben sich folgende Anforderungen an einen Sublayer:

- Informationen von einem überstehenden Layer sollen übernommen und zusammengefasst werden können.
- Mehrere Endpunkte (Nodes) und Verbindungen (Edges) sollen zu einem Node bzw. Edge zusammengeführt werden können, beispielsweise für die Bündelung der physikalischen Links.
- Ein Layer kann mehrere Sublayer besitzen, die voneinander abhängig sein können.

Layer 2

Da der zweite Layer ohne die Verbindungen des unterliegenden Layers nicht funktionieren kann, muss die Grundstruktur in diesem Layer übernommen werden. Wenn keine Verbindung auf Layer 1 vorhanden ist, gibt es auch keine Layer 2 Verbindung.

Folgende Informationen sind auf Layer 2 zu speichern:

- MAC-Adressen der Endpunkte von Layer 1
- Protokoll für die Kommunikation (Ethernet)

Folgende Anforderungen sind für den Sublayer von Nöten:

- Um die Endpunkte auf dem Layer 2 zu verwenden, müssen die Informationen aus dem unteren Layer 1 kopiert bzw. verbunden werden.

VLAN VLAN wird für die Unterteilung eines Netzes in kleiner Netzwerke verwendet. Es stellen sich dabei folgende Fragen zu VLANs:

- Sollen mehrere VLANs auf dem Switch definiert werden?
- Welche Nr. und Name wird für das VLAN gebraucht?
- Wie ist der Status des VLAN auf dem Switch?
- Soll der Switchport für ein (Access-Port) oder mehrere VLANs (Trunk-Port) verwendet werden?

Demnach ergeben sich folgende Anforderungen an einen Sublayer:

- Mehrere Sublayer können unabhängig voneinander sein. Ein Switch hat beispielsweise mehrere VLANs konfiguriert, welche sich nicht gegenseitig beeinflussen und autonom funktionieren.

- mehrere Attribute sollen auf einem Edge oder Node gespeichert werden können, z.B. VLAN-Name, -Status, -Nr.

VLAN Trunking Um mehrere VLANs über einen Link zu senden z.B. zwischen zwei Switches werden sogenannte Trunkports verwendet. Dabei stellen sich folgende Fragen zu VLAN Trunks:

- Soll ein eigenes native VLAN auf einem Trunk-Port verwendet werden?
- Welche Art von Trunking soll angewendet werden (DTP, 802.1q)?
- Was für ein Modus soll bei DTP verwendet werden (auto, desirable, trunk, access)?
- Sollen nur bestimmte VLANs auf dem Link erlaubt werden?

Daraus ergeben sich folgende Anforderungen an einen Sublayer:

- Eine Liste von gleichen Attributen sollen auf einem Node oder Edge definiert werden können. Beispielsweise eine Liste von VLANs, die auf dem Link erlaubt sind.
- mehrere Attribute müssen einem Node zugeordnet werden können.

STP STP wird verwendet, um Schleifen auf dem Data Link Layer zu vermeiden. Da oft in der Praxis STP innerhalb eines VLANs verwendet wird, ist der STP Layer ein Sublayer des jeweiligen VLANs, der Informationen von diesem erbt.

Demnach stellen sich folgende Fragen zu STP:

- Welches Spanning-Tree Protocol wird verwendet (STP, RSTP, PVST+, Rapid PVST+, VSTP, MSTP)?
- Welcher Switch ist die Root-Bridge?
- Welche Bridge-Priority besitzt der Switch?
- In welchem Status befinden sich die Switchports (discarding, learning, forwarding)?
- Welche Rolle hat der Port (Designated, Root, Alternativ)?
- Was sind die Kosten zur Root-Bridge?

Daraus ergeben sich folgende Anforderungen an einen Sublayer:

- Informationen der unteren Layern, wie die Kosten der Links, benötigt.
- Sublayer können mehrere Sublayer haben, beispielsweise wird auf dem VLAN 10 STP angewendet. Dieser STP-Sublayer ist vom Sublayer VLAN 10 abhängig.

Layer 3

Auch der dritte Layer ist von den beiden unterliegenden Layern abhängig. Wir beschränken uns bei dieser Arbeit auf das IP.

Folgende Informationen sind auf Layer 3 zu speichern:

- IP-Adressen der Endpunkte von Layer 1

- Protokoll für die Kommunikation (IP)
- Routing-Informationen

Routing Das Routing des Netzwerkverkehrs erfolgt über mehrere Teilnetze auf dem Network Layer.

Folgende Informationen werden benötigt:

- Next Hop ist anhand des Graphen ersichtlich
- Routing Tabelle mit Default und statischen Routen

Folgende Anforderungen sind für das Routing gegeben:

- Routing-Nachbarn und Next Hop Informationen müssen nicht als Attribut gespeichert werden, da dies bereits mit den Verbindungen der einzelnen Nodes innerhalb des Graphen gegeben ist.
- Es sollen die Default -und statische Routen gespeichert werden können.
- Routing-Protokoll spezifische Routen werden im jeweiligen Sublayer gesichert.

VLAN Routing Beim Einsatz von mehreren VLANs stellt sich die Frage, wie die einzelnen Clients mit einem anderen Subnetz kommunizieren können. Dazu gibt es verschiedene Arten, um Routing von VLANs zu ermöglichen.

- *Legacy Inter VLAN:* für jedes VLAN wird ein eigener Router-Port benötigt
- *Router-on-a-stick:* auf einem Router-Interface wird pro VLAN ein eigenes Sub-Interface erstellt.
- *L3 Switch Routing:* 1 VLAN-Interface wird pro VLAN definiert.

Daraus ergeben sich folgende Anforderungen an einen Sublayer:

- Ein Interface (Node) kann ein oder mehrere Sub-Interfaces besitzen.
- Es kann nicht immer die Informationen des untergeordneten Layers verwendet und verknüpft werden. Beispielsweise besitzt neben einem physischen Port auch ein Loopback oder ein virtuelles Interface eine IP-Adresse, welche in den unteren Layer nicht zum Einsatz kommen und demnach dort nicht abgebildet werden.

VPN VPN erlaubt eine Verbindung zu einem privaten Netz über ein öffentliches Netz wie das Internet aufzubauen. Benutzer können Daten mit dem verbundenem privaten Netzwerk austauschen. Es stellen sich folgende Fragen zu VPN:

- Welches Tunneling-Protokoll soll verwendet werden (IPsec, L2TP, GRE, VXLAN)?
- Wie soll der Tunnel verschlüsselt werden (IPsec, IKE, 3DES)?
- Wie soll die Authentifizierung gemacht werden (Username + Passwort oder Zertifikat)?
- Welche IP-Adresse und Subnetzmaske soll innerhalb des Tunnels für die Kommunikation verwendet werden?
- Welcher Hash-Algorithmus soll angewendet werden?

Daraus ergeben sich folgende Anforderungen an einen Sublayer:

- Ein Sublayer kann auch nicht komplett abhängig von unteren Layern sein. Das VPN interessiert es z.B. nicht, was in den unteren Layern für Technologien zum Einsatz kommen. Wichtig ist nur, dass die Verbindung nicht unterbrochen wird.

Open Shortest Path First (OSPF) OSPF ist ein Link State Routing Protokoll und gehört zu den Interior Gateway Protokollen. Jeder Router erstellt einen eigenen Graph mit den jeweiligen Distanzen zum Zielnetzwerk. Es stellen sich folgende Fragen zu OSPF:

- Ist es eine Multi oder Single Area OSPF Umgebung?
- Welche Netzwerke sollen in welche OSPF-Area propagiert werden?
- Welche Router-ID soll der OSPF-Router besitzen?
- Gibt es Interfaces, die keine Hello Pakete senden sollen (Passive Interfaces)?
- Was sind die OSPF-Neighbours des Routers, über welches Interface sind sie erreichbar und welche IP-Adresse besitzen sie?
- Wird die auto-cost-reference-bandwith auf den Routern verwendet? Sonst könnte eine falsche Routenberechnung ausgeführt worden sein.
- Was sind die Kosten zu den jeweiligen Routen?

Daraus ergeben sich folgende neue Anforderungen an einen Sublayer:

- Zusammengesetzte Attribute für die Angaben von Netzwerken mit ihren jeweiligen OSPF-Area.

2.2.4 Use Cases

Nachfolgend sind die wichtigen Funktionen im Fully Dressed und weitere Funktionen im Brief Format beschrieben.

Fully Dressed

Use Case 1	**Netzwerk abspeichern**
Priority:	1
Scope:	System-wide
Level:	User-goal
Primary Actor:	End user
Stakeholders and Interests: 1. **End user:** Komplette Netzwerkkonfiguration abgespeichert & Darstellung des Netzwerkes im Graphen	
Preconditions:	Zugangsdaten zu den gewünschten Netzwerkgeräten sind im System hinterlegt
Preconditions:	Neztwerkgeräte sind direkt über das Netzwerk oder Management-Netz erreichbar.
Postconditions:	Netzwerkgeräte, Verbindungen und Konfigurationen sind im persistentem Datenspeicher
Main Success Scenario: 1. System frägt Konfiguration des Gerätes ab 2. System erweitert Graphen des Netzwerks anhand der Konfiguration 3. System speichert neuen Graphen	

Special Requirements:

1. Netzwerkgeräte unterstützen die ausgewählte Verbindungsmethode.

Technology and Data Variations List:	Netzwerkgeräte Hersteller
Frequency of Occurrence:	häufig

Use Case 2	**Digital Twin manuell erweitern**
Priority:	1
Scope:	System-wide
Level:	User-goal
Primary Actor:	End user

Stakeholders and Interests:

1. **End user:** Netzwerk erweitern

Preconditions:	Ein Digitaler Twin eines Netzwerks existiert bereits.
Postconditions:	Hinzugefügte Komponente ist im Digital Twin im persistenten Datenspeicher abgespeichert.

Main Success Scenario:

1. User wählt aus, dass er manuell eine Komponente hinzufügen möchte.
2. User Wählt Typ der Komponente aus welche hinzugefügt werden soll.
3. User gibt benötigte Informationen und Attribute zur neuen Komponente ein.
4. System gibt eine Zusammenfassung der zu hinzufügenden Komponente aus
5. User bestätigt das Hinzufügen der zusammengefasten Komponente
6. System speichert neue Komponente in Datenbank und gibt eine Meldung an den User

Extensions or alternative flows:

5.b User wählt aus, dass er Informationen ändern möchte
6.b Prozess springt zu Schritt 4 im Main Success Scenario und wird von dort weitergeführt

---

Special Requirements:

1. Keine.

Frequency of Occurrence: häufig

Use Case 3	**Netzwerkgeräte Zugang abspeichern**
Priority:	1
Scope:	System-wide
Level:	User-goal
Primary Actor:	End user

Stakeholders and Interests:

1. **End user:** Zugang zum Netzwerkgerät abgespeichert.

Preconditions:	Keine.
Postconditions:	Zugangsdaten des Netzwerkgerätes sind im persistenten Datenspeicher gespeichert.

Main Success Scenario:

1. User wählt aus, dass er Zugangsdaten hinzufügen möchte.
2. User gibt Hostname oder IP-Adresse und Passwort oder ssh Key ein.
3. System speichert Zugangsdaten in Datenbank.
4. Vorgang kann pro gewünschtes Netzwerkgerät wiederholt werden.

Special Requirements:

1. Zugangsdaten zum Netzwerkgerät sind bekannt.

Frequency of Occurrence:	selten

Use Case 4	**Digital Twin oder Netzwerk darstellen**
Priority:	1
Scope:	System-wide
Level:	User-goal
Primary Actor:	End user

Stakeholders and Interests:

1. **End user:** Graphische Darstellung des Digital Twin oder des Netzwerks.

Preconditions:	Digital Twin oder Netzwerkverbindungen zu allen Geräten muss vorhanden sein.
Postconditions:	Graph wird graphisch ausgegeben.

Main Success Scenario:

1. User wählt aus, ob er den Digital Twin oder das Netzwerk graphisch darstellen möchte.
2. User wählt aus, welche Layer und Sublayer im Graphen enthalten sein sollen.
3. User wählt Graphen erstellen aus.
4. System zeigt den erstellten Graphen dar.

Frequency of Occurrence:	häufig

Use Case 5	**Verbindung zwischen zwei Endpunkten überprüfen**
Priority:	2
Scope:	System-wide
Level:	User-goal
Primary Actor:	End user

Stakeholders and Interests:

1. **End user:** Sicherstellen von Verbindung zwischen zwei Netzwerkendpunkten.

Preconditions:	Digital Twin muss vorhanden sein.
Postconditions:	System gibt aus, ob die ausgewählten Komponenten in Verbindung zueinander stehen oder nicht.

Main Success Scenario:

1. User wählt aus, dass er Verbindung zwischen zwei Endpunkten überprüfen möchte.
2. User wählt die zwei gewünschten Endpunkte aus.
3. System gibt zurück, ob die Endpunkte in Verbindung stehen oder nicht.

Technology and Data Variations List:

Frequency of Occurrence:	gelegentlich

Brief

Use Case 6	**Netzwerke vergleichen**
brief:	Zwei Netzwerke oder zwei verschiedene Stände des selben Netzwerks können miteinander verglichen werden, und die Differenzen werden aufgezeigt. Vergleich der beiden Netzwerk Graphen.

Use Case 7	**Potenzielle Schwachstellen im Netzwerk finden**
brief:	Der User wählt das gewünschte Teilnetzwerk aus und gibt an, ob er Verbindungen oder Netzwerkgeräte als potenzielle Schwachstellen finden möchte. Das System gibt eine Liste der Resultate zurück.

Use Case 8	**Maximalen Netzwerkfluss berechnen**
brief:	Der User wählt aus dem Digital Twin eine Verbindung zwischen zwei Endpunkten auf dem Routing Layer aus. Das System generiert den Shortest Path der Verbindung, welche dem Routing entspricht. Anschliessend berechnet das System anhand der Verbindungen im Graphen den maximalen Netzwerkfluss zwischen den beiden Endpunkten.

Use Case 9	**Script in Digitalem Twin testen**
brief:	Man kann ein Script, welches man im produktiven Netzwerk einsetzen möchte, zuerst auf dem Digital Twin laufen lassen und testen. Das Verhalten des Netzwerkes wird prognostiziert, so dass der User mögliches Fehlverhalten identifizieren kann.

Use Case 10	**Netzwerk aus Digital Twin erstellen**
brief:	Aus dem Digitalen Twin kann wieder ein Netzwerk erzeugt werden. Die gespeicherten Informationen im Digital Twin werden dazu benutzt, um ein virtuelles Netzwerk zu erzeugen.

2.2.5 Use Case Diagramm

Nachfolgend sind die einzelnen Use Cases der Applikation mit dessen Aktoren ersichtlich.

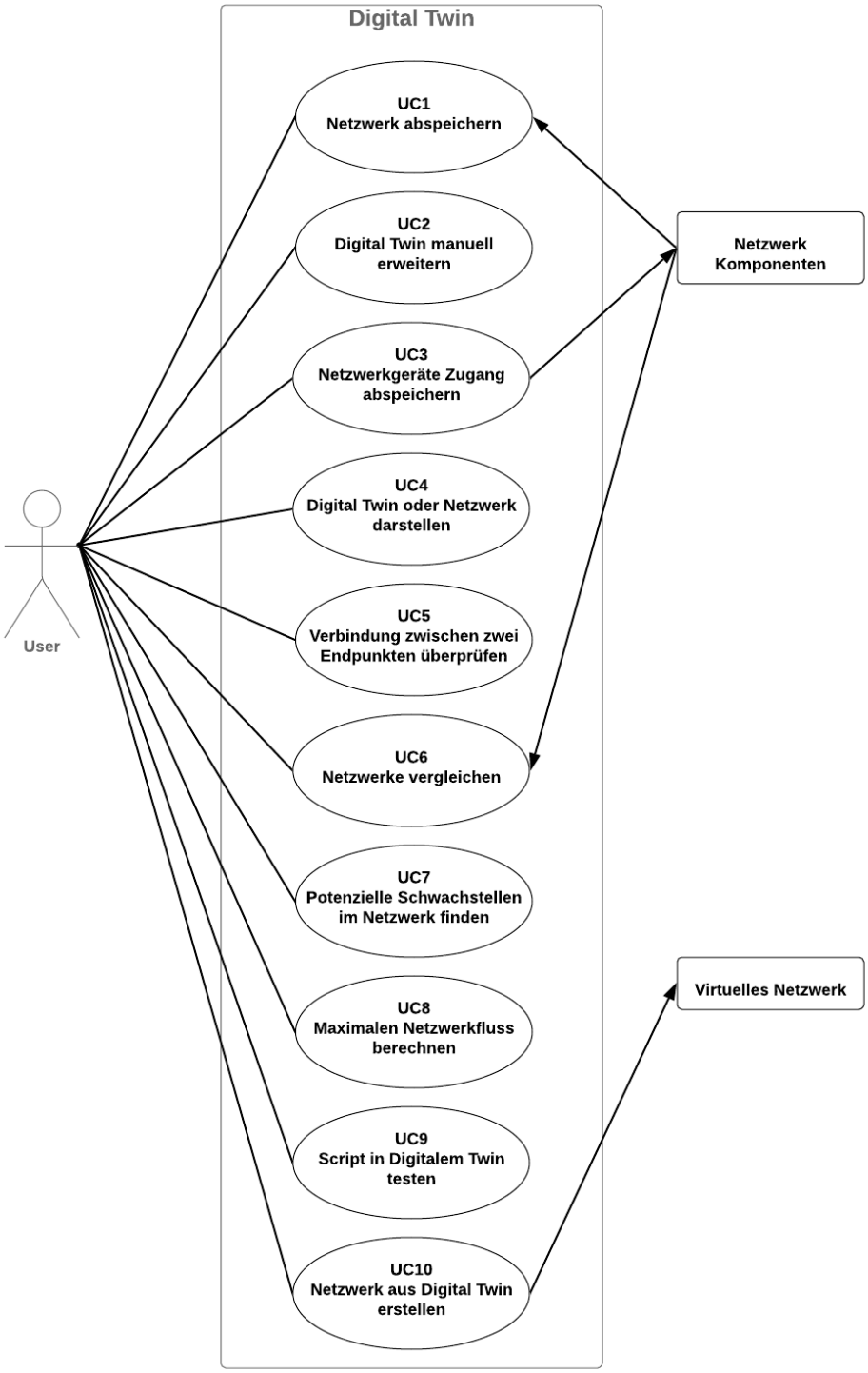


Abbildung 2.21: Use Case Diagram

2.2.6 Nicht funktionale Anforderungen

Anforderung	**Beschreibung**
Erweiterbarkeit	Die Applikation soll so entworfen werden, dass in Zukunft weitere Funktionalität, welche in Netzwerken dazu kommen, auch erweitert werden können.
Skalierbarkeit	Da Netzwerke in der Grösse sehr unterschiedlich sein können, soll das System gut skalieren.
Sicherheit	Gespeicherte Daten dürfen nur von Berechtigten Benutzern eingesehen und bearbeitet werden.
Verarbeitungszeit	Das System soll kleinere Operationen in kurzer Zeit bearbeiten. Bei grösseren Operationen wie das Klonen des Netzwerks muss dem Benutzer klar angezeigt werden, dass das System beschäftigt ist Damit für den Benutzer kein Gefühl entsteht, dass das System eingefroren ist.
Wartbarkeit	Der Code soll einfach gewartet werden können. Dazu soll er einfach zu verstehen sein, konsistent und gut dokumentiert.

2.3 Architektur

In diesem Kapitel wird die gewählte Architektur des Digital Twin gezeigt, beschrieben und erklärt, warum man sich für die einzelnen Technologien entschieden hat. Der Aufbau dieses Kapitels wurde von der Architektur Vorlage arc42[13] inspiriert.

2.3.1 Stakeholders

Nachfolgend werden alle direkt an der Studienarbeit beteiligten Personen mit ihren Funktionen aufgelistet.

Rollen	**Namen**	**Beschreibung**
Supervisors	Beat Stettler & Marc Sommerhalder	Unterstützung konzeptionellen und technischen Fragen sowie Bewertung der Arbeit
Frontend Leader	Patrick Lehmann	Anleitung zur Umsetzung
Deployment Leader	Dario Caluzi	Anleitung zur Umsetzung
Doku Leader, Designer & UX Leader	Patrick Lehmann	Anleitung zur Umsetzung
Backend Leader	Dario Caluzi	Anleitung zur Umsetzung

2.3.2 Organisatorische Einschränkungen

Nachfolgend sind Einschränkungen erwähnt, welche die Studienarbeit beeinflussen.

Name	**Beschreibung**
Zeit	Das Projekt muss innerhalb des Herbstsemesters 2019 abgeschlossen werden.
Budget	Das Projekt sollte innerhalb 480 Arbeitsstunden Aufwand abgeschlossen werden.

2.3.3 Lösungsstrategie

Nachfolgend werden die Qualitätsziele zu den einzelnen Themen wie Skalierbarkeit, Wartbarkeit usw. definiert, die in dieser Arbeit erreicht werden sollen.

[13] https://arc42.de/

Quality Goal	**Scenario**	**Solution**
Skalierbarkeit	Grosses Netzwerk, welches viele Geräte umfasst	Neo4j Datenbank
Wartbarkeit	Code Qualität	Code Reviews, Linting, Style Guides
Deployment Geschwindigkeit	Deployment des Code	Gitlab CI
Kosten	Kosten tief behalten	Azure Cloud auto scheduling VM
Security	Keine unerlaubte Datenzugriffe	Neo4j
Privatsphäre	Verschlüsselung der Daten beim Transfer	https, bolt(Secure Neo4j Protokoll)

2.3.4 Baustein-Ansicht

In diesem Kapitel wird eine Übersicht über das System und die einzelnen Schnittstellen und Komponenten aufgezeigt.

Whitebox Übersicht des Systems

In Abbildung 2.22 wird eine Übersicht von den einzelnen Schnittstellen zum Digital Twin dargestellt.

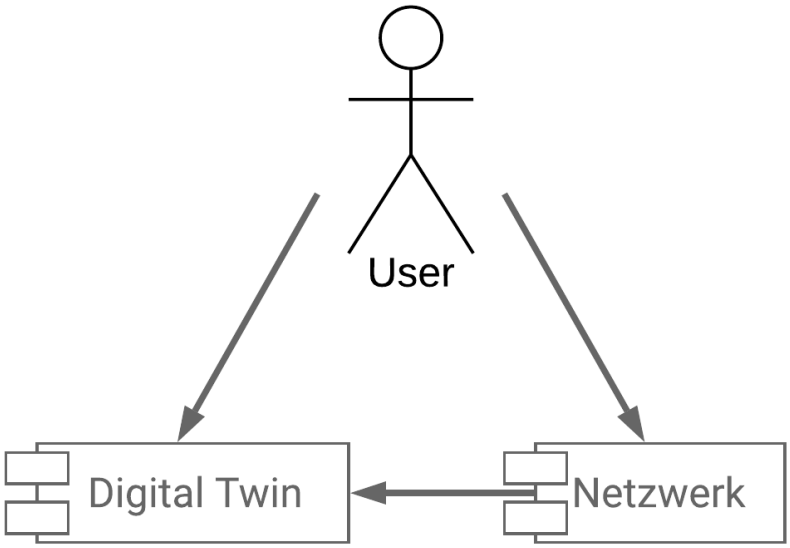


Abbildung 2.22: Whitebox System Übersicht

Name	**Verantwortung**
Digital Twin	Ermöglicht das Abspeichern und Bearbeiten eines Netzwerk in einem digitalen Twin.
Netzwerk	Das Netzwerk, welches die Basis für den Digital Twin liefert.

White Box Ansicht des *Digital Twin*

In Abbildung 2.23 werden die einzelnen Komponenten des Digital Twin gezeigt.

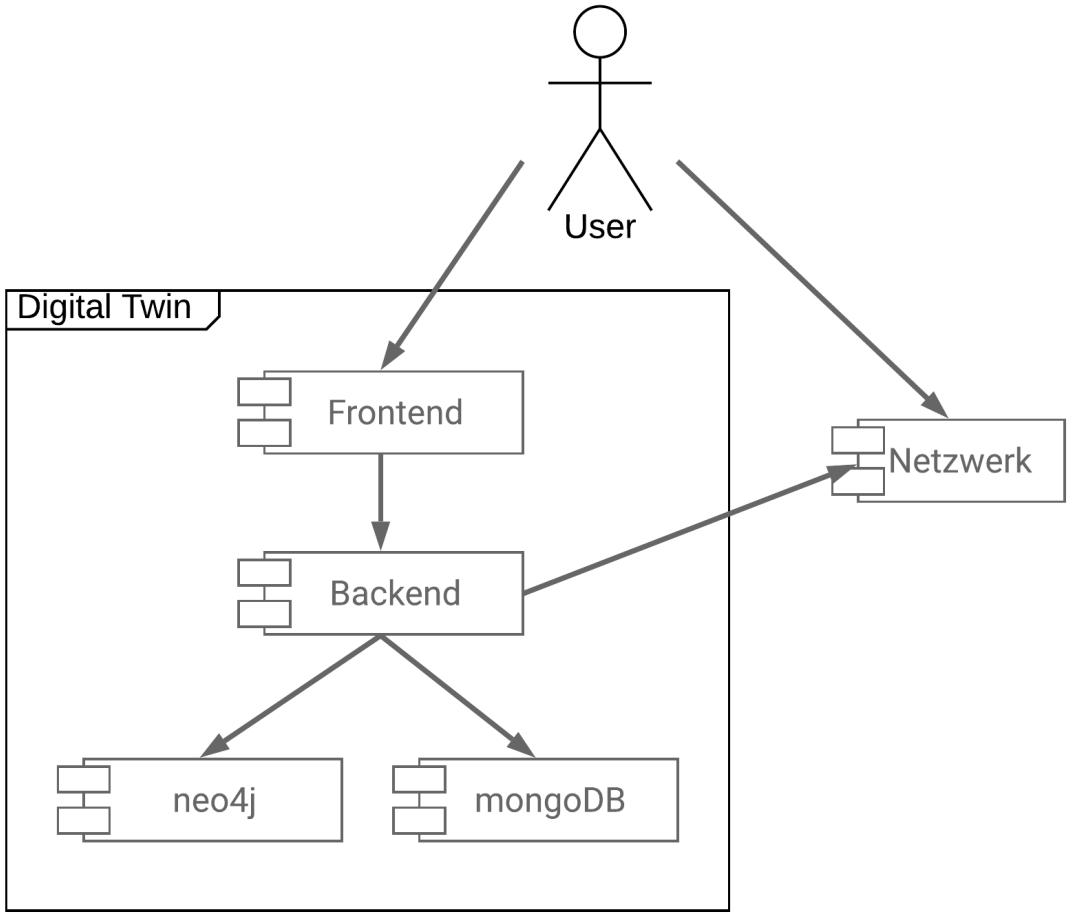


Abbildung 2.23: Whitebox Digital Twin

Motivation:
Das Ziel ist eine strikte Trennung zwischen Frontend, Backend und Persistenz.

Komponenten des Digital Twin

Name	**Verantwortung**
Frontend	User Interaktion
Backend	Programm Logik
Neo4j	Graphendatenbank, um Digital Twin abzuspeichern
MongoDB	Datenbank, um Netzwerk Zugangsdaten abzuspeichern

Black Box Beschreibung des *Frontend*

Verantwortung	Eine webbasierte Oberfläche für den User bieten, um mit der Applikation zu interagieren.
Interface	Web Browser
Location	Link zum Source Code

Black Box Beschreibung des *Backend*

Verantwortung	Anbieten der Programm Logic durch Funktionen, welche für das Frontend erreichbar sind.
Interface	API
Location	Link zum Source Code

Black Box Beschreibung von *Neo4j*

Verantwortung	Graphdatenbank, um den Graph des Digital Twin abzuspeichern.
Interface	Bolt
Location	Azure

Black Box Beschreibung von *mongoDB*

Verantwortung	noSQL Datenbank, um die Zugangsdaten für alle Netzwerkgeräte abzuspeichern.
Interface	API
Location	Azure

2.3.5 Sequenzdiagramme

Digital Twin erstellen

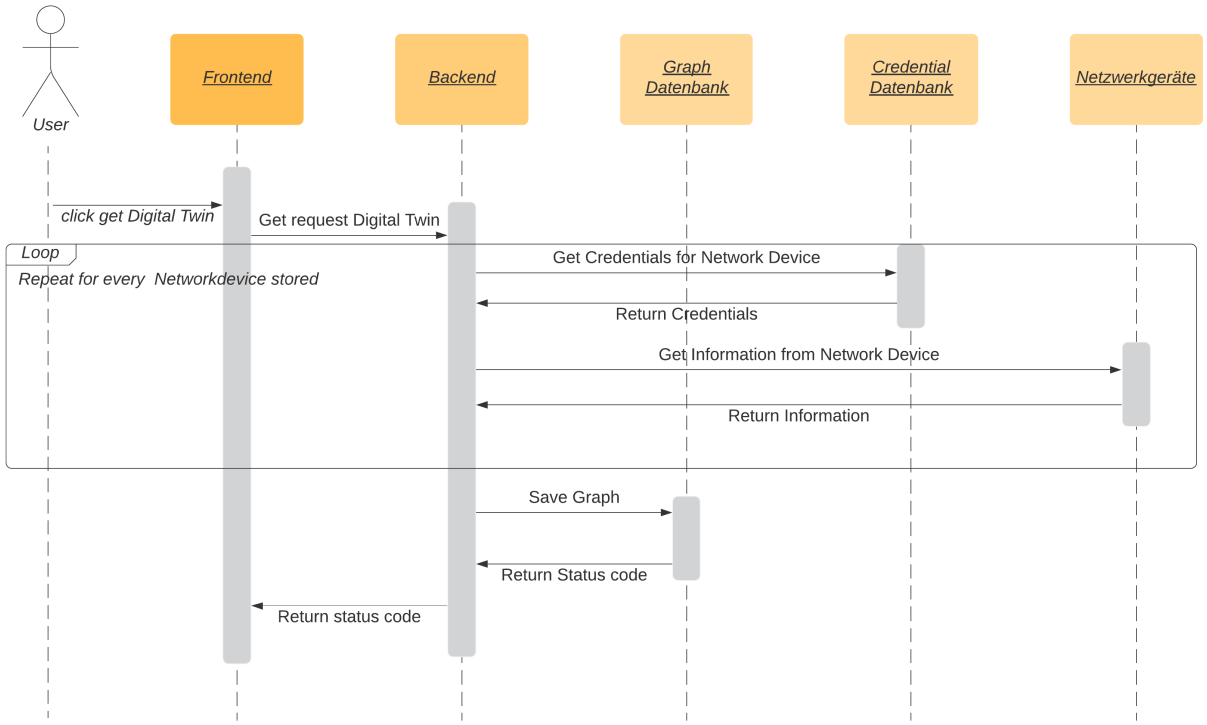


Abbildung 2.24: Sequenz Diagramm Digital Twin erstellen

Der Benutzer klickt im Frontend auf einen Button, welchen einen GET-Request auf das Backend absetzt. Das Backend holt sich als allererstes die Credentials der einzelnen Netzwerkgeräten. Anschliessend werden die Geräte abgefragt und in die Graphdatenbank gespeichert. Der Benutzer erhält eine Nachricht, das der Digital Twin erstellt wurde.

Anzeigen des Digital Twin

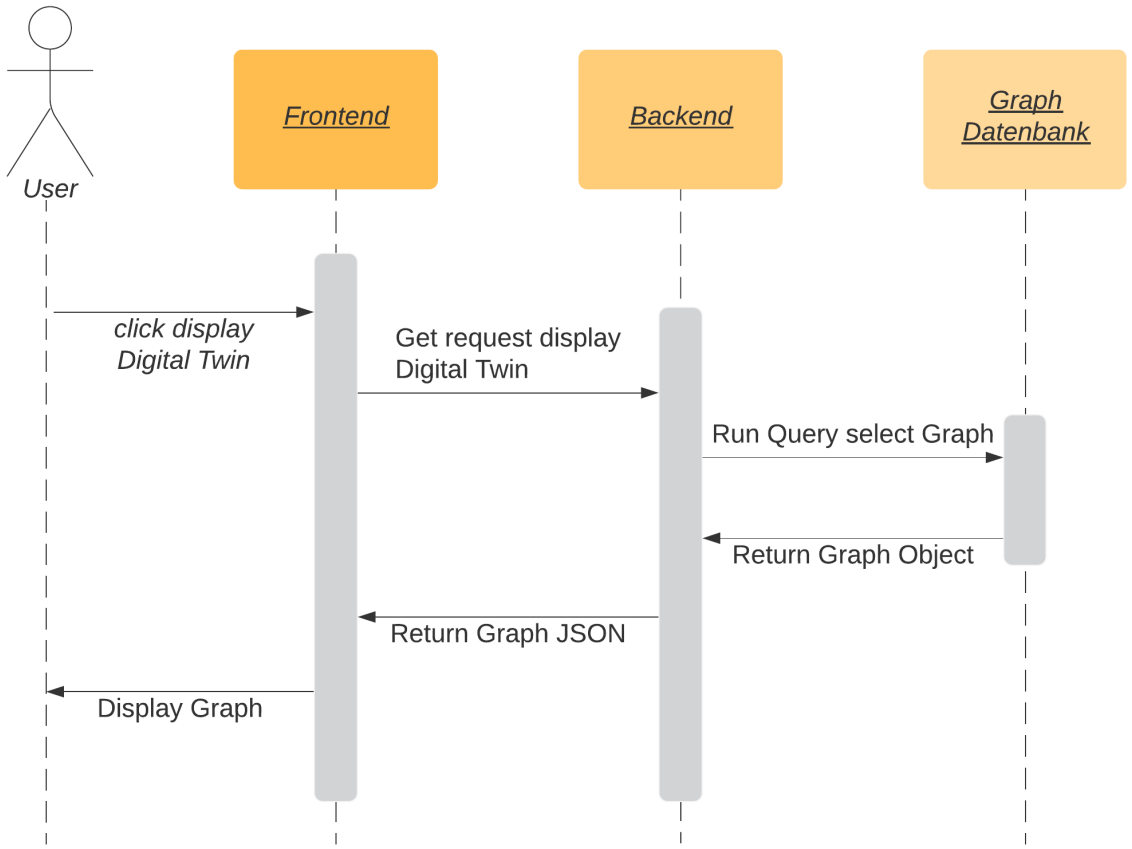


Abbildung 2.25: Sequenz Diagramm Digital Twin darstellen

Der Benutzer möchte den Digital Twin des Netzwerks sehen und drückt auf den entsprechenden Button. Dabei wird ein Get-Request an das Backend geschickt. Das Backend setzt wiederum eine Abfrage mit den gewünschten Informationen an die Graphdatenbank. Die von der Graphendatenbank erhaltenen Daten werden als JSON an das Frontend geschickt, wo der Graph für den Benutzer dargestellt wird.

Erstellen von Netzwerk aus Digital Twin

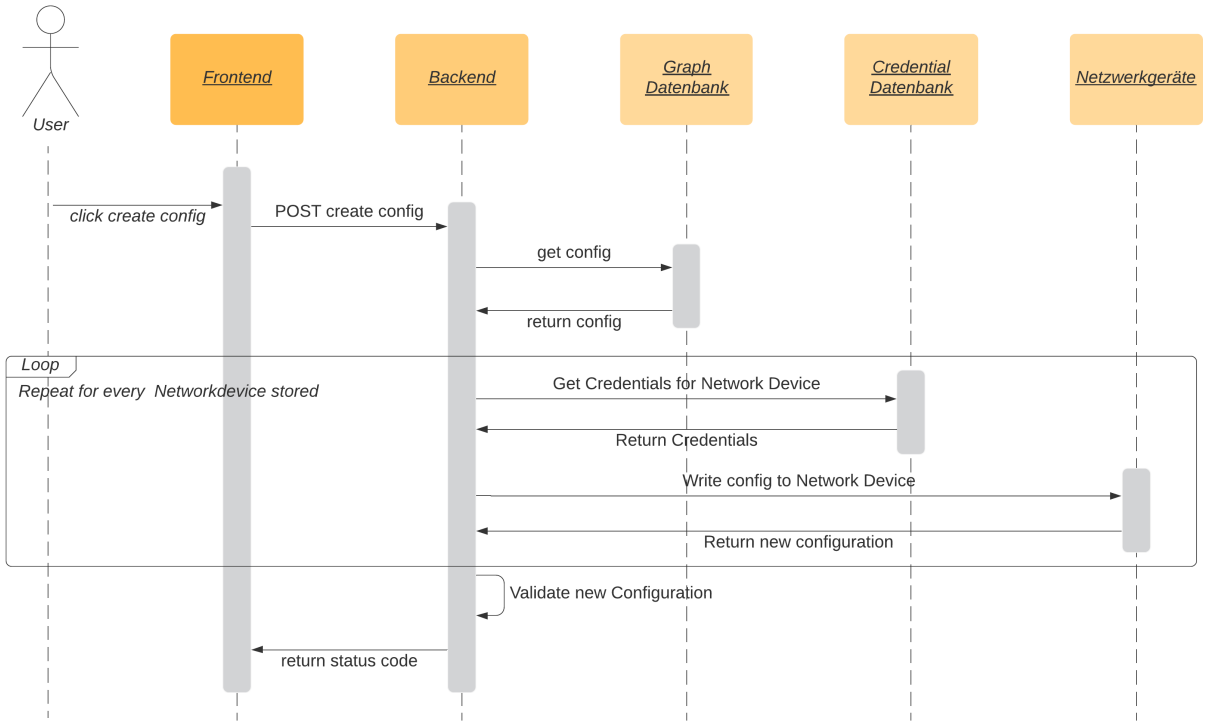


Abbildung 2.26: Sequenz Diagramm Netzwerk aus Digital Twin erstellen

Der Benutzer möchte eine Konfigurationsänderung, welche er im Digital Twin getätigt und getestet hat auf das produktive Netzwerk spielen. Dazu klickt er auf den Button *create config* und löst eine POST-Abfrage an das Backend aus. Das Backend holt sich die Konfiguration, die in der Graphdatenbank abgelegt ist, und die Credentials für jedes Netzwerkgerät, das geändert werden soll. Anschliessend schickt das Backend einzeln die neue Konfiguration an die Netzwerkgeräte. Nachdem überprüft wurde, ob die Änderungen erfolgreich durchgeführt wurden, schickt das Backend die Antwort an das Frontend, in welchem eine Meldung dem Benutzer angezeigt wird, ob die Konfigurationsänderung erfolgreich war oder nicht.

Netzwerk und Digital Twin vergleichen

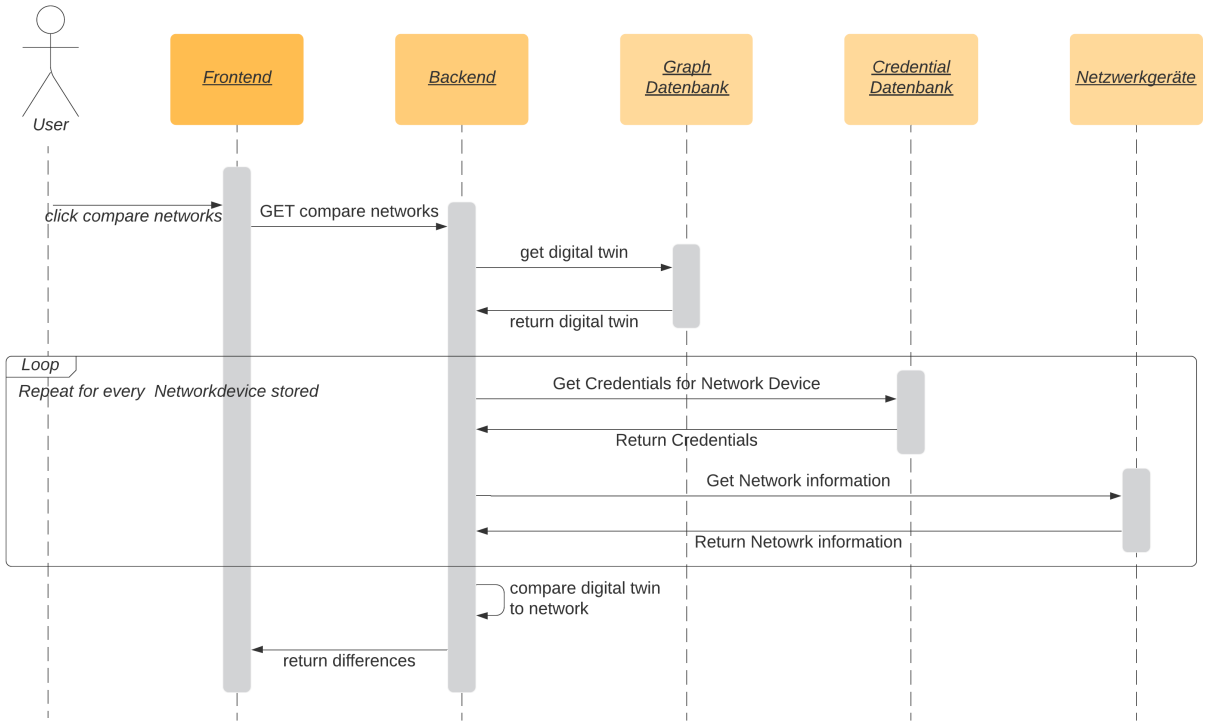


Abbildung 2.27: Sequenz Diagramm Netzwerk und Digital Twin vergleichen

Der Benutzer drückt auf den Button *compare networks* und möchte den Digital Twin mit dem produktiven Netzwerk vergleichen. Durch diese Aktion wird ein GET-Request an das Backend geschickt. Das Backend holt sich die Daten aud der Graphdatenbank. Anschliessend holt er sich die Zugangsdaten der Netzwerkgeräte. Mit den Zugangsdaten fragt das Backend die Netzwerkgeräte ab und vergleicht diese Informationen mit denen aus der Graphendatenbank. Das Backend schickt anschliessend die Differenzen an das Frontend zurück.

2.3.6 Domain Modell

Weil wir für die persistente Datenspeicherung die Graphdatenbank Neo4j verwenden, welche schemalos ist, gibt es kein traditionelles, statisches Domain Model.

Eine Graphdatenbank besteht aus Nodes und Edges, welche beide Daten in Form von Attributen abspeichern können. Die Nodes stehen in Verbindung miteinander, welche durch die Edges repräsentiert werden. So können analog zu Schema Datenbanken Verknüpfungen zwischen den Daten erstellt werden.

Dummy Netzwerk

Als Basis für das Domain Modell wurde im Cisco Packet Tracer ein Beispiel-Netzwerk erstellt. Darin wurden die Technologien STP mit VLAN, LACP und OSPF auf Routern, L2- und L3-Switches realisiert.

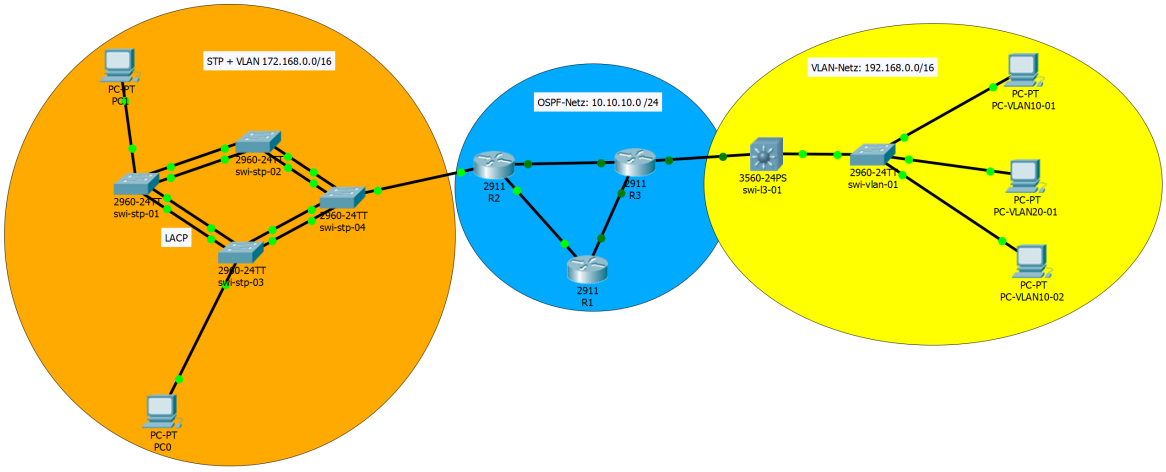


Abbildung 2.28: Dummy Netzwerk

Nodes

Für jeden Layer und Sublayer im Graphen werden wir einen spezifischen Typ von Node erstellen, bei welchem vordefinierte Attribute vorhanden sind. Dies unterstützt die klare Auftrennung der einzelnen Netzwerklayer.

Die jeweiligen Typen von Nodes werden in Neo4j dynamisch mit jeweils einer Farbe hinterlegt, damit sie visuell einfach erkennbar sind.

Die folgenden Typen von Nodes haben wir: [14]

- Device
- Physical
- LinkAggregation
- VLANInterface

[14] das sind die effektiven Namen, die in der Neo4J Datenbank hinterlegt sind.

- Loopback
- DataLink
- VLAN
- PVST (pro VLAN)
- Network
- OSPF

Edges

Es gibt drei verschiedene Arten von Edges. Eine Art Edge sind Verbindungen zwischen Netzwerklayern, welche zum selben physischen Gerät gehören.

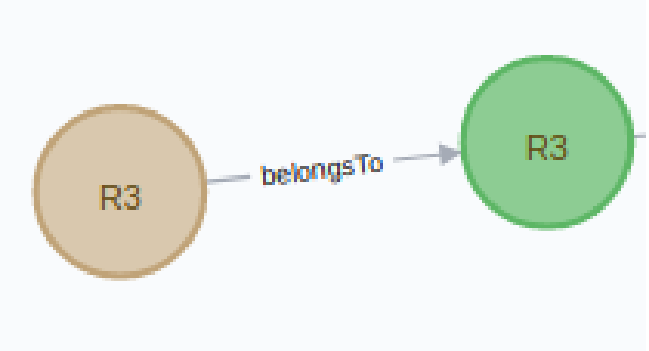


Abbildung 2.29: belongsTo Verbindung

In Abbildung 2.29 ist eine Verbindung von einem Port und einer MAC Adresse des selben Netzwerkgerätes ersichtlich.

Die zweite Art Edges sind Verbindungen, welche eine Netzwerkkomponente mit einer anderen verbindet. Diese Verbindungen bekommen noch das Attribut Bandwidth dazu, damit die Geschwindigkeit der Verbindung gespeichert werden kann, sowie den jeweiligen Layer, in dem die Verbindung besteht.

Abbildung 2.30: connectedTo Verbindung

In Abbildung 2.30 ist eine Verbindung von zwei Ports auf dem Physical Layer zu sehen.

Die letzte Art von Edges ist eine Verbindung von einem Interface, sei es physisch oder logisch, zu einer Netzwerkkomponente.

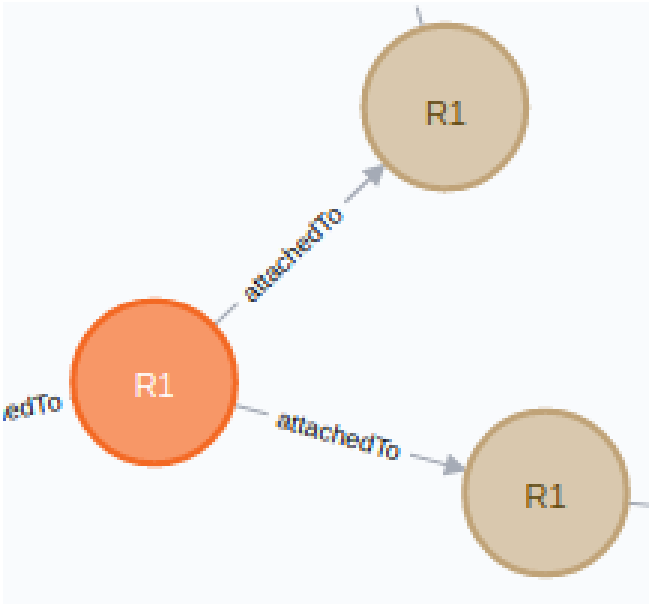


Abbildung 2.31: attachedTo Verbindung

In Abbildung 2.31 ist ein Device Node (orange R1) und zwei Physical Ports Nodes (braune Nodes), welche jeweils durch eine attachedTo Beziehung dem Device zugewiesen werden.

Verbindungsart

Bei Neo4j ist es nicht möglich eine Verbindung bidirektional anzulegen. Dies ist aber für uns kein Problem, da Neo4j bei Abfragen des Graphen und bei Algorithmen die Richtung der Verbindung ignorieren kann. So werden die Verbindungen jeweils in eine Richtung erstellt. In die andere Richtung braucht es keine Verbindung, da es sonst zu Schlaufen resultieren würde.

In diesem online Artikel wird diese Art der Verbindung empfohlen, aufgrund dessen, haben wir uns für diese Variante entschieden. [15]

[15] https://graphaware.com/neo4j/2013/10/11/neo4j-bidirectional-relationships.html

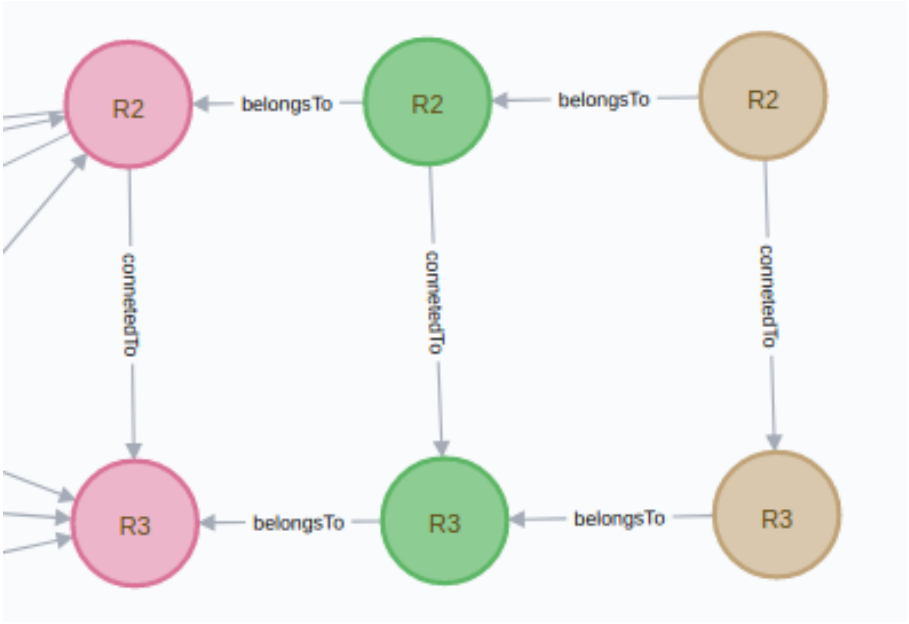


Abbildung 2.32: Kleiner Beispielgraph

In der Abbildung 2.32 sehen wir ein sehr kleines Beispiel eines Graphen zur Veranschaulichung der Verbindungen und Nodes. Das Beispiel sind zwei Router, welche miteinander verbunden sind.

Die braunen Nodes sind jeweils die physischen Interfaces, die grünen sind MAC- und die roten sind IP-Adressen. Die Nodes mit dem Namen R2 gehören alle zum Router R2. Diese Nodes bzw. Interfaces sind deshalb mit belongsTo Verbindungen miteinander verknüpft.

Die Verbindungen zwischen den Nodes mit der gleichen Farbe sind Verbindungen auf dem selben Layer von unterschiedlichen Netzwerkgeräten. Darum sind dies Verbindungen vom Typ connectedTo.

Bei den Nodes im Network Layer sind noch ein- und ausgehende Verbindungen zu anderen IP Adressen im Network Layer zu sehen. Damit es übersichtlich bleibt, wurden diese Verbindungen und Nodes in der Abbildung abgeschnitten.

2.3.7 Datenmodell

In diesem Kapitel beschreiben wir, welche Informationen zu welchem Layer respektive Typ Node benötigt werden. Anschliessend gibt es jeweils ein Beispiel von einem Node des jeweiligen Typ in Cypher, was die Abfragesprache für Neo4j ist.

Device

- Hostname

Beispiel Node: (:Device Hostname: 'R1')

Physical

- Hostname
- Port
- DuplexMode
- Speed

Beispiel Node: (:Physical Hostname: 'R1', Port: 'GigabitEthernet0/1', DuplexMode: 'auto', Speed: 'auto')

Link Aggregation

- Hostname
- Port
- Protokoll

Beispiel Node: (:LinkAggregation Hostname: 'R2', Port: 'GigabitEthernet0/1', Protocol: 'LACP')

VLANInterface

- Port
- VLAN ID
- Hostname
- Encapsulation

Beispiel Node: (:VLANInterface Port: 'vlan10', VLANID: 10 , Hostname: 'swi-l3-01' , Encapsulation: 'ARPA')

Loopback

- Port
- Hostname

Beispiel Node: (:Loopback Hostname: 'swi-l3-01', Port: 'Loopback0')

Data Link

- Hostname
- MAC Adresse

Beispiel Node: (:DataLink Hostname: 'R1', MAC: '000B.BE4A.3202')

VLAN

- Hostname
- VLAN ID
- Encapsulation

Beispiel Node: (:VLAN VLANID: 30 , Hostname: 'R2' , Encapsulation: 'dot1Q');

STP

- Bridge ID
- Rootbridge
- Portstatus
- STP Modus

Beispiel Node: (:stp BridgeID : '32769.000B.BE4A.3203', STPMode: 'pvst', RootBridge: False, PortStatus: 'Designated Port')

PVST (per VLAN Spanning Tree)

Pro VLAN kann es einen eigenen STP geben, diese bekommen jeweils einen eigenen Typ Node, damit es etwas übersichtlicher ist.

- Hostname
- VLAN ID
- Bridge ID
- Rootbridge
- Priority

Beispiel Node: (:PVST Hostname: 'swi-l3-01', VLANID: 10, RootBridge: False, BridgeID: '00D0.D3AD.D277', Prority: 32778) –> STP Node aufbauend auf dem VLAN 10.

Network

- Hostname
- IP Adresse
- Subnetzmaske

Beispiel Node: (:Network Hostname: 'R2', IP: '10.10.10.2', Subnetmask: '255.255.255.252')

OSPF

- Hostname
- ID
- Area
- Referenz Bandbreite
- Interface Bandbreite
- Authentication
- Hello Intervall
- Dead Intervall

Beispiel Node: (:OSPF Hostname: 'R2', ID: '2.2.2.2', Area: 0, ReferenceBandwidthth: 100, Interface-Bandwidthth: 100, Authentication: 'None', HelloIntervall: 10, DeadIntervall: 10);

2.3.8 Erfahrungen durch Erstellung der Dummy Datenbank

Damit wir das Domain und Datenmodell erstellen konnten, haben wir auf Basis des Dummy Netz 2.3.6 manuell eine Datenbank, wie wir sie uns vorstellen, anhand unserer vorgängigen Analyse erstellt. Die Kapitel Domain Modell 2.3.6 und Daten Modell 2.3.7 mussten mehrfach überarbeitet werden, da man durch die Evaluation neue Erfahrungen gesammelt hatte, die für die Umsetzung des Digital Twin notwendig waren.

In diesem Kapitel halten wir Erkenntnisse, die wir durch die Erstellung der Dummy Datenbank und Probleme, auf welche wir gestossen sind, fest.

Virtuelle Interfaces

Bei der Erstellung der Datenbank ist uns aufgefallen, dass für OSPF und VLAN, zusätzlich zu den physischen Ports, auch virtuelle Ports benötigen. Aus diesem Grund haben wir und dazu entschlossen, die beiden Nodetypen Loopback und VLANInterface dem Datenmodell hinzuzufügen. Somit können diese auch erfasst werden, jedoch sie sind klar separierbar von den physischen Ports.

Verbindungen einzelner Ports

Bei der Überprüfung der Datenbank ist herausgestochen, dass es schwierig ist, die Nodes auf Layer 1 korrekt zuzuweisen, da sie keine Verbindungen ausser zu der entsprechenden MAC Adresse haben. Damit es etwas übersichtlicher ist, haben wir uns dazu entschieden, jeweils einen Node vom Typ Device für jede Netzwerkkomponente hinzuzufügen. Daraus ist eine neue Art von Verbindung (Edge) entstanden: die attachedTo Verbindung. Dies hat zur Folge, dass die physischen und virtuellen Ports mit dem entsprechenden Device Node verbunden werden. Nun ist es beispielsweise möglich, nach dem Device Node zu suchen und man sieht direkt, welche Ports auf dem Gerät konfiguriert sind.

Spanning Tree pro VLAN

Die ursprüngliche Idee Nodes von zwei Typen zu kombinieren, um Spanning Tree für jedes einzelne VLAN abzuspeichern, hat nicht gut funktioniert, da im Nachhinein nicht gut nach den entsprechenden Typen abgefragt werden konnte. Wenn man beispielsweise nach Nodes vom Typ VLAN sucht, hat man auch Nodes mit dem zusätzlichen Typ STP gefunden. Um dies zu vereinfachen, haben wir uns dazu entschieden, für jeden Spanning Tree auf einem VLAN einen neuen Typ Node zu erstellen, welcher dann aber einzigartig ist. Dadurch hat man sowohl eine eindeutige Zuweisung zum Spanning Tree Protokoll wie auch zum entsprechenden VLAN, ohne dass bei einer Suche nach einem VLAN die STP Nodes auch erscheinen.

Diese Variante ist jedoch auch nicht ausreichend, weil so der Graph und die Typen an Nodes nicht generisch, sondern direkt vom Netzwerk abhängig sind und unterschiedlich sein können. Deshalb werden per VLAN ein Spanning Tree Node vom Typ PVST erstellt und jeweils mit der VLAN ID als Attribut versehen. So gibt es nur einen Typ, welcher generisch durch das Attribut dem korrekten VLAN zugewiesen werden kann.

Verbindungen von VLAN und STP
Da der Spanning Tree auf dem VLAN aufbaut, haben wir uns dafür entschieden, die STPperVLAN Nodes direkt mit dem VLAN Nodes zu verbinden, damit dies klar ersichtlich ist. Jedoch ist daraus das Problem entstanden, dass ein Spanning Tree von einem VLAN, welches auf keinem physischen oder virtuellen Port des Switches konfiguriert ist, nicht im Graphen dargestellt werden kann.

VLAN Node bei VLAN Interface

Bei einer Suche nach dem Typ VLAN ist auf dem VLAN Interface nicht ersichtlich, dass es dazugehört. Deshalb wird eine weitere Verbindung vom DataLink Layer respektive der MAC Adresse des VLAN Interfaces zum entsprechenden VLAN Node erstellt. Danach ist die Verbindung vom Interface zum VLAN klar ersichtlich.

Verbindung von VLAN zu VLAN Gateway IP

Die Verbindung vom VLAN zu der IP Adresse des VLAN Gateways ist nur auf dem Node-Stack des VLAN Interfaces ersichtlich. Da aber alle anderen VLAN Mitglieder des VLAN's auch über diesen Gateway kommunizieren können, wird eine Verbindung des VLAN Nodes von der entsprechenden VLAN ID zur IP Adresse des VLAN Gateways erstellt.

2.3.9 Kontrollabfragen Dummy DB

Um die Funktionsweise der manuell erstellten Neo4j Datenbank zu überprüfen, haben wir diverse Abfragen auf der Datenbank gemacht und dabei kontrolliert, ob die Ergebnisse dem des Dummy Netzwerks entspricht, welches im Cisco Packet Tracer erstellt und als Basis der Datenbank verwendet wurde.

Verbindung von zwei Geräten

Um zu kontrollieren, ob zwei Geräte (Nodes vom Typ Device) miteinander verbunden sind, haben wir folgende Cypher Query abgesetzt:
MATCH (start:Device Hostname:'R1'), (end:Device Hostname:'R2'), p = shortestPath((start)-[*]-(end)) return p

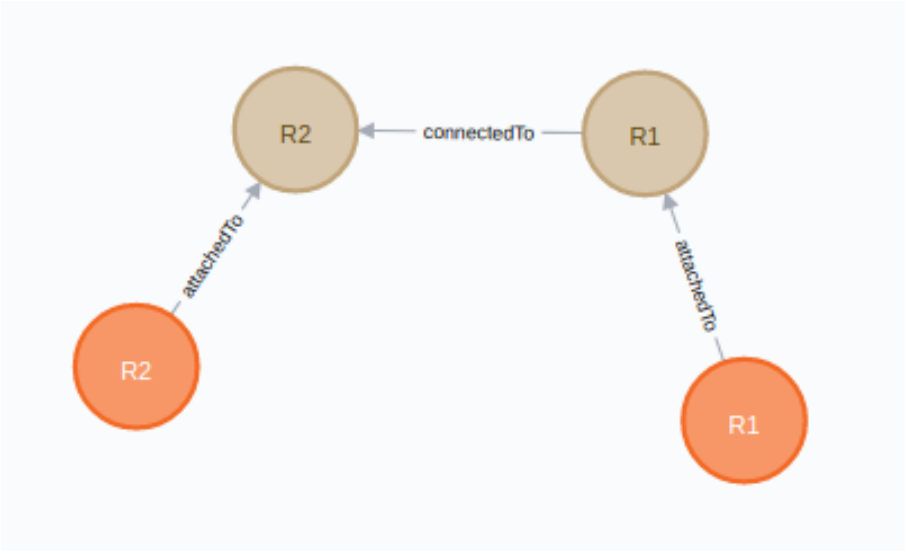


Abbildung 2.33: Geräte Verbindung

In der Abbildung 2.33 ist das Ergebnis ersichtlich, dass die beiden Geräte miteinander über eine Verbindung auf Layer 1 verbunden sind. Die optimale Verbindung ist jeweils über den Port GigabitEthernet0/1 von R1 wie auch von R2, zu welchen das Device jeweils eine attachedTo Verbindung hat. Zwischen den beiden Ports existiert eine connectedTo Verbindung.

IP Subnetz Konnektivität

In einem IP Subnetz sollte eine Full Mesh Verbindung der einzelnen IP Adressen existieren, da alle direkt miteinander kommunizieren können. Dies überprüfen wir im Subnetz 172.168.0/24 mit diesem Cypher-Befehl:
Match(n:Network) where n.IP starts with '172.168.150' return n

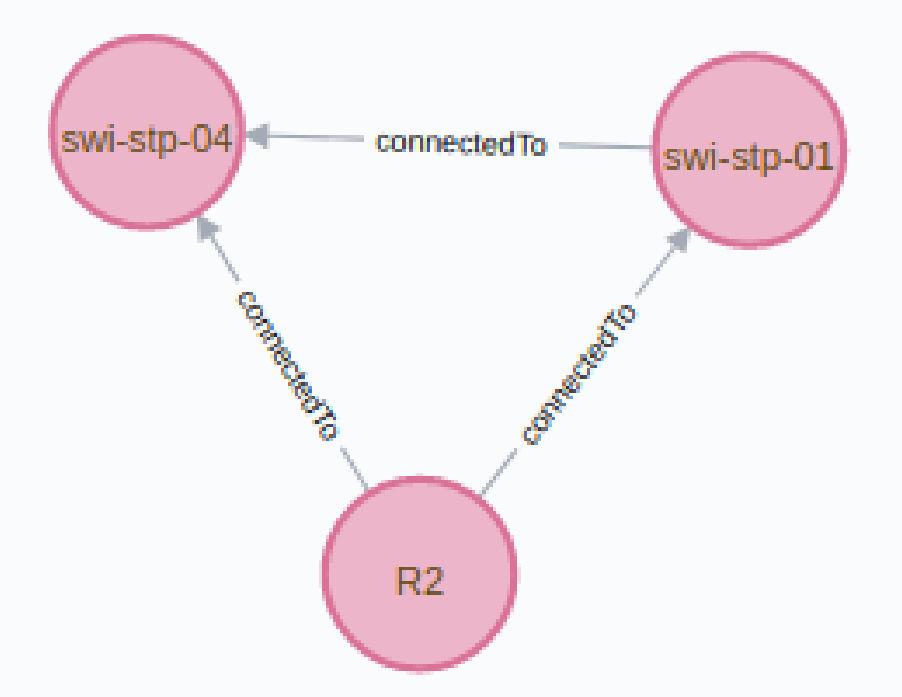


Abbildung 2.34: Subnet Konektivität

In der Abbildung 2.34 ist zu sehen, dass alle drei Nodes mit den IP Adressen 172.168.150.1, 172.168.150.10, 172.168.150.40 immer zwei connectedTo Verbindungen zu den anderen beiden IP Adressen haben, was eine Full Mesh Verbindung ergibt.

OSPF Shortest Path

Die OSPF ID 4.4.4.4 vom Layer 3 Switch swi-l3-01 sollte von der ID 1.1.1.1 des Router 1 nur über die ID 3.3.3.3 erreichbar sein. Um dies zu testen, haben wir ein Shortest Path von 1.1.1.1 nach 4.4.4.4 ausgeführt. Das ist dazugehörige Cypher Query:
MATCH (start:OSPF ID: '1.1.1.1'), (end:OSPF ID: '4.4.4.4'), p = shortestPath((start)-[:connectedTo*]-(end)) return *

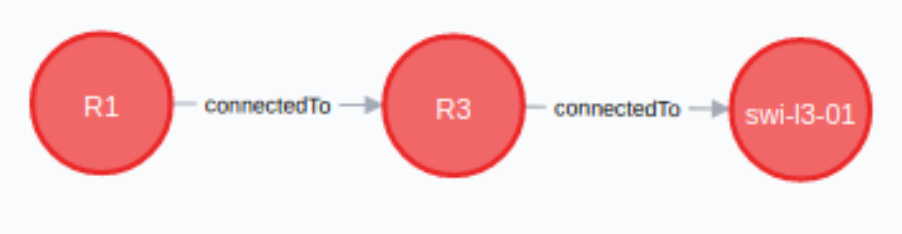


Abbildung 2.35: OSPF Shortest Path

In der Abbildung 2.35 ist im Shortest Path klar ersichtlich, dass nur eine Verbindung von R1 zum swi-l3-01 über den Router R3 im OSPF möglich ist, was dem Basisnetzwerk entspricht.

VLAN Konnektivität über das native VLAN

Beide Switches swi-stp-01 und swi-stp-04 besitzen das native VLAN 150 und haben jeweils eine Management IP Adresse in diesem VLAN konfiguriert. Die beiden IP's im VLAN 150 sollten dadurch in der Lage sein über Layer 3 miteinander zu kommunizieren. Zusätzlich braucht es eine Layer 2 Verbindung, um den nicht getaggten Netzwerkverkehr über den Trunk Port auszutauschen.

Dies haben wir mit folgendem Cypher Query überprüft:
MATCH(n) where n.Hostname starts with 'swi-stp-0' AND n:Network OR n:VLAN AND n.VLANID=150 return n

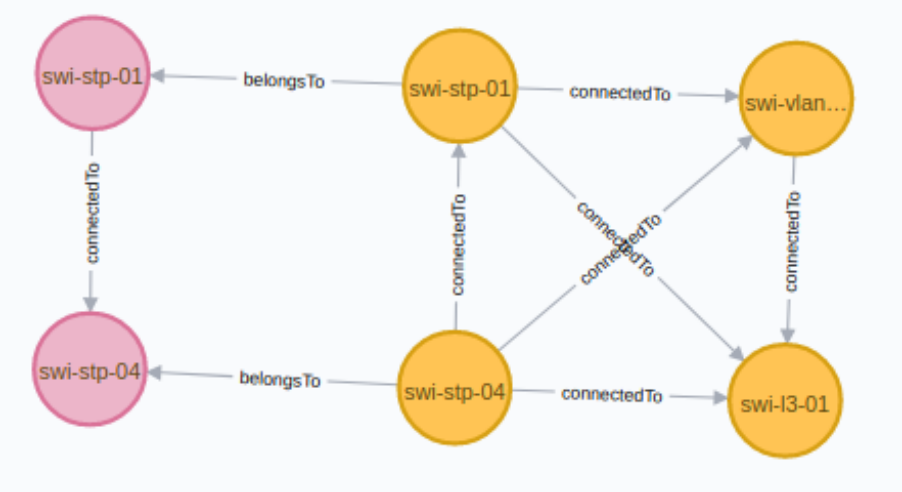


Abbildung 2.36: VLAN Gateway Konnektivität

In der Abbildung 2.36 ist erkennbar, dass das VLAN 150 auf den Geräten swi-stp-01 und 04 sowie auch auf swi-l3-01 und swi-vlan-01 konfiguriert ist. Innerhalb des VLAN's haben alle vier Geräte eine Verbindung. Der Hauptteil der Überprüfung gilt jedoch, dass eine IP-Verbindung innerhalb des VLAN 150 von swi-stp-01 zu swi-stp-04 möglich ist. Man kann erkennen, dass jeweils von VLAN zur IP eine belongsTo Verbindung besteht und danach die beiden IP-Adressen des VLAN 150 auch untereinander eine connectedTo Verbindung besitzen. Dies ist korrekt abgebildet.

2.3.10 Architektur Ansicht

Deployment Diagramm

Dies ist das Deployment Diagramm während der Entwicklungsphase der Studienarbeit. Die Applikation ist jeweils auf dem Entwicklungsgerät des Programmierers und die Datenbanken befinden sich auf einer virtuellen Maschine in der Azure Cloud von Microsoft. Zusätzlich stellte uns das INS zwei Cisco NX-OS Switche und ein Cisco CSR 1000v Router zur Verfügung, welche via RESTCONF abgefragt werden.

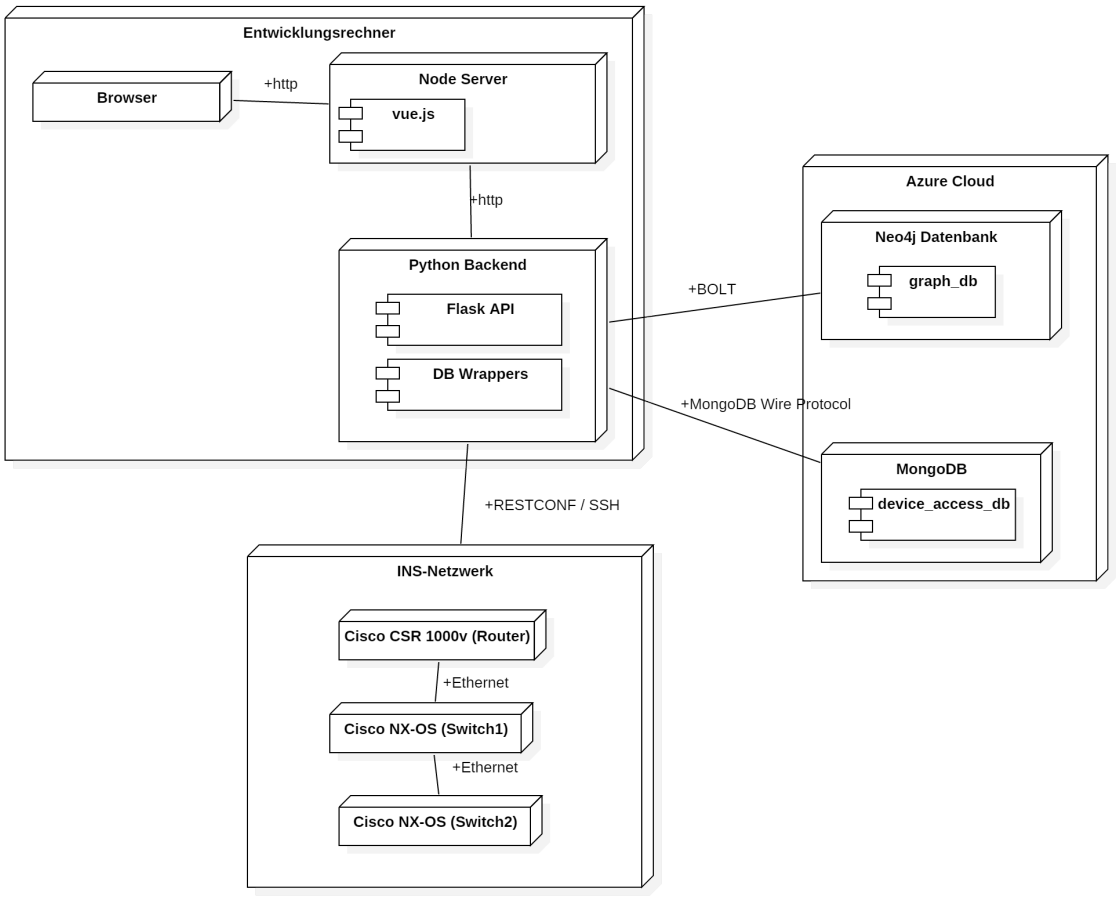


Abbildung 2.37: Deployment Diagramm

Schichtendiagramm

Die Applikation besteht aus vier Schichten: UI (User Interface), Business Layer, Services und Model. Das UI bilden die einzelnen Komponenten von Vue.js Komponenten aus denen eine Webseite generiert werden. Der in Vue.js integrierte Router stellt die Anfragen an die API vom Backend. Der Layer Manager unter Services ist der Mittelpunkt des Backends. Er nimmt die Informationen, die er von den Netzwerkgeräten erhält, verarbeitet und speichert sie in die Graph-DB über den Neo4J-DB-Wrapper. In der MongoDB unter Model sind die Zugangsdaten der Netzwerkgeräte verschlüsselt abgelegt.

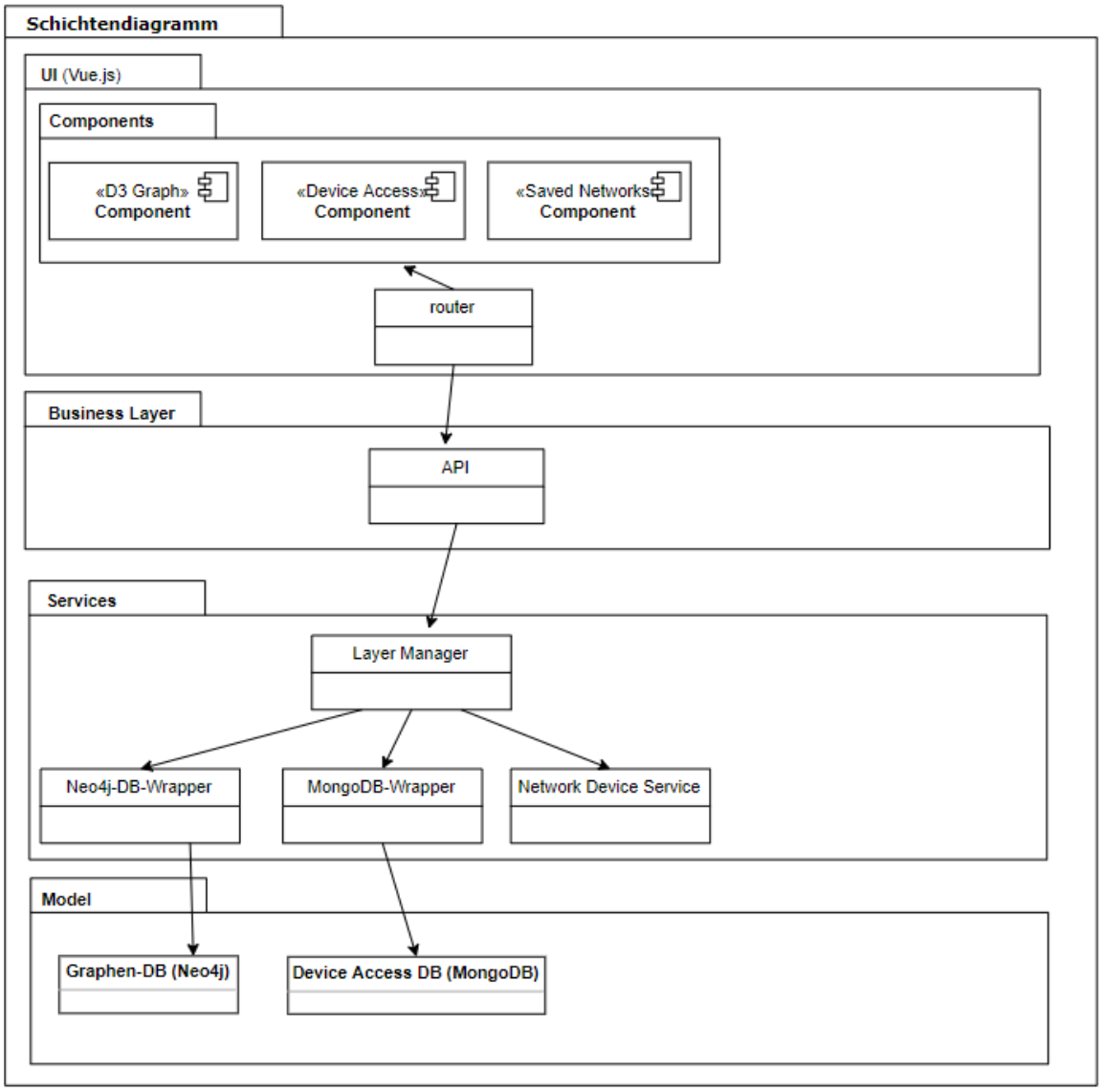


Abbildung 2.38: Layer Diagram

2.3.11 Architektur Entscheidungen

Nachfolgend werden die einzelnen Technologien aufgelistet, die für den Digital Twin benötigt werden, und erläutert, warum man sich dafür entschieden hat.

Neo4j

Context	Das persistente Abspeichern von einem Netzwerk inklusive den Konfigurationen braucht viel Speicherplatz.
Decision	Ein Netzwerk kann optimal in einem Graphen dargestellt werden. Daher benutzen wir Neo4j als Graphdatenbank.
Status	abgeschlossen
Consequences	Wir verwenden die Logik von Neo4j für das Einfügen und Auslesen von Daten. Da Neo4j bereits diverse Alogrithmen zur Verfügung stellt, werden wir versuchen, diese so weit wie möglich zu verwenden.

Zugang zu Netzwerkgeräte speichern

Context	Die Zugänge für die Abfrage der Netzwerkgeräte müssen abgespeichert werden. Da es keinen Sinn macht, die Informationen der Netzwerkgeräte auch in der Graphendatenbank abzuspeichern, soll eine andere Datenbank Technologie verwendet werden
Decision	Es wurde entschieden MongoDB als Datenbank für die Speicherung der Netzwerkgeräte zu verwenden, da für die Datenspeicherung die JSON-Syntax gebraucht wird.
Status	abgeschlossen
Consequences	Der Vorteil der JSON-Syntax ist, dass die Daten für die API nicht noch von einem anderen Format ins JSON umgewandelt werden müssen.

Backend

Context	Das Backend soll als API realisiert werden, damit das Frontend darauf zugreifen kann. Dafür braucht es eine Sprache, welche für die Abfragen von Netzwerkgeräten und mit der Graphen Datenbank Neo4J geeignet ist.
Decision	Python hat sich als die Sprache für die Netzwerkprogrammierung durchgesetzt. Wir hatten Python als kleine Einführung im Modul CloudInf und möchten unsere Kenntnisse weiter vertiefen.
Status	abgeschlossen
Consequences	Für Python stehen etliche Libraries zur Verfügung, unter anderem auch für Neo4j. Zusätzlich kann mit Flask eine API für den Digital Twin realisiert werden.

Datenabfrage der Netzwerkgeräte

Context	Es soll möglich sein, die benötigten Daten von den Netzwerkgeräten abzufragen, um diese dann im Netzwerk darzustellen. Zudem soll es auch umgekehrt die Konfiguration vom Digital Twin wieder auf die Geräte einspielen können.
Decision	RESTCONF ist eine weit verbreitete, herstellerunabhängige REST-Schnittstelle, die unter anderem auch von Cisco unterstützt wird. Aus diesem Grund haben wir uns für RESTCONF entschieden.
Status	abgeschlossen
Consequences	Die Daten der Netzwerkgeräte können via bekannte REST-Operationen wie get, put, patch etc. abgefragt und angepasst werden.

Frontend

D3.js
Data-Driven Documents (D3) ist eine JavaScript (JS)-Library um datanbasierte Dokumente zu manipulieren. Dies kann mithilfe von HTML, SVG oder CSS realisiert werden. D3 bedient sich einer grossen Beliebtheit und viele Graphenbeispiele von Entwicklern sind auf dem Internet zu finden.

D3.js ist ein umfangreiches Werkzeug für die Visualiserung von Daten. Die Einarbeitung davon wird eine Menge Zeit in Anspruch nehmen, dafür bietet es viele Möglichkeiten.

Plotly
Plotly ist eine Charting-Library und baut auf d3.js und stack.gl auf. Sie bietet über 20 Diagrammen an wie 3D-, statistische Diagramme und SVG-Karten an. Diese Library steht für die Sprachen Python, R und JS zur Verfügung.

Plotly bietet zwar einen 3D Point Clustering Diagramm an, welches Punkte in einem dreidimensionalen Raum darstellt, aber die Übersicht des Diagramms ist nicht zufriedenstellend und einen vernünftigen Graphen als Netzwerk kann nicht dargestellt werden.

sigma.js
Sigma ist eine JS-Library, um Graphen zu zeichnen, die mehrheitlich von zwei Entwicklern programmiert wurde.

Mit Sigma.js kann man schnell und einfach einen Graphen darstellen. Als Datenquellen können JSON-Files oder sogar direkt die Daten einer Neo4j-DB dienen.

3D Force-Directed Graph
3D Force-Directed Graph ist eine Webkomponente, um Graphen im dreidimensionalen Raum darzustellen. Für das 3D-Rendering wird ThreeJS/WebGL und für das physikalische Verhalten der Nodes entweder d3-force-3d oder ngraph verwendet.

Fazit
Keine der Libraries bietet von Haus aus einen Multilevel Graph an. Dieser muss mit der gewählten Library selber von Hand programmiert werden. Da 3D Force-Directed Graph Nodes und Edges im dreidimensionalen Raum darstellen kann, vereinfacht dies die Übersicht des Multilevel Graph. Somit haben

wir uns für die Visualisierung des Graphen für diese Technologie entschieden.

Context	Mit dem Frontend soll der Digital Twin graphisch dargestellt werden. Dabei ist zu beachten, dass ein Multigraph realisiert und die Informationen aus der Datenbank visualisiert werden können.
Decision	Wir haben uns für eine Webseite als Frontend entschieden. Dabei kommt das Web-Frontend Vue.js zum Einsatz und zur Visualisierung des Graphen die Library 3D Force-Directed Graph.
Status	abgeschlossen
Consequences	Dank einer Webseite spielt das Betriebssystem auf dem Computer eines Benutzers keine Rolle. Es braucht lediglich einen Browser, welcher HTML 5 unterstützt. Zudem gibt es diverse JS-Libraries, mit welchen man Graphen darstellen kann.

Code Quality (Testing und Linting)

Context	Um einen schönen und funktionierenden Code zu gewährleisten, werden statische Code-Analyse-Tools und Testingframeworks benötigt.
Decision	**Python:** Wir haben uns für pytest-Framework mit Tox für das Testing in Python entschieden, weil sie ständig weiterentwickelt werden und eine grosse Community dahinter steht. Für die statische Code-Analyse soll Pylint, welcher sehr vielseitig ist, verwendet werden. **JavaScript**: Erste Erfahrungen mit dem Testframework Mocha inkl. der Assertion Chai Library wurden bereits im Modul WED2 gesammelt. Für JS gibt es einen eigenen Linter: ESLint und dieser soll als statische Code-Analyse für dieses Projekt eingesetzt werden.
Status	abgeschlossen
Consequences	Diese Frameworks stellen einen sauberen und korrekten Code sicher.

2.3.12 Umfang der Studienarbeit

In der Studienarbeit soll ein Netzwerk als Digital Twin abgespeichert und dargestellt werden können, welcher die ersten drei Layer eines Netzwerks mit den Technologien STP mit VLAN, LACP, OSPF beinhaltet. Darin soll es möglich sein, eine Verbindung zwischen zwei Endpunkten zu überprüfen oder den Digital Twin manuell zu erweitern.

Folgende Use Cases gehören zum Minimum Viable Product (MVP) und sollen während der Studienarbeit umgesetzt werden:

- Netzwerk abspeichern

- Digital Twin manuell erwweitern
- Netzwerkegeräte Zugang abspeichern
- Digital Twin oder Netzwerk darstellen
- Verbindung zwischen zwei Endpunkten überprüfen

Zusätzliche Use Cases werden umgesetzt, sofern noch Zeit dafür übrig bleibt.

Bei dieser Arbeit wird sich auf Cisco Geräte mit RESTCONF beschränkt, um die Daten der Netzwerkgeräte abzufragen. Dies hat den Grund, da wir die Komplexität am Anfang klein halten wollen und die weiteren Varianten wie NAPALM später dazu genommen werden können. Bei Bedarf stellt das Institute for Networked Solutions Cisco Netzwerkgeräte zur Verfügung, welche RESTCONF unterstützen.

Die Software soll so gebaut werden, dass die übrigen OSI-Layer und zusätzlichen Technologien leicht im nachhinein hinzugefügt werden können.

2.3.13 Qualitätsanforderungen

Für den Digital Twin wurden diverse Qualitätsmerkmale definiert, welche die Qualität der Applikation sicherstellen soll.

Quality Tree

Nachfolgend sind die Qualitätsmerkmalen des Digital Twin in einer Baumstruktur visualisiert.

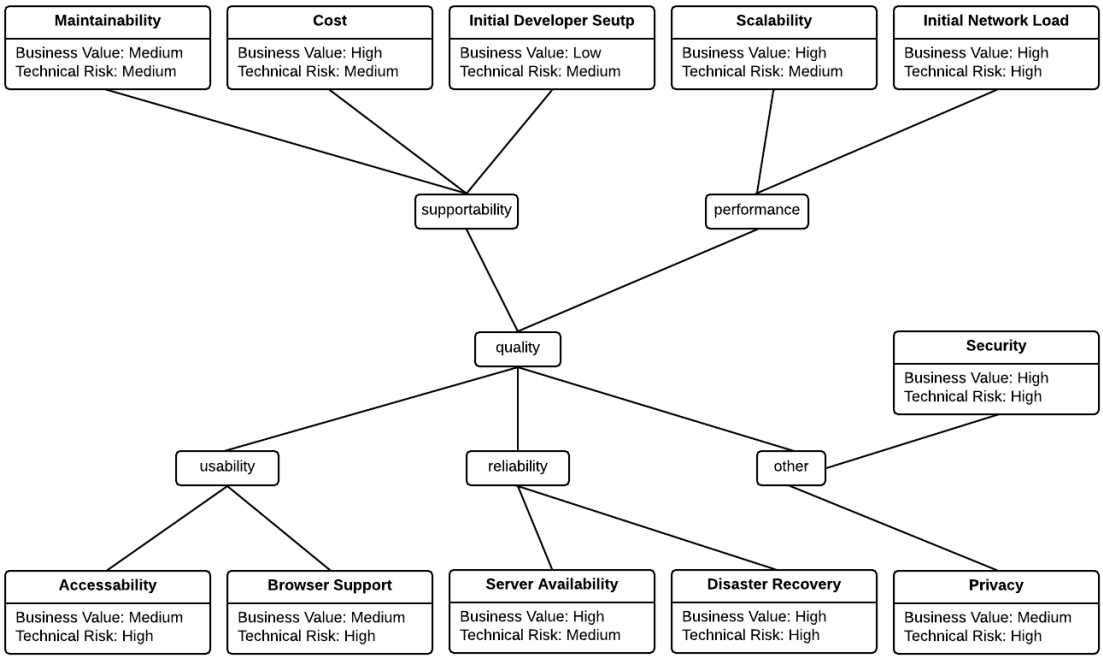


Abbildung 2.39: Quality Tree

Landing Zones

In der Landing Zone sind die einzelnen Qualitätsziele mit jeweils drei Zonen: Minimum Zone, Ziel Zone, aussendordentliche Zone aufgelistet. Die Qualitätsziele sollen gut messbare Werte sein, welche man mit den jeweiligen Zonen vergleichen kann, um festzustellen was für eine Qualität man erreicht hat.

Qualitätsziele	Beschreibung	Minimum Zone	Ziel Zone	ausserordentliche Zone
Maintainability	Die Einfachheit, um den Code in der Zukunft zu warten.	Manuelle code reviews	code quality Metriken erstellt	Automatische code reviews
Cost	Die Kosten, um das System zu betreiben.	<= CHF 75 / Monat	<= CHF 50 / Monat	<= CHF 25 / Monat
Initial Developer Setup	Benötigte Schritte, um das Developer Setup zu erstellen, damit entwickelt werden kann.	<= 10 Schritte	<= 5 Schritte	<= 2 Schritte
Scalability	Anzahl an Netzwerkgeräte, welche im Digital Twin problemlos abgebildet werden können.	>= 5	>= 20	>= 100
Initial Network load	Zeit, die gebraucht wird, um das komplette Netzwerk abzuspeichern.	<= 2m	<= 1m	<= 30s
Accessibility	Massnahmen, damit das System für alle verfügbar ist.	0 errors	<= 5 warnings	0 warnings
Disaster Recovery	Arbeit, um ein Disaster Recovery durchzuführen	Manuelle Backups	automatisierte Backups	automatisiertes Recovery
Privacy	Funktionalität, um die Privatsphäre zu schützen	nur interne Verfügbarkeit	Authentisierung	Auftrennung von Daten verschiedener Benutzer
Security	Funktinalität, um Sicherhet zu verbessern	interne Audit	Seurity Metriken erstellt	Externes Audit

2.4 Design Mockups

In diesem Kapitel werden die ersten Entwürfe der Benutzeroberfläche gezeigt.

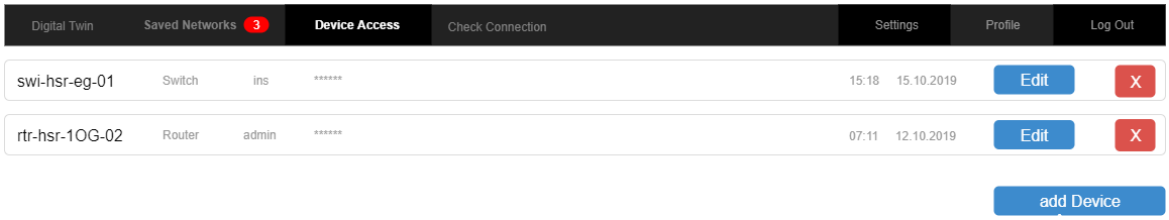


Abbildung 2.40: Mockup Device Access Webseite

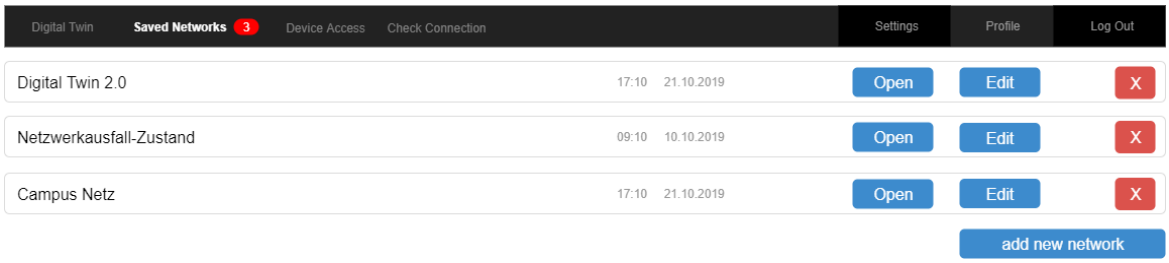


Abbildung 2.41: Mockup Saved Networks Webseite

Abbildung 2.42: Mockup Check Connection Webseite

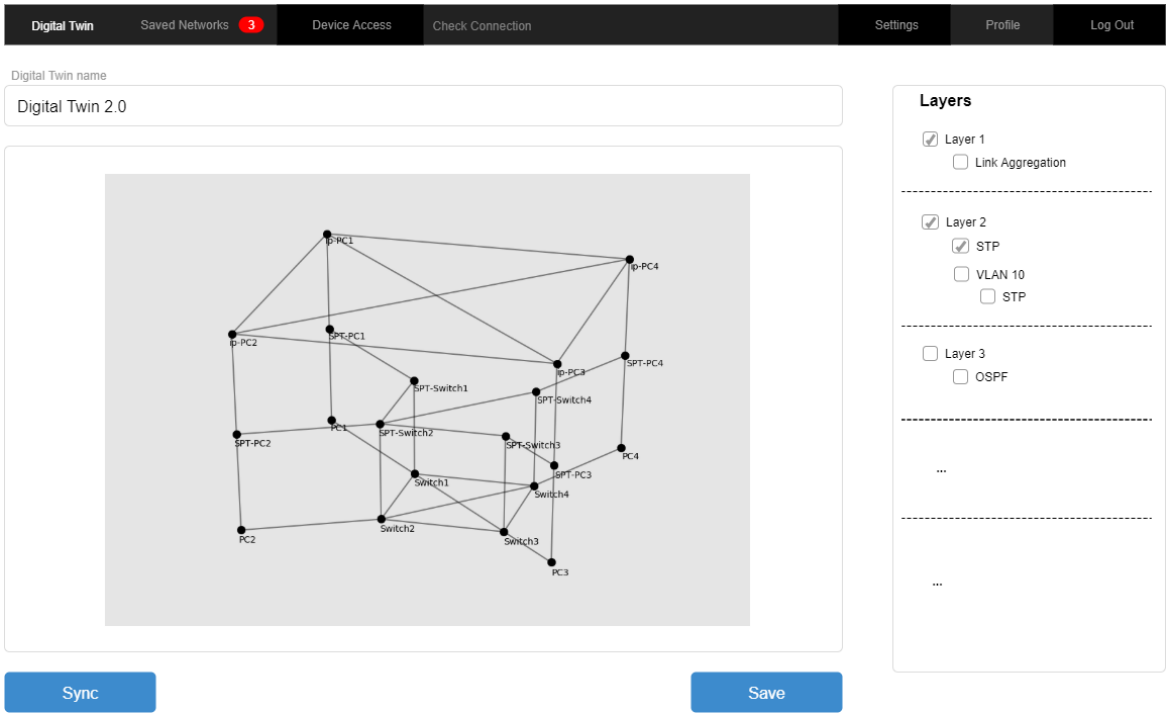


Abbildung 2.43: Mockup Digital Twin Webseite

2.5 Projektplan

2.5.1 Einführung

Zweck

Diese Projektplanung dient als Übersicht über die geplanten Meilensteine und Rahmenbedingungen (Werkzeuge Methodiken, usw.) für die Umsetzung der Studienarbeit. Es werden die Qualitätskriterien definiert, sowie ein Risikomanagement erstellt.

Gültigkeitsbereich

Dieses Dokument ist gültig während der ganzen Dauer (19.08.2019 bis 20.12.2019) und im Kontext der Studienarbeit.

2.5.2 Projekt Übersicht

Zweck und Ziel

Durch die zunehmende Komplexität und Grösse der Netzwerke ist es für Netzwerk-Ingenieure schwieriger auftretende Fehler im Netzwerk zu finden und zu beheben. Hilfreich könnte ein digitales Abbild, ein sogenannter "Digital Twin", des Netzwerks sein, welches den einwandfreien Soll Zustand mit korrekter Konfiguration abbildet und in einer Datenbank abgespeichert wird.

Die Informationen über die Komponenten des Netzwerk und die verwendeten Protokolle sollen nach OSI Layer strukturiert abgebildet und miteinander verknüpft werden. Dadurch kann der reale Zustand jeweils mit dem Abbild verglichen werden und bei allfälligen Problemen zu einer schnelleren Eingrenzung führen.

2.5.3 Projektorganisation

Organisationsstruktur

Das Team besteht aus den Mitgliedern Dario Caluzi und Patrick Lehmann, welche derzeit beide im 7. Semester vom Bachelor-Informatikstudium an der HSR sind. Betreut wird die Studienarbeit vom Dozent Beat Stettler und Marc Sommerhalder vom INS.

Da dieses Projekt lediglich von zwei Personen ausgeführt wird, teilen sich die beiden die Projektleitung und es werden keine weiteren Rollen zugeteilt. Die Arbeit wird individuell im Team aufgeteilt.

2.5.4 Management Abläufe

Kostenvoranschlag

Die Studienarbeit wird mit 8 ECTS-Punkte bewertet, was einen Zeitaufwand von 240 Stunden entspricht. Daraus ergibt sich rund 17 Wochenstunden pro Person.

Zeitliche Planung

Für die zeitliche Planung des Projekts wird Scrum [38] verwendet. Eine Sprintlänge von zwei Wochen wurde für diese Arbeit gewählt. Die Ausnahme bildet der erste Sprint die Inception- und die siebte während der Construction Phase. Daraus ergibt sich eine Anzahl von acht Sprints. Neben Scrum orientiert sich der Projektablauf an den Phasen des Unified Processes.

Iterationen

Name	Ziel	Dauer
Sprint 1: Inception	Erstellung Projektplans	1 Woche
Sprint 2: Elaboration	Analyse	2 Wochen
Sprint 3: Elaboration	Anforderungsspezifikation	2 Wochen
Sprint 4: Elaboration	Entwicklung Architektur-Prototyp	2 Wochen
Sprint 5: Construction	Entwicklung von Funktionalitäten	2 Wochen
Sprint 6: Construction	Entwicklung von Funktionalitäten	2 Wochen
Sprint 7: Construction	Feinschliff	1 Wochen
Sprint 8: Transition	Erstellung Abstract und Abgabe der Studienarbeit	2 Wochen

Meilensteine

Nachfolgend sind die einzelnen Meilensteine der Studienarbeit aufgelistet und beschrieben:

M1 - Abgabe Projektplan 23.09.2019
Projektplan abgeliefert

- Projektplan inkl. Risiko- und Qualitätsmanagement
- Erstellung LaTeX-Template für die Dokumentation der Arbeit
- Set Up von Gitlab Projekten für Dokumentation und Code

M2 - Analyse abgeschlossen 07.10.2019
gründliche Einarbeitung in das Thema

- Thematik verstehen
- Netzwerk in ein Graphen abspeichern
- Ausfall von Knoten in einen Graphen
- Netzwerkgeräte-Informationen abfragen

M3 - Anforderungsspezifikation abgeschlossen 21.10.2019
Alle wichtigen funktionalen und nicht-funktionalen Anforderungen erfasst.

- Fully Dressed Core Use Cases
- Brief Optional Use Cases
- Quality Attributes
- Domainmodel
- UI Mockup

M4 - End of Elaboration 04.11.2019
Alle Punkte der End of Elaboration Checkliste ist erledigt und ein Prototyp der Architektur steht.

- End of Elaboration Checkliste
- Tools und Technologien ausgewählt und konfiguriert
- Qualitätsprogramme ausgewählt und konfiguriert
- Architecture Prototype

M5 - Architekturdokumentation fertiggestellt 11.11.2019
Architektur ist stabil und dokumentiert

- Beschreibung der Architektur und -Entscheidungen
- Layer Model
- Deployment Model

M6 - Feature Freeze (End of Construction) 09.12.2019
Funktionsumfang ist befriedigend

- Release Note draft

M7 - Abgabe Studienarbeit
Produkt ist präsentierbar

- Abgabe Abstract 16.12.2019
- Abgabe Studienarbeit 20.12.2019
- Schlusspräsentation 09.01.2020

2.5.5 Besprechungen

Es wurde jeweils am Montag Nachmittag ein wöchentliches Meeting mit Beat Stettler und Marc Sommerhalder angesetzt, um den aktuellen Fortschritt der Arbeit zu besprechen.
Zusätzlich findet alle zwei Wochen zum Sprint Ende ein Scrum Meeting zwischen den beiden Projektmitgliedern statt, um die Sprintplanung, -Review und Entscheidungen zu besprechen.

2.5.6 Risikomanagement

Risiken

Die Risiken werden Aufgrund der einfacheren Zusammenarbeit in einem Google Spreadsheet gepflegt.[16] Nachfolgend der aktuelle Stand der technischen Risiken.

[16] https://drive.google.com/open?id=1ElbiccmdT23rRE-n0iQzQ0Wtg9wmHfn7aq1PQCrdZAo

Risikomanagement

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Visualisierung des Netzwerks	Die ausgewählte Technologie unterstützt nicht alle Anforderungen für das Anzeigen des Netzwerks		10%	0	Es wird bereits während der Elaboration Phase überprüft, ob die wichtigsten Funktionalitäten abgebildet werden können.	Die Komponenten müssen selbst entwickelt werden, oder neue Technologie eingesetzt werden, was neu Implementierung bedeutet.
R2	Gitlab Issue Tracking	Das Issue Tracking mit Gitlab ist ungenügend. Es erschwert die Sprintplanung und es kann nicht fliessend damit gearbeitet werden.	16	40%	6.4	Wir benutzen die Gitlab Issues bereits für die Inception Phase und sammeln unsere Erfahrungen damit.	Die Issues werden auf Redmine migriert.
R3	Latex Dokumentation	Es werden Funktionalitäten in der Dokumentation benötigt, welche nur sehr schwer oder gar nicht mit Latex gelöst werden können.	10	50%	5	Nicht zu viel Zeit verlieren mit Details, damit diese zwingend in Latex selber gemacht werden sollen.	Ausweichen auf ein anderes Medium für diesen Spezialfall. Anschliessend können die Resultate in die Dokumentation z.B. mit Bildern oder PDF Files eingebunden werden.
R4	Ungenaue Schätzung	Durch eine ungenaue Schätzung können die Meilensteine nicht erreicht werden und dadurch das ganze Projekt in Gefahr geraten, da die Dauer gesetzt ist.	30	80%	24	Abschätzungen anpassen nach der Iteration.	Überprufen welche Teile wirklich wichtig sind und welche nicht, um den Schaden möglichst klein zu halten.
R5	CI/CD	Die Gitlab CI/CD funktioniert nicht wie angedacht und die Pipeline schlägt regelmässig fehl und muss gefixt werden.	16	40%	6.4	Einzelne Projektmitarbeiter haben die Gitlab CI/CD schon an mehreren Orten angewendet und besitzen somit ein gewisses KnowHow.	Fixen der CI/CD Pipeline
R6	Zeiterfassung	Die Reporting Möglichkeiten des Zeiterfassungstool Tmetric sind ungenügend.	8	10%	0.8	Es ist sichergestellt, das gewisse Basis Reports bereits vorhanden sind. Zusätzlich gibt es eine Export Funktion.	Es wird mittels Excel oder anderen Tools die Auswertungen aus den Rohdaten aufbereitet.
R7	Inhaltliche Probleme	Themen wie das Netzwerk in einen Graphen darstellen oder wie können die Daten von versch. Gerätetypen und Hersteller abgefragt werden, können nicht so umgesetzt werden wie man es sich konzeptionell überlegt hat	10	35%	3.5	Intensives Evaluieren der Technologien.	Ausserordentliche Meetings mit Fachspezialisten vereinbaren, um mögliche Alternativen zu besprechen.
R8	Thematik nicht verstanden	Die Studenten verstehen die Anforderungen des Kunden nicht und haben Schwierigkeiten Lösungen für die Anforderungen zu finden.	4	25%	1	Wöchentlichen Meetings mit den Betreuern (Kunden) und das Nachfragen bei Problemen via Mail	Ausserordentliche Sitzungen vereinbaren, um die Unklarheiten zu beseitigen und das weitere Vorgehen zu besprechen.
R9	Entwicklung und Testing	Infolge von fehlenden Regressions-Tests treten bereits bekannte Bugs wieder auf.	18	15%	2.7	Beim Review wird sichergestellt, dass für alle Bugs ein zuständiger Unit Test erstellt wird.	Erneutes Fixen des Bugs und hinzufügen eines Unittests.
R10	Vergessene Anforderungen	Nicht alle Anforderungen wurden aufgenommen. Je nach Projektfortschritt müssen andere Technologien verwendet werden	30	50%	15	gründliche Anforderungsanalyse während der Elaboration Phase	Das Verwenden einer anderen Technologie und die Anpassung der Architektur
R11	Zu hohe Lernkurve	Einarbeitung in gewisse Themen/Tools/Methoden dauert zu lange	15	30%	4.5	Zeit Puffer einbauen	Sich gemeinsam in etwas arbeiten um die erforderliche Zeit aufzuteilen. Allenfalls Tool/Methode ändern.
R12	CI/CD Build Minuten	Die kostenlosen Build Minuten auf GitLab sind aufgebraucht.	4	100%	4	Regelmässige Prüfung der verfügbaren Build Minuten	Aufsetzen eines eigenen Build Servers oder kaufen von Build Minuten
R13	Problem mit ausgewählter Architektur	Probleme beim Anwenden der gewählten Architektur, z.B. Schwierigkeiten beim Anwenden der Graphen-DB, um die verfügbaren und unterbrochenen Links abzubilden	6	30%	1.8	Beim Evaluieren der Tools probieren die Kernanforderungen zu testen	Ausserordentliche Meetings mit Fachspezialisten / Dozenten vereinbaren, um mögliche Lösung zu besprechen.
R14	Architektur falsch konzipiert	Durch eine falsche konzipierte Architektur können die Anforderungen nicht in der Anwendung umgesetzt werden	15	25%	3.75	Überprüfen, ob alle Kernanforderungen mit der konzipierten der Architektur umgesetzt werden kann.	Überarbeitung der Architektur. Evtl. neue Tools evaluieren, konfigurieren und testen
R15	End-to-End Tests	Die automatisierten End-to-End Tests benötigen mehr Aufwand als erwartet.	16	50%	8	Es wird zu Beginn der Umsetzung geprüft, ob der Aufwand für End-to-End der Erwartung entspricht	Es wird eine Liste der End-to-End Tests geführt und jeweils am Sprint Ende manuell ausgeführt und protokolliert.
Summe			**80**		**86.85**		

2.5.7 Arbeitspakete

Arbeitspakete werden in diesem Projekt durch Gitlab Issues repräsentiert. Die komplette Liste der Issues inklusive dem jeweiligen Status sind unter der folgenden URL zu finden:
gitlab.dev.ifs.hsr.ch/sa_digital_twin/digital_twin/issues

Da die Issues unter permanenter Anpassung sind, wird in diesem Dokument auf einen Auszug verzichtet.

2.5.8 Infrastruktur

Gitlab und Gitlab CI

Zur Versionierung der Codebase und auch der Dokumentationen in Latex wird der git Service des HSR Gitlab [17] Systems verwendet.

Die Gitlab CI Pipelines aus dem selben System werden zur Continuous Integration verwendet.

Wir konnten bereits einige Erfahrungen mit Gitlab im Engineering Projekt sammeln. Ausserdem ist eine gute Dokumentation dazu vorhanden.

2.5.9 Qualitätsmassnahmen

Reviews

Alle Änderungen im Programmcode sowie auch in den Dokumentationen werden vom anderen Teammitglied kontrolliert (Review), bevor diese in den Master Branch zusammengeführt (Merge) werden dürfen. Ausser es sind triviale Änderungen wie z.B. das Beheben von Schreib- oder Formatierungsfehlern.

Reviews werden über Gitlab Merge Requests gemacht, welche eine Plattform für Anmerkungen und Diskussionen bietet. Dadurch wird die Kommunikation dokumentiert und kann im Zweifelsfall im Nachhinein konsultiert werden.

Die Merge Requests werden direkt dem anderen Teammitglied zugewiesen, damit dieser eine Benachrichtigung erhält, dass ein neuer Request zum Review bereit steht.

Pipelines

Nach jedem Commit wird auf dem jeweiligen Branches eine Pipeline ausgeführt, welche den geschriebenen Text bzw. Programmcode überprüft. Bevor eine Änderung mittels Commit vermerkt wird, muss der Code kompiliert, mit dem jeweilige Linter geprüft werden und die Tests erfolgreich durchgelaufen sein, damit die Pipeline nicht fehl schlägt. Falls es doch mal zu dieser Situation kommen sollte, muss die Pipeline sofort wieder zum Laufen gebracht werden.

[17] gitlab.dev.ifs.hsr.ch/sa_digital_twin

Dokumentation

Dokumente werden in LaTeX verfasst, mit Git versioniert und auf GitLab[18] gehostet. Dies ermöglicht es, dass beide Teammitglieder gleichzeitig am selben Dokument arbeiten und Merge Requests für Reviews (siehe 2.5.9) nutzen können.

Projektmanagement

Für das Projektmanagement werden GitLab Issues[19] eingesetzt. GitLab kennt keine Sprints, deshalb wird das Kanban Board als Übersicht dafür verwendet.

Issues werden jeweils mit Labels kategorisiert. Beispielsweise gibt es ein *Dokumentation* Label, um anzugeben, welche Issues für Dokumentationsarbeiten erforderlich sind.

Zeiterfassung

Die Teammitglieder erfassen ihre Arbeitszeit in TMetric[20], da gitlab eine mangelnde Zeiterfassung bietet. Es kann mit Gitlab verknüpft werden, um die Zeiten einzelner Issues zu pflegen. Dabei fügt die TMetric Browser Extension einen Button in die GitLab Benutzeroberfläche ein, mit dem ein Timer auf einem bestimmten Issue gestartet und wieder gestoppt werden kann.

TMetric übernimmt automatisch das Projekt aus GitLab und die Art der Tätigkeit kann als Tag zugewiesen werden.

Definition of Done

Ein Feature wird als erledigt betrachtet, wenn der Merge Request erfolgreich den Review-Prozess (2.5.9) durchlaufen hat und gemerged wurde.

[18] gitlab.dev.ifs.hsr.ch/sa_digital_twin/sa_docs
[19] gitlab.dev.ifs.hsr.ch/sa_digital_twin/digital_twin/issues
[20] tmetric.com

3 Anhänge

3.1 Installationsanleitung

3.1.1 Frontend

Anforderungen

Um die Applikation starten zu können, müssen bereits folgende Programme installiert sein:

- Node.js Version 11.12 oder neuer
- npm Version 6.7.0 oder neuer

Einstellungen

Damit das Frontend korrekt mit dem Backend kommunizieren kann, muss die Adresse der API des Backends im Frontend eingestellt werden.

Dazu gibt es im Projektverzeichnis das File **.env**. Darin muss die Variable **VUE_APP_BACKEND_URI** auf die korrekte URI des Backends gesetzt werden.

Starten

Wenn die Applikation lokal gestartet werden möchte, kann man dies direkt mit npm tun. Navigieren sie in der Konsole in das Projektverzeichnis und geben sie folgenden Befehl ein:

```
npm run serve
```

Dann wird das Frontend gebildet und direkt lokal ausgeführt.

Deployment

Die Frontendapplikation kann jedoch auch auf einen beliebigen HTTP Server bereitgestellt werden. Dazu müssen die Ressourcen zuerst gebildet werden.

Das können Sie in der Konsole mit folgendem Befehl erreichen:

```
npm run build
```

Dieser Befehl generiert im Projektverzeichnis ein neues Verzeichnis **dist**, dieses kann danach gemäss ihrem HTTP Server bereitgestellt werden.

3.1.2 Backend

Anforderungen

Um das Backend des Digital Twins auf einem Server auszuführen, können Sie sich die Sourcen in das gewünschte Verzeichnis kopieren.

Diese Programme müssen bereits auf dem Zielsystem installiert und funktionsfähig sein:

- Python Version 3.7.4 oder neuer
- pip Version 19.2.3 oder neuer

Damit alle Anforderungen für das Ausführen des Backend für den Digital Twin gegeben sind, müssen alle Packages, welche im requirements.txt enthalten sind, auf dem System installiert sein.

Dies kann mit folgendem Befehl auf der Konsole ausgeführt werden:

```
pip install -r requirements.txt
```

Einstellungen

Für die Einstellungen des Backend wird eine Konfigurationsdatei verwendet. Im Projekt ist bereits eine Datei vorhanden. Damit die Applikation allerdings die Konfigurationsdatei findet, muss eine Umgebungsvariable mit dem absoluten Pfad auf die Datei erstellt werden.

Unter **Linux** den Pfad zu Digital Twin Settings als Environment Variable exportieren:

```
export DIGITAL_TWIN_SETTINGS=/path_to_project/digital_twin_backend.cfg
```

Unter **Windows** den Pfad zu Digital Twin Settings als Environment Variable exportieren:

```
set DIGITAL_TWIN_SETTINGS=C:\path_to_project\digital_twin_backend.cfg
```

Nun können Sie im File **digital_twin_backend.cfg** im Projektverzeichnis die gewünschten Einstellungen für die Applikation vornehmen. Hier ist eine Beispiel Konfiguration ersichtlich:

```
# Flask app configuration
HOST='127.0.0.1'
PORT=5000
# Debug level configuration
DEBUG = False

# MongoDB Configuration
MONGODB_URI = 'mongodb://192.168.10.1:27017/',
MONGODB_USER = 'dt_user'
MONGODB_PASSWORD = 'asdsfIFWNVowv'
MONGODB_AUTH_SOURCE = 'digital_twin_credentials'
MONGODB_AUTH_MECHANISM = 'SCRAM-SHA-256'
MONGODB_KEY = '46Nts9eFKsqSd7L2KFIJ_yzVX1sf32f84de2f1HcyXv6hdM='

# Neo4j Configuration
NEO4J_URI = 'bolt://192.168.10.2:7687'
NEO4J_USER = 'neo4j'
NEO4J_PASSWORD = 'ivJFZ/62u48N'
```

Abbildung 3.1: Beispielkonfiguration für Digital Twin Backend

Das Konfigurationsfile, wie in Abbildung 3.1 ersichtlich, ist auf drei verschiedene Bereiche aufgeteilt.

Zuerst sind die Einstellungen für die Flask App selber einzustellen, wie, auf welcher IP und Port das Backend erreichbar sein soll. Zusätzlich kann man auch den DEBUG Level wie gewünscht setzen.

Zweitens sind die Einstellung für die MongoDB zur Verwaltung der Credentials zu tätigen. Alle ersichtlichen Variablen sind Pflicht. Fehlt eine, wird die Applikation nicht laufen. Bis auf die letzte sind alle Variablen jedoch Standard für MongoDB Verbindungen. Der MONGODB_KEY am Ende wird benötigt, um die Passwörter in der Datenbank zu verschlüsseln und zur Benutzung im Backend wieder zu entschlüsseln. **Achtung:** Bitte beachten Sie, dass wen der Key nach dem Abspeichern von Passwörtern geändert wird, können alle zuvor verschlüsselten Passwörter nicht mehr entschlüsselt werden, und die Credentials sind somit nutzlos.

Zuletzt bei den Einstellungen für Neo4j sind auch alle Variablen Pflicht für die Funktionsfähigkeit der Applikation. Hier werden auch standardmässig Pfad zur Datenbank sowie Benutzername und Passwort verlangt.

Starten

Das Backend kann nun gestartet werden, indem man im Projektverzeichnis folgenden Befehl in der Konsole ausführt:

```
python src/app.py
```

Auf der Konsole wird nun ausgegeben auf welchem Socket die Applikation erreichbar ist, mit beispielsweise **Running on http://127.0.0.1:5000/**. Anschliessend werden nun auch Infos und allfällige Fehler ausgegeben.

3.1.3 Persistenz

Für die Persistenz kann jegliche MongoDB Datenbank dazu verwendet werden, die Credentials der Netzwerkgeräte zu verwalten. Analog dazu kann natürlich auch jegliche neo4j Datenbank für den Graphen verwendet werden.

Die entsprechende Datenbank und Collections müssen, wie in Kapitel 3.1.2 beschrieben, im Konfigurationsfile der Backend Applikation eingetragen werden.

3.2 Benutzeranleitung

In diesem Kapitel werden die wichtigsten Funktionen beschrieben, die beim Digital Twin ausgeführt werden können.

3.2.1 Graph anzeigen

Unter dem Menüpunkt *Digital Twin* kann der Graph des Digital Twin betrachtet werden. Standardmässig wird der ganze Graph angezeigt.

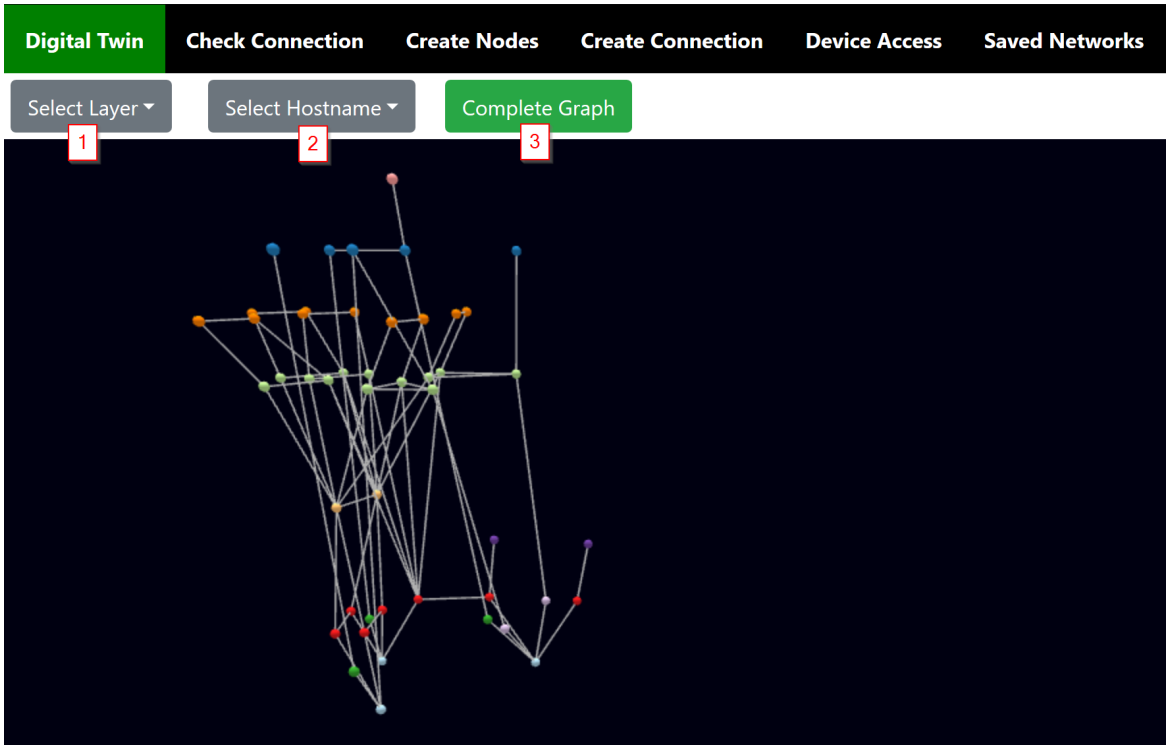


Abbildung 3.2: Ansicht Digital Twin Seite

1. Bestimmte OSI-Layer können im Dropdown Menü *Select Layer* ausgewählt werden.
2. Um einen Graph von einem bestimmten Gerät anzeigen zu lassen, kann der entsprechende Hostname im Dropdown Menü *Select Hostname* ausgewählt werden.
3. Falls wieder der komplette Graph angezeigt werden soll, kann der Button *Complete Graph* gedrückt werden.

Hinweis: Wenn ein Graph nicht angezeigt wird, kann das zwei verschiedene Gründe haben. Dabei wird jeweils dem User eine entsprechende Meldung angezeigt:

- Der Graph besteht nur aus Nodes, die keine Verbindungen zueinander haben.
- Es konnte keine Verbindung zum Backend hergestellt und somit die Daten nicht abgefragt werden.

3.2.2 Überprüfe Verbindung zwischen zwei Nodes

Um eine Verbindung zwischen zwei Geräten zu überprüfen, kann der Menüpunkt *Check Connection* gewählt werden.

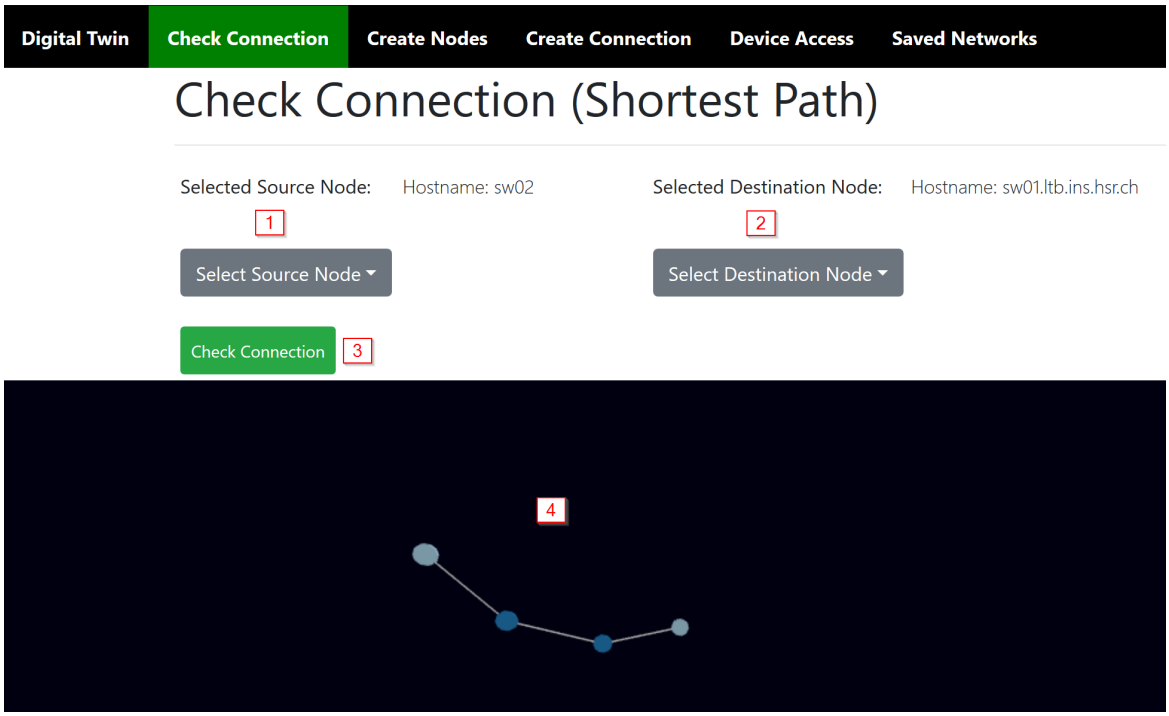


Abbildung 3.3: Ansicht Check Connection Seite

1. Als erstes muss im Dropdown Menü *Select Source Node* der Node ausgewählt werden, von dem aus die Verbindung getestet werden soll.
2. Als nächstes muss noch der Ziel Node im Dropdown Menü *Select Destination Node* selektiert werden.
3. Zum Schluss muss der Button *Check Connection* gedrückt werden, um den Shortest Path zwischen den beiden Geräten anzeigen zu lassen.
4. Hier wird die Verbindung mit den einzelnen Nodes als Graph dargestellt.

3.2.3 Neuer Node erstellen

Der Digital Twin kann um weitere Nodes erweitert werden. Dies kann im Menüpunkt *Create Nodes* getätigt werden.

Es kann ein Node von einem der folgenden Typen erstellt werden:

- Device Node
- Physical Layer Node
- Link Aggregation Node

- VLAN-Interface Node
- Loopback Node
- Data Link Layer Node
- VLAN Node
- PVST Node
- Network Layer Node
- OSPF Node

Es wird anhand eines Physical Layer Node gezeigt, wie ein neuer Node erstellt werden kann.

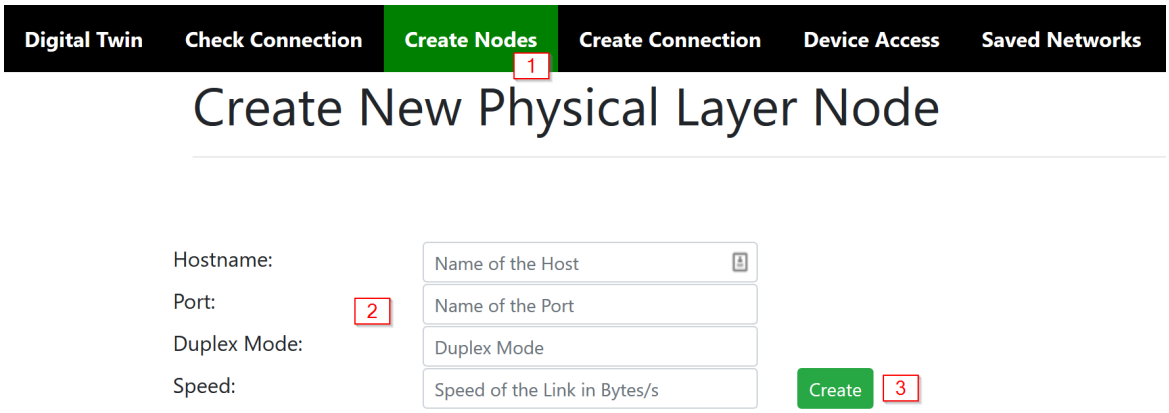


Abbildung 3.4: Ansicht Create New Physical Layer Node Seite

1. Als erstes wählt man im Dropdown Menü vom Menüpunkt *Create Nodes* den Physical Layer aus.
2. Je nach Nodetyp unterscheidet sich die Angaben, die dafür benötigt werden. Alle diese Felder müssen ausgefüllt werden, bevor der Node erstellt werden kann.
3. Abschliessend muss der Button *Create* gedrückt werden, um den Node zu erstellen. Es wird eine Nachricht angezeigt, ob der Node erstellt werden konnte oder nicht.

3.2.4 Neue Verbindung erstellen

Zwischen bestehenden Verbindungen können drei verschiedene Arten von Verbindungen erstellt werden:

- *ConnectionTo:* Verbindung zwischen zwei Nodes von unterschiedlichen Geräten
- *BelongsTo:* eine Verbindung innerhalb eines Geräts
- *AttachTo:* eine Verbindung zwischen einem Device Node und einem Port Node.

Im Menüpunkt *Create Connection* kann eine neue Verbindung zwischen zwei Nodes erstellt werden.

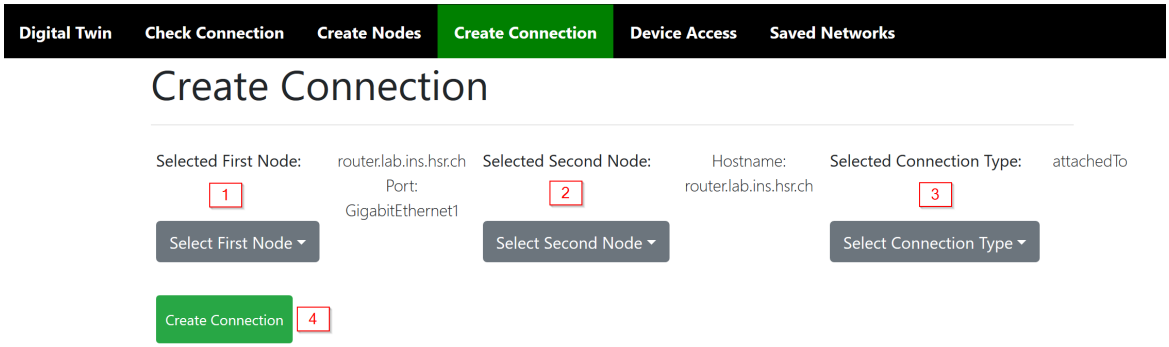


Abbildung 3.5: Ansicht Create Connection Seite

1. Im Dropdown Menü *Select First Node* muss der erste Node ausgewählt werden.
2. Der zweite Node wird im Dropdown Menü *Select Second Node* selektiert.
3. Nun muss noch bei *Select Connection Type* die Art der Verbindung angegeben werden.
4. Zum Schluss muss auf den Button *Create Connection* geklickt werden.

3.2.5 Zugangsdaten eines Netzwerkgeräts erstellen

Um den Digital Twin mit weiteren Netzwerkgeräten zu erweitern, müssen die Zugangsdaten unter dem Menüpunkt *Device Access* erfasst werden.

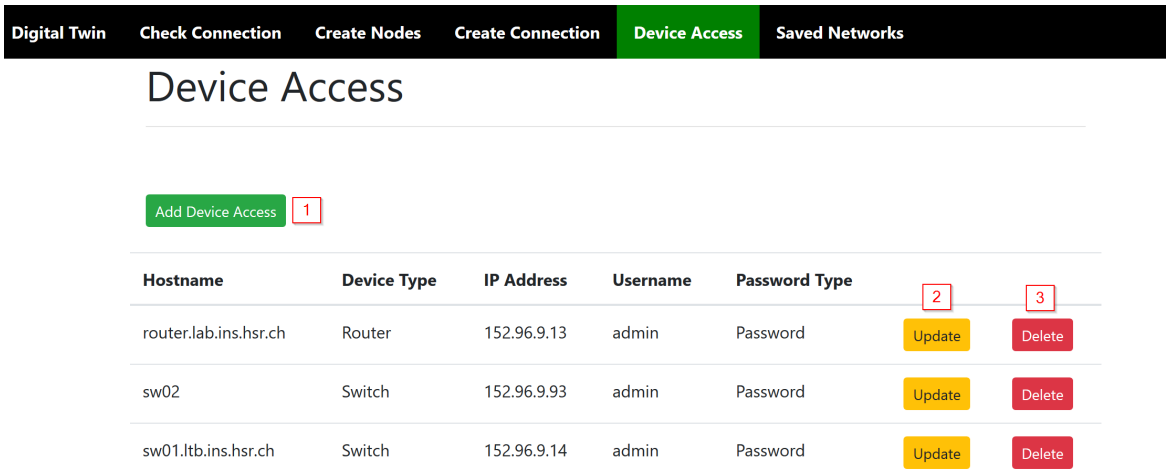


Abbildung 3.6: Ansicht Device Access Seite

1. Es können neue Zugangsdaten mit den Button *Add Device Access* hinzugefügt werden. Dabei erscheint ein Fenster, in welchem die benötigten Daten über das Gerät angegeben werden müssen.
2. Es kann jedoch auch bestehende Zugänge mit den Button *Update* geändert werden.

3. Zum Schluss kan ein Gerät mit *Delete* aus der Liste entfernt werden.

Hinweis: Es ist wichtig, dass die einzelnen Informationen über das Netzwerkgerät richtig ausgefüllt werden. Sonst kann das Backend keine Verbindung zum Netzwerkgerät herstellen, um die Informationen abzufragen.

3.2.6 Digital Twin erstellen

Unter dem Menüpunkt *Saved Networks* kann der aktuelle Zustand des Netzwerks geklont werden.

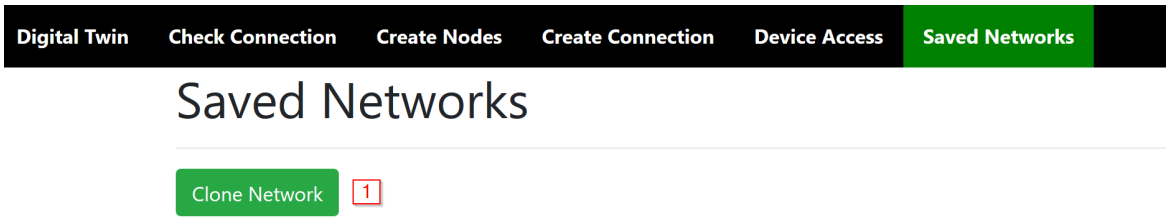


Abbildung 3.7: Ansicht Saved Networks Seite

1. Mit dem Button *Clone Network* werden alle Netzwerkgeräte abgefragt, die auf der Seite *Device Access* erfasst wurden, und deren Informationen in der Graphdatenbank abgespeichert. Beim Drücken des Buttons erscheint eine Infomeldung, ob man wirklich das Netzwerk klonen möchte, da die bestehenden Daten in der Graphdatenbank gelöscht und mit neuen Informationen befüllt wird.

Hinweis: Das Klonen des Netzwerks kann einige Zeit in Anspruch nehmen, da alle Geräte abgefragt werden und deren Informationen vearbeitet und in die Datenbank abgespeichert werden müssen. Der Button symbolisiert mit einem Spinner, ob das Klonen noch im Gange ist.

3.3 Glossar

CAN Campus Area Network

LAN Local Area Network

WAN Wide Area Network

VPN Virtual Private Network

MPLS Multiprotocol Label Switching

ISP Internet Service Provider

IAN Internet Area Network

VLAN Virtual Local Area Netwerk

BFS Breadth First Search

DFS Depth First Search

STP Spanning Tree Protocol

IP Internet Protocol

CLI Command Line Interface

LAG Link Aggregation Group

LACP Link Aggregation Control Protocol

OSPF Open Shortest Path First

MVP Minimum Viable Product

HTML Hypertext Markup Language

JS JavaScript

D3 Data-Driven Documents

INS Institute for Networked Solutions

LTE Long-Term Evolution

SNMP Simple Network Management Protocol

MIB Management Information Base

3.4 Literaturverzeichnis

[1] Mikko Kivelä, *Multilayer Networks*, 2014.

[2] (2019). Network performance monitor | SolarWinds, Adresse: https://www.solarwinds.com/network-performance-monitor (besucht am 15. 12. 2019).

[3] (2019). PRTG network monitor » all-in-one network monitoring software, Adresse: https://www.paessler.com/prtg (besucht am 15. 12. 2019).

[4] (2019). Vue.js, Adresse: https://vuejs.org/ (besucht am 16. 12. 2019).

[5] V. Asturiano, *vasturiano/3d-force-graph*, original-date: 2017-02-27T04:09:20Z, 16. Dez. 2019. Adresse: https://github.com/vasturiano/3d-force-graph (besucht am 16. 12. 2019).

[6] (2015). Flask, Pallets, Adresse: https://palletsprojects.com/p/flask/ (besucht am 16. 12. 2019).

[7] (2019). The most popular database for modern apps, MongoDB, Adresse: https://www.mongodb.com (besucht am 16. 12. 2019).

[8] (2019). PyMongo 3.9.0 Documentation — PyMongo 3.9.0 documentation, Adresse: https://api.mongodb.com/python/current/ (besucht am 16. 12. 2019).

[9] (2019). Neo4j database, Neo4j Graph Database Platform, Adresse: https://neo4j.com/neo4j-graph-database/ (besucht am 16. 12. 2019).

[10] (2018). Neo4j Bolt Driver 1.7 for Python — Neo4j Bolt Driver 1.7 for Python, Adresse: https://neo4j.com/docs/api/python-driver/current/index.html#api-documentation (besucht am 16. 12. 2019).

[11] (2017). Fernet (symmetric encryption) — Cryptography 2.9.dev1 documentation, Adresse: https://cryptography.io/en/latest/fernet/ (besucht am 15. 12. 2019).

[12] (15. Dez. 2019). ipaddress — IPv4/IPv6 manipulation library — Python 3.8.1rc1 documentation, Adresse: https://docs.python.org/3/library/ipaddress.html (besucht am 17. 12. 2019).

[13] R. M. Wyatt, «Link Aggregation», Pat., US Patent 6,765,866 B1, 2004. Adresse: https://patents.google.com/patent/US6765866B1/en.

[14] (11. Okt. 2013). Detect cycle in an undirected graph, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/detect-cycle-undirected-graph/ (besucht am 13. 12. 2019).

[15] (28. Nov. 2018). Detect cycle in a directed graph using BFS, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/detect-cycle-in-a-directed-graph-using-bfs/ (besucht am 13. 12. 2019).

[16] (20. Jan. 2017). Graph implementation using STL for competitive programming | set 2 (weighted graph), GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/graph-implementation-using-stl-for-competitive-programming-set-2-weighted-graph/ (besucht am 13. 12. 2019).

[17] (). Graphs and Subgraphs - Mathonline, Adresse: http://mathonline.wikidot.com/graphs-and-subgraphs (besucht am 13. 12. 2019).

[18] (13. Juli 2018). Mathematics | walks, trails, paths, cycles and circuits in graph, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/mathematics-walks-trails-paths-cycles-and-circuits-in-graph/ (besucht am 13. 12. 2019).

[19] (1. Jan. 2019). Difference between graph and tree, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/difference-between-graph-and-tree/ (besucht am 13. 12. 2019).

[20] (). Data Structure & Algorithms - Spanning Tree - Tutorialspoint, Adresse: https://www.tutorialspoint.com/data_structures_algorithms/spanning_tree.htm (besucht am 13. 12. 2019).

[21] (17. Apr. 2018). Multistage graph (shortest path), GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/multistage-graph-shortest-path/ (besucht am 13. 12. 2019).

[22] (12. Okt. 2017). Detect a negative cycle in a graph | (bellman ford), GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/detect-negative-cycle-graph-bellman-ford/ (besucht am 13. 12. 2019).

[23] (). Strongly connected components tutorials & notes | algorithms, HackerEarth, Adresse: https://www.hackerearth.com/practice/algorithms/graphs/strongly-connected-components/tutorial/ (besucht am 13. 12. 2019).
[24] (3. Nov. 2013). Travelling salesman problem | set 1 (naive and dynamic programming), GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/ (besucht am 18. 12. 2019).
[25] (5. Sep. 2019). Check if given an edge is a bridge in the graph | algorithms, Adresse: https://algorithms.tutorialhorizon.com/check-if-given-an-edge-is-a-bridge-in-the-graph/ (besucht am 13. 12. 2019).
[26] (21. Mai 2013). Articulation points (or cut vertices) in a graph, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/ (besucht am 13. 12. 2019).
[27] (30. Okt. 2012). Kruskal's minimum spanning tree algorithm | greedy algo-2, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/ (besucht am 13. 12. 2019).
[28] (18. Nov. 2012). Prim's minimum spanning tree (MST) | greedy algo-5, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/ (besucht am 13. 12. 2019).
[29] (12. Mai 2013). Topological sorting, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/topological-sorting/ (besucht am 15. 12. 2019).
[30] (25. Nov. 2012). Dijsktra's algorithm, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/ (besucht am 07. 10. 2019).
[31] (). Bellman–Ford Algorithm | DP-23 - GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/ (besucht am 07. 10. 2019).
[32] (7. Juni 2012). Floyd warshall algorithm | DP-16, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/ (besucht am 07. 10. 2019).
[33] (20. Aug. 2014). Tarjan's algorithm to find strongly connected components, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/ (besucht am 07. 10. 2019).
[34] (22. Mai 2013). Bridges in a graph, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/bridge-in-a-graph/ (besucht am 07. 10. 2019).
[35] (). Articulation Points (or Cut Vertices) in a Graph - GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/ (besucht am 07. 10. 2019).
[36] (). Prim's Minimum Spanning Tree (MST) | Greedy Algo-5 - GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/ (besucht am 07. 10. 2019).
[37] (3. Juli 2013). Ford-fulkerson algorithm for maximum flow problem, GeeksforGeeks, Adresse: https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/ (besucht am 07. 10. 2019).
[38] Jeff Sutherland and Ken Schwaber, *The Scrum Guide*. 2017.

3.5 Abbildungsverzeichnis