Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Entwurf und Implementierung eines web-basierten Centralized Network Controller (CNC) für Time-Sensitive Networking (TSN)

Mickey Redecker

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Christian Becker
Betreuer/in:	Dr. rer. nat. Frank Dürr
Beginn am:	2. November 2023
Beendet am:	2. Mai 2024

Kurzfassung

Time-Sensitive Networking (TSN) ist eine Netzwerktechnologie, die deterministische Echtzeitkommunikation in Ethernet-Netzwerken ermöglicht. Diese Technologie findet insbesondere in Netzwerken, deren zuverlässige Kommunikation für sicherheitskritische Aufgaben notwendig ist, Anwendung.

Die manuelle Konfiguration von TSN-Netzwerken ist jedoch häufig ein aufwändiger Prozess, der die separate Konfiguration der einzelnen Netzwerkswitches mithilfe unterschiedlicher Konfigurationswerkzeuge beinhaltet. Diese Arbeit widmet sich der Aufgabe, diesen manuellen Konfigurationsprozess zu vereinfachen. Eine Möglichkeit, um die Switches eines TSN-Netzwerks zu konfigurieren, ist der Einsatz eines sogenannten Centralized Network Controller (CNC). Um die manuelle Konfiguration von TSN-Netzwerken zu vereinfachen, wird in dieser Arbeit ein web-basierter CNC entworfen und implementiert.

Die in dieser Arbeit entworfene Web-based Centralized Network Controller (WebCNC)-Architektur legt besonderen Wert auf die Erweiterbarkeit des CNC, um eine Ausstattung des WebCNC mit weiteren Funktionalitäten bei geringem Aufwand zu ermöglichen. Dies wird durch eine modulare, komponentenbasierte Architektur des CNC erreicht, deren Struktur sich an einer Drei-Schichten-Architektur mit Präsentationsschicht, Anwendungsschicht und Datenschicht orientiert. Da es sich um einen web-basierten CNC handelt, teilen sich diese Schichten auf Frontend und Backend auf. Die Präsentationsschicht zur Interaktion mit dem Nutzer ist in einem Frontend verortet und kommuniziert zur Realisierung von Nutzeranfragen mit dem Backend über eine HTTP-Schnittstelle. Das Backend beinhaltet die Anwendungs- und Datenschicht des WebCNC und dient als Brücke zwischen dem Frontend und den Netzwerkswitches.

Ein weiterer Kernaspekt der WebCNC-Architektur ist die Portabilität. So basiert der WebCNC auf Standards, die herstellerübergreifend genutzt werden, um Kompatibilität mit Geräten verschiedener Hersteller zu erreichen. Der WebCNC selbst wird mithilfe von Standard-Technologien realisiert, die eine Ausführung des WebCNC auf beliebigen gängigen Systemen ermöglichen sollen. Außerdem ist die Benutzerfreundlichkeit ein relevanter Faktor für die getroffenen Entwurfsentscheidungen. Als Web-Anwendung soll es Nutzern möglich sein, den WebCNC ohne eine lokale Installation von spezieller Software zu nutzen. Um mit dem WebCNC zu interagieren, sollen Benutzer lediglich einen Webbrowser benötigen.

Neben der WebCNC-Architektur wird auch ein Software-Prototyp als Proof-of-Concept vorgestellt, der diese Architektur in der Praxis umsetzt und zur Validierung der Architekturziele genutzt wird. Um die Erweiterbarkeit der entwickelten Architektur zu evaluieren, werden in dieser Arbeit einige denkbare Erweiterungen für den WebCNC vorgestellt. Es wird betrachtet, wie verschiedene Klassen von Erweiterungen in den bestehenden WebCNC-Prototypen integriert werden können. Die gewünschte Portabilität erreicht der Prototyp im Frontend durch den Einsatz einer generischen Webanwendung, die von einem beliebigen Webserver an Clients ausgeliefert werden kann. Das Backend wird als Java-Anwendung realisiert, um eine Ausführung des Backends auf allen Systemen, für die eine Java-Laufzeitumgebung existiert, zu ermöglichen. Die Benutzerfreundlichkeit wird erreicht, indem eine graphische Nutzeroberfläche als Frontend entwickelt wird. Diese Oberfläche wird in mehrere Ansichten unterteilt, die verschiedene Aufgaben erfüllen. Der in dieser Arbeit entworfene, klar definierte Navigationsfluss zwischen den Ansichten soll eine intuitive Nutzung des WebCNC ermöglichen.

Inhaltsverzeichnis

1 Einleitung **13**

2 Hintergrund **17**
2.1 Simple Network Management Protocol (SNMP) 17
2.2 Network Configuration Protocol (NETCONF) 21
2.3 Time-Sensitive Networking (TSN) 21
2.4 Link Layer Discovery Protocol (LLDP) 25
2.5 Hypertext Transfer Protocol (HTTP) 26

3 Verwandte Arbeiten **29**
3.1 TSN-Network-Controller 29
3.2 SDN-Network-Controller 32

4 Systemmodell und Problemstellung **35**
4.1 Systemmodell 35
4.2 Problemstellung 36

5 Architektur **37**
5.1 Überblick 37
5.2 Präsentationsschicht 38
5.3 Anwendungsschicht 39
5.4 Datenschicht 39

6 Entwurf und Implementierung des Backends **41**
6.1 Verwendete Technologien und Bibliotheken 41
6.2 Typ-Definitionen 42
6.3 netStateCache-Komponente 46
6.4 switchCredStore-Komponente 48
6.5 switchComms-Komponente 49
6.6 setup-Komponente 53
6.7 restcontroller-Komponente 55

7 Entwurf und Implementierung des Frontends **59**
7.1 Funktionsweise einer Web-Anwendung 59
7.2 Verwendete Technologien, Bibliotheken und Werkzeuge 60
7.3 Datenstrukturen 60
7.4 Ansichten und Navigationsfluss 61

8 Validierung **69**
8.1 Funktionstests 69

8.2 Architekturziele 74

9 Fazit und Ausblick **81**
9.1 Fazit 81
9.2 Ausblick 82

Literaturverzeichnis **83**

Abbildungsverzeichnis

2.1 Architektur eines SNMP-Netzwerks 18
2.2 Teilbaum der MIB-II OID-Struktur [MS05] 19
2.3 Architektur eines TAS-Ports [DN16] 23
2.4 Zustandsmaschinen des Time-Aware Shaper (TAS) [IEEE16b] 23
2.5 Aufbau eines Link-Layer Discovery Protocol (LLDP)-Pakets für IEEE 802.3 Ethernet-Netzwerke [IEEE16a] 25

3.1 CNC-Architektur [ÁSP+22] 30
3.2 Microservice-Architektur der OpenCNC [CJ] 31

4.1 Systemmodell 35

5.1 Architektur-Übersicht 38

6.1 Klassendiagramm der Backend-Klassen für Switch-Daten 43
6.2 Hilfsklassen und Enums des Backends 46
6.3 Interface der netStateCache-Komponente 47
6.4 Interface der switchCredStore-Komponente 48
6.5 Aufbau der switchComms-Komponente 50
6.6 Sequenzdiagramm der WebCNC-Initialisierung 53
6.7 Sequenzdiagramm einer AddSwitchController-Anfrage 56
6.8 Sequenzdiagramm einer SavePortController-Anfrage 57
6.9 Sequenzdiagramm einer ReloadBackendController-Anfrage 58

7.1 Typdefinitionen im Frontend 61
7.2 Zustandsdiagramm der Frontend-Ansichten 62
7.3 Switchübersicht-Ansicht 63
7.4 Netzwerkvisualisierung-Ansicht 64
7.5 Portkonfiguration-Ansicht 65
7.6 SwitchHinzufügen-Ansicht 66

8.1 Versuchsaufbau 70
8.2 Netzwerkvisualisierung des WebCNC 71
8.3 Switchübersicht des WebCNC 71
8.4 LLDP-Informationen von Switch1 72
8.5 LLDP-Informationen von Switch2 72
8.6 Port-Konfiguration des Switch-Webinterfaces 72
8.7 Gate Control List (GCL)-Konfiguration des Switch-Webinterfaces 73
8.8 TSN-Konfiguration des Ports im WebCNC 73
8.9 TSN-Konfiguration des Ports im WebCNC nach Konfigurationsänderung 75

8.10 Port-Konfiguration des Switch-Webinterfaces nach Konfigurationsänderung . . . 75
8.11 GCL-Konfiguration des Switch-Webinterfaces nach Konfigurationsänderung . . . 76

Abkürzungsverzeichnis

AES Advanced Encryption Standard. 21

API Application Programming Interface. 39

CA Certificate Authority. 27

CBS Credit-Based Shaper. 25

CLI Command Line Interface. 31

CNC Centralized Network Controller. 3, 14

CORS Cross-Origin Resource Sharing. 54

CPS Cyber-Physical Systems. 13

CPU Central Processing Unit. 41

CSS Cascading Style Sheet. 26

CUC Centralized User Controller. 14

DES Data Encryption Standard. 21

FCS Frame Check Sequence. 25

FIFO First In First Out. 22

GCL Gate Control List. 7, 65

gNMI gRPC Network Management Interface. 30

GUI Graphical User Interface. 31

HTML HyperText Markup Language. 26

HTTP Hypertext Transfer Protocol. 15, 17, 26

HTTP/1.1 Hypertext Transfer Protocol version 1.1. 26

HTTP/2 Hypertext Transfer Protocol version 2. 26

HTTP/3 Hypertext Transfer Protocol version 3. 26

HTTPS Hypertext Transfer Protocol Secure. 27

IEEE Institute of Electrical and Electronics Engineers. 13

IETF Internet Engineering Task Force. 17

IP Internet Protocol. 17

JRE Java Runtime Environment. 41, 74

JSON JavaScript Object Notation. 26

JVM Java Virtual Machine. 41

LAN Local Area Network. 25

LLDP Link-Layer Discovery Protocol. 7, 14, 17, 25

LLDPDU Link Layer Discovery Protocol Data Unit. 25

MAC Media Access Control. 25

MD Managed Device. 17

MD5 Message-Digest 5. 21

MIB Management Information Base. 17

NETCONF Network Configuration Protocol. 14, 17, 21

NMS Network Management System. 17

OID Objekt Identifier. 18

PCP Priority Code Point. 22

PDU Protocol Data Unit. 20

PTP Precision Time Protocol. 24

REST Representational State Transfer. 55

RFC Request For Comment. 17

RPC Remote Procedure Call. 21

SDN Software-Defined Networking. 15

SHA1 Secure Hash Algorithm 1. 21

SM State Machine. 24

SMI Structure of Management Information. 18

SMIv1 Structure of Management Information version 1. 18

SMIv2 Structure of Management Information version 2. 18

SNMP Simple Network Management Protocol. 14, 17

SNMPv3 Simple Network Management Protocol version 3. 17

SPA Single Page Application. 59

SR Stream Reservation. 25

SSH Secure Shell Protocol. 21

TAS Time-Aware Shaper. 7, 14, 17, 22

TCP Transmission Control Protocol. 18

TLS Transport Layer Security. 27

TLV Type-Length Value. 25

TSN Time-Sensitive Networking. 3

UDP User Datagram Protocol. 18

UI User Interface. 60

UML Unified Modeling Language. 42

WebCNC Web-based Centralized Network Controller. 3, 14

1 Einleitung

Im Rahmen der zunehmenden Automatisierung in Industrie und Alltag ist es notwendig, Steuerungs- und Informationsdaten zwischen einer Vielzahl an beteiligten Geräten auszutauschen. Viele dieser Anwendungen sind sicherheitskritisch, insbesondere im Bereich der Cyber-Physical Systems (CPS). Dies sind Systeme, in denen mechanische Komponenten durch Softwarekomponenten, die in einem Netzwerk miteinander verbunden sind, gesteuert werden.

Im Rahmen dieser Systeme gibt es häufig sicherheitskritische Datenströme, deren Versagen nicht nur unprofitabel oder unkomfortabel, sondern sogar gefährlich für Menschen in ihrer Umgebung sein können. So ist beispielsweise beim autonomen Fahren die Reaktionszeit der Steuerungssysteme ein sicherheitskritisches System. Hier ist es unabdingbar, dass Sensordaten von Kameras, LiDAR-Sensoren, Ultraschallsensoren und anderen Datenquellen die Fahrzeugsteuerung rechtzeitig erreichen, um den sicheren Betrieb des Fahrzeugs zu gewährleisten und nicht in Unfälle mit anderen Verkehrsteilnehmern verwickelt zu werden.

Um den anforderungsgerechten Austausch von kritischen Datenströmen zu gewährleisten, ist eine deterministische Kommunikation mit Echtzeitgarantien an die beteiligten Systeme notwendig. Um dies umzusetzen, wird von der Institute of Electrical and Electronics Engineers (IEEE) Time-Sensitive Networking (TSN) Task Group [IEEE24] eine Reihe von Standards entwickelt, die es erlauben, mit TSN deterministische Kommunikation in Netzwerken zu gewährleisten. So werden in IEEE 802.1Q [IEEE22b] verschiedene Erweiterungen für IEEE 802.3 (Ethernet)-Netzwerke [IEEE22a] standardisiert, die es ermöglichen, TSN in Ethernet-Netzwerken umzusetzen.

TSN findet in vielen verschiedenen Bereichen Anwendung. Wie bereits motiviert, wird TSN genutzt, um in autonomen Fahrzeugen sicherzustellen, dass Informationen über die Umgebung des Fahrzeugs rechtzeitig an der Steuerungseinheit zur Verfügung stehen [SZ18]. Auch im Rahmen der Industrie 4.0 wird TSN eingesetzt, um Systeme in der Fertigung von Produkten zu steuern und somit die Produktionsprozesse zu optimieren [LS19]. Mit Blick auf die Industrie 4.0 ist auch die Umsetzung von TSN in drahtlosen Netzwerken mittels 5G-Technik Gegenstand der Forschung [LLM+20]. Auch in der Energieversorgung kann TSN genutzt werden, um die Robustheit und Effizienz von Stromnetzen durch den genaueren Zeitabgleich der Daten verschiedener Messgeräte zu verbessern [TD21]. Im Bereich der Medizin kann deterministische Kommunikation eingesetzt werden, um eine gelungene Behandlung der Patienten mit modernen Techniken zu ermöglichen [RKS+20]. So werden für Operationen häufig Computer-gesteuerte Geräte eingesetzt, deren korrekte Ansteuerung für den Erfolg oder Misserfolg einer Operation entscheidend sein kann.

TSN macht sich das Konzept der Unterscheidung von Daten- und Kontrollebene in Kommunikationsnetzwerken zu Nutze. Die Datenebene ist für die Weiterleitung von Nachrichten in Netzwerken zuständig. Im Fall eines Ethernet-Netzwerks kann dies ein Netzwerkswitch sein, der eingehende Pakete nach vorgegebenen Regeln an einen anderen Port des Switches weiterleitet. Die Kontrollebene

hingegen steuert, wie die Datenebene Nachrichten weiterleitet. Unter anderem ist die Konfiguration von Routing-Tabellen, die bestimmen, welche Pakete wohin weitergeleitet werden sollen, Aufgabe der Kontrollebene.

Um deterministische Kommunikation zu realisieren, nutzt TSN die Kontrollebene von Netzwerken. So standardisiert IEEE 802.1Qbv [IEEE16b] einen Time-Aware Shaper (TAS), der spezielle Scheduling-Tabellen einsetzt und konfiguriert, die bestimmten Netzwerkverkehrsklassen Zeiträume zur Kommunikation reservieren. Damit kann Datenflüssen, deren Nachrichten diesen Klassen angehören, deterministische Kommmunikation zugesichert werden. Um dies zu erreichen, wird der TAS auf Netzwerkswitches implementiert.

In komplexen Netzwerken, die aus mehreren Switches bestehen, müssen zum Erreichen deterministischer Kommunikation die TSN-Konfigurationen der Netzwerkswitches aufeinander abgestimmt werden. Ein Ansatz zum Erzielen einer gemeinsamen TSN-Konfiguration mehrerer Netzwerkswitches, der auf einer zentralisierten Lösung basiert, ist der Einsatz eines Centralized Network Controller (CNC). Ein CNC ist ein System, das einen Überblick über die Topologie des gesamten Netzwerkes hat und die Konfigurationen der Netzwerkswitches kennt. Ein TSN-CNC ist in der Lage, eine aufeinander abgestimmte TSN-Konfiguration der einzelnen Netzwerkswitches zu definieren und diese auf den Switches zu aktivieren. CNCs können für eine manuelle Konfiguration des Netzwerkes eingesetzt werden. Es ist jedoch auch möglich, mithilfe eines Centralized User Controller (CUC) automatisch auf Basis von Netzwerkverkehrsanforderungen der Netzwerkteilnehmer ein passendes TSN-Scheduling zu bestimmen, das ohne menschliches Eingreifen die angeforderten, deterministischen Datenströme ermöglicht.

Diese Arbeit befasst sich mit dem Entwurf und der Implementierung eines CNC zur manuellen Konfiguration von Netzwerken mit TSN-fähigen Netzwerkswitches. Es soll ein Web-based Centralized Network Controller (WebCNC) für die Konfiguration der TSN-Switches im Netzwerk entworfen und implementiert werden, der es ermöglicht, manuell alle Switches eines Netzwerks mittels einer Web-Browser-Anwendung zu konfigurieren. Hierfür müssen zwei miteinander kommunizierende Systeme entworfen werden: Ein Frontend, das im Browser des Nutzers für die Interaktion mit dem CNC genutzt wird, und ein Backend, das die Kommunikation zwischen Switch und Frontend umsetzt und Daten über den Zustand des Netzwerkes vorhält. Zentrale Aspekte der Arbeit sind neben der Kernfunktionalität die Modularität, Portabilität und Benutzerfreundlichkeit des WebCNC. Modularität ist zentral für die Wiederverwendbarkeit und Erweiterung der vorgestellten Architektur und des Prototypen. Die Ergebnisse dieser Arbeit sollen auch für die Nutzung in Netzwerken, deren Switches andere Netzwerkkonfigurationsprotokolle verwenden, anpassbar sein. Außerdem soll die Erweiterung der CNC um weitere Funktionalitäten ohne eine starke Modifikation des bestehenden Systems möglich sein. Der Einsatz des WebCNC soll ohne hohe Infrastruktur-Anforderungen auf gängigen Systemen möglich sein. Da der WebCNC eine manuelle Konfiguration des Netzwerks durch Menschen vorsieht, ist auch die Nutzererfahrung von entscheidender Bedeutung. Der WebCNC soll – mit einem Grundverständnis von TSN-Konfigurationen und den anzupassenden Parametern – intuitiv nutzbar sein. Auch sollen dem Nutzer Informationen über das vorliegende Netzwerk präsentiert werden, damit eine manuelle Konfiguration des Netzwerkes ohne detaillierte A-Priori-Kenntnisse der gesamten Netzwerkstruktur vorgenommen werden kann.

Die Arbeit ist in 9 Kapitel aufgeteilt, in denen die verschiedenen Aspekte der Arbeit beleuchtet werden. Kapitel 2 befasst sich mit dem technischen Hintergrund des CNC und stellt die verwendeten Netzwerktechnologien vor. Neben TSN werden mit dem Simple Network Management Protocol (SNMP), dem Network Configuration Protocol (NETCONF), dem Link-Layer Discovery Protocol

(LLDP) und dem Hypertext Transfer Protocol (HTTP) weitere, für den CNC wichtige Protokolle vorgestellt. In Kapitel 3 werden Arbeiten vorgestellt, die sich mit verwandten Fragestellungen beschäftigen oder ähnliche Netzwerk-Controller bereitstellen. Neben anderen TSN-CNCs werden hier auch Software-Defined Networking (SDN)-Controller betrachtet. Kapitel 4 beschreibt ein Systemmodell, das als Basis für die Implementierung des WebCNC dient, und erläutert die dieser Arbeit zugrundeliegende Problemstellung sowie die Ziele der Arbeit genauer. In Kapitel 5 wird ein komponentenbasierter Architektur-Entwurf des CNC skizziert und die Interaktion der verschiedenen Komponenten betrachtet. Kapitel 6 stellt das Backend und seine Implementierung im Detail vor. Zudem werden alle wesentlichen Design-Entscheidungen der Backend-Architektur beleuchtet. Neben der reinen Funktionalität werden insbesondere auch Entwurfsentscheidungen betrachtet, welche die Erweiterbarkeit des WebCNC gewährleisten. In Kapitel 7 wird das Frontend, seine Implementierung und die Interaktion mit dem Nutzer betrachtet. Hier liegt der Fokus nicht nur auf der Interaktion der Komponenten, sondern auch auf den Überlegungen hinsichtlich der Nutzererfahrung. Kapitel 8 befasst sich mit der Validierung des Software-Prototypen. Zur Validierung der Funktionalität werden mehrere Konfigurationsszenarien definiert und dieses anschließend mithilfe des Prototypen durchgeführt. Außerdem wird diskutiert, wie die in Kapitel 4 vorgestellten Architekturziele erfüllt wurden. Kapitel 9 zieht ein Fazit über die gesamten Ergebnisse der Arbeit und stellt Perspektiven für weitere Arbeiten vor, die aus den Ergebnissen dieser Arbeit hervorgehen. Insbesondere werden Erweiterungsmöglichkeiten für den WebCNC diskutiert. Neben der erwarteten Erweiterbarkeit um weitere Funktionalitäten und andere Netzwerkkonfigurationsprotokolle stellt sich hierbei heraus, dass die Architektur des WebCNC auch eine Erweiterung um einen CUC, der die automatische Konfiguration des TSN-Netzwerks auf Basis von Datenstromanforderungen der Netzwerkteilnehmer erlaubt, möglich ist.

2 Hintergrund

Der in dieser Arbeit entworfene CNC und der daraus entstandene Software-Prototyp basieren auf einer Reihe grundlegender Netzwerktechnologien und -standards. Dieses Kapitel widmet sich der Erläuterung der Technologien, die eine maßgebliche Rolle für den Entwurf des CNC spielen. Zunächst wird das Simple Network Management Protocol (SNMP) eingeführt, welches zur standardisierten Kommunikation mit den Netzwerkswitches eingesetzt wird. Dieses wird kurz mit dem Network Configuration Protocol (NETCONF), einer Alternative zu SNMP, verglichen. Anschließend wird TSN und insbesondere der in IEEE 802.1Qbv [IEEE16b] definierte Time-Aware Shaper (TAS) dargestellt, dessen Konfiguration Kernziel dieser Arbeit ist. Außerdem wird das Link-Layer Discovery Protocol (LLDP) vorgestellt, ein Protokoll zum Austausch von Geräteinformationen zwischen benachbarten Geräten, welches zentral für die Erfassung der zu präsentierenden Netzwerktopologie ist. Schließlich wird noch das Hypertext Transfer Protocol (HTTP) vorgestellt. HTTP bietet eine standardisierte Basis für die Kommunikation zwischen dem Frontend und dem Backend des CNC.

2.1 Simple Network Management Protocol (SNMP)

SNMP ist ein Protokoll zum Überwachen und Verwalten von Netzwerkgeräten über ein Internet Protocol (IP)-Netzwerk, das von der Internet Engineering Task Force (IETF) [IETF24] standardisiert wurde. Die aktuelle Version des Protokolls, Simple Network Management Protocol version 3 (SNMPv3), wird in Request For Comment (RFC) 3410 [CMPS02] standardisiert. Details zu einzelnen Aspekten von SNMP sind in weiteren RFCs definiert [BW02; FLWR03; HCWP02; HWP02; MBM04; MLS02; MSPM99a; MSPM99b; MSPM99c; MWP02; Pre02a; Pre02b; RM90; RM91; Ros91]. Die folgenden Ausführungen zu SNMP basieren auf diesen RFCs. Im folgenden Abschnitt wird zunächst die grundlegende Architektur eines SNMP-Systems erklärt. Anschließend werden die Datenstrukturen – in SNMP auch Management Information Base (MIB) genannt – in denen zu speichernde Objekte bzw. Variablen definiert und strukturiert werden, näher betrachtet. Daraufhin werden die verschiedenen von SNMP unterstützten Operationen zur Verwaltung von Geräten über Netzwerke erläutert. Zuletzt werden noch die mit SNMPv3 eingeführten Sicherheitsfeatures betrachtet.

2.1.1 SNMP-Architektur

In einem Netzwerk, dessen Geräte mittels SNMP verwaltet werden, gibt es mehrere miteinander interagierende Akteure. So übernimmt ein Network Management System (NMS) die Überwachung und Verwaltung der Geräte des Systems, während ein Managed Device (MD) ein solches, verwaltetes Gerät darstellt. Ein Überblick über diese Architektur ist in Abbildung 2.1 dargestellt.

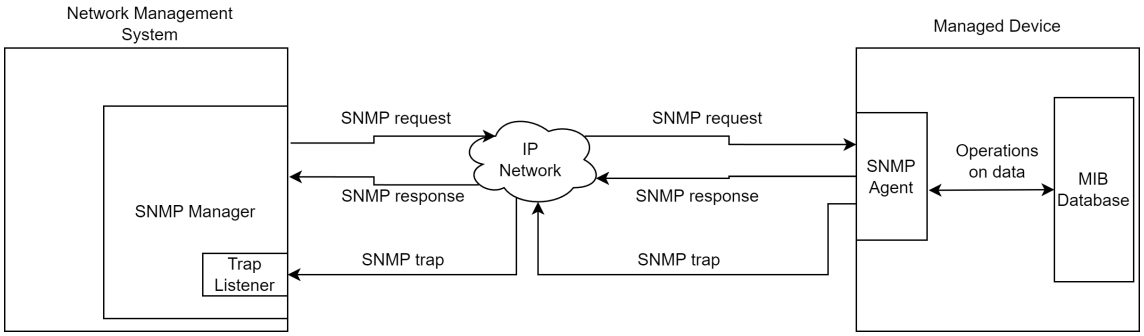

Abbildung 2.1: Architektur eines SNMP-Netzwerks

Ein NMS ist ein System, das andere Geräte im Netzwerk mittels SNMP verwaltet und überwacht. Dafür läuft auf dem NMS ein SNMP Manager, der in der Lage ist, mittels des User Datagram Protocol (UDP) oder des Transmission Control Protocol (TCP) SNMP-Anfragen an ein MD zu senden. In diesen Anfragen wird angegeben, welche Objekt-Werte abgefragt oder gesetzt werden sollen. Ein MD besitzt eine Datenbank für unterstützte Objekte, deren Werte Informationen über den Gerätestatus oder die Gerätekonfiguration darstellen. Das NMS kann diese Werte nicht direkt auslesen oder manipulieren. Eintreffende SNMP-Anfragen werden von einem auf dem MD laufenden SNMP-Agent entgegengenommen, der überprüft, ob die angeforderten Operationen gültig sind – so können zum Beispiel zu niedrige oder zu hohe Werte bei Schreiboperationen abgelehnt werden – und ob die in der SNMP-Anfrage enthaltenen Zugangsdaten korrekt sind und Zugriff auf die geforderten Objekte erlauben. Ist die Operation validiert und ausgeführt, wird vom SNMP-Agent eine SNMP-Antwort an den SNMP-Manager gesendet. Diese Antwort enthält eine Bestätigung und im Falle von Leseoperation die angeforderten Werte. Konnte die Operation vom SNMP-Agent nicht validiert werden, wird eine Antwort mit einer Fehlerbeschreibung zurückgesendet.

Auch ist es möglich, bei einem MD so genannte SNMP-Traps zu registrieren. Traps sind eigenständig vom MD an das NMS gesendete Werte aus der MIB-Datenbank, ohne dass unmittelbar zuvor eine SNMP-Anfrage für diesen Wert eingegangen ist. Stattdessen wird das Senden von Traps durch vordefinierte Konditionen ausgelöst. So kann eine Trap beispielsweise gesendet werden, wenn das MD einen Neustart durchführt oder wenn es ein Firmware-Update bekommt.

2.1.2 Management Information Bases (MIBs)

MIBs werden in SNMP dafür genutzt, die Datenstrukturen, in denen mit SNMP verwaltbare Objekte und ihre Werte gespeichert werden, zu definieren. Geschrieben werden MIBs in einer speziellen Sprache, der Structure of Management Information (SMI). Die aktuelle Version, Structure of Management Information version 2 (SMIv2), wird in RFC 2578 [MSPM99b], RFC 2579 [MSPM99c] sowie RFC 2580 [MSPM99a] definiert. Um ältere MIBs zu unterstützen, ist SNMPv3 außerdem kompatibel mit der älteren Structure of Management Information version 1 (SMIv1), die in RFC 1155 [RM90], RFC 1212 [RM91] sowie RFC 1215 [Ros91] standardisiert wird.

Die Objekte in SNMP sind hierarchisch in einer Baumform strukturiert. Zur Adressierung von Objekten wird jedem Objekt ein Objekt Identifier (OID) zugewiesen. Ein Beispiel dieser Baumstruktur ist in Abbildung 2.2 dargestellt. Ein Objekt, das kein Blattknoten ist, besitzt keinen Wert, sondern wird nur für die Strukturierung der MIB genutzt. Blattknoten hingegen besitzen einen

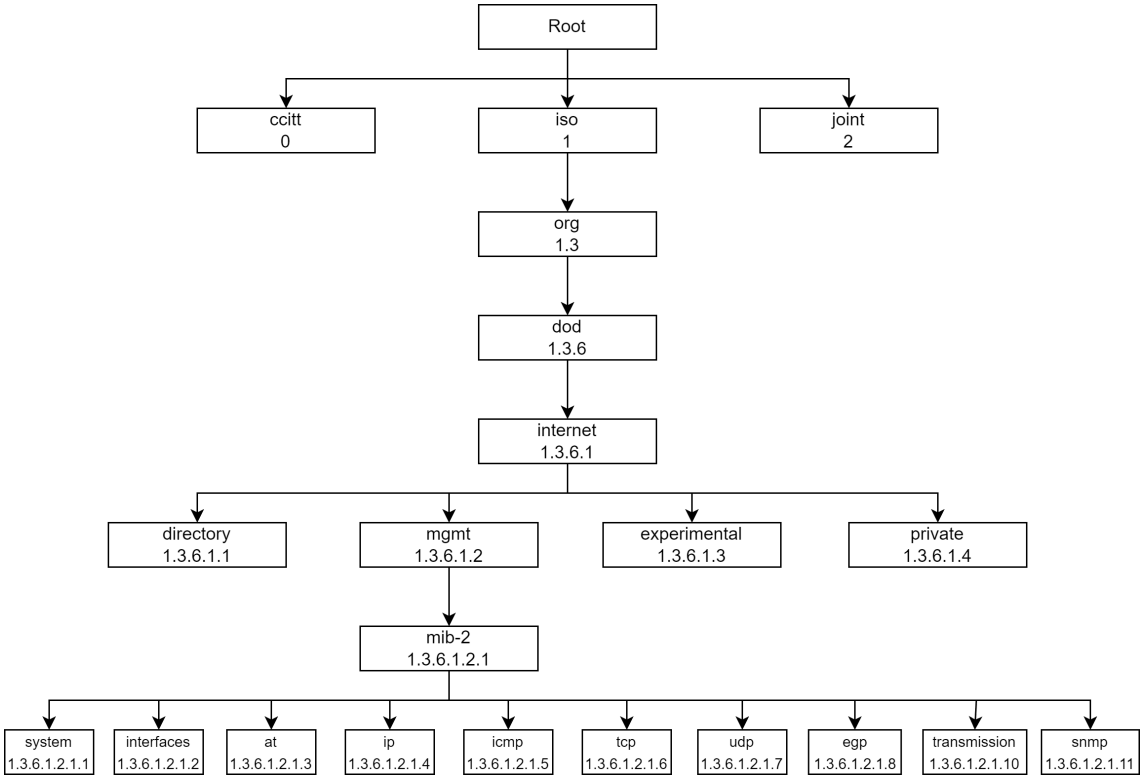

Abbildung 2.2: Teilbaum der MIB-II OID-Struktur [MS05]

Wert von einem in der SMI definierten Datentyp. Außerdem besitzt ein Objekt ein Zugriffsattribut. Dieses Attribut bestimmt, ob und in welchem Kontext der Wert des Objekts mittels SNMP gelesen oder geschrieben werden darf. Bei Nicht-Blattknoten ohne Wert ist dieses Attribut stets als nicht zugänglich gesetzt. Um die verschiedenen Zugriffs-Stufen abzudecken, kann das Attribut gemäß SMIv2 [MSPM99b] die Werte *not-accessible*, *accessible-for-notify*, *read-only*, *read-write* oder *read-create* annehmen.

Zur Adressierung einer dynamischen Anzahl von Objekten gleichen Typs gibt es in SNMP-MIBs Tabellen. Für die Strukturierung von Tabellen werden zunächst Sequenzen eingeführt. Sequenzen sind Folgen von mehreren Einzel-Wert-Objekten, die über eine Basis-OID der Sequenz und die Position des Objekts in der Sequenz adressiert werden. So wird die OID des n-ten Objekts in einer Sequenz als Konkatenation der Sequenz-OID und der Position definiert: *<SequenzOID>.n* .

Tabellen werden genutzt, um eine beliebige Anzahl gleicher Sequenzen mit unterschiedlichen Werten zu speichern. Jede Zeile einer Tabelle besteht aus einer solchen Sequenz. Jede Tabelle besitzt eine Basis-OID. Die Spalten der Tabelle beinhalten die Objekte der Sequenzen - so stehen in der n-ten Spalte die n-ten Objekte der gespeicherten Sequenzen. Spalten werden mittels Konkatenation der Tabellen-Basis-OID und der Spaltenposition adressiert. Um die gewünschte Zeile – also die spezifische Sequenz – zu adressieren, besitzt jede Zeile einen Index. Dieser Index wird aus einer eindeutigen Kombination von Sequenz-Werten gebildet. Stellen die Werte der Sequenz-Objekte n und m zusammen einen eindeutigen Identifikator für die Sequenz dar, bilden diese in Konkatenation

den Index <*Wert(n)*>.<*Wert(m)*> der Zeile. Damit lassen sich einzigartige OIDs für jedes Objekt der Tabelle ableiten. Ein Tabellen-Objekt wird als Konkatenation von Tabellen-OID, Spaltennummer und Zeilenindex adressiert: <*Tabellen-OID*>.<*Spaltennummer*>.<*Zeilenindex*> .

2.1.3 SNMP-Nachrichten

Da SNMP ein Netzwerkmanagementprotokoll ist, findet die Kommunikation zwischen SNMP Managern und SNMP Agenten mittels Nachrichten über ein Netzwerk statt. Eine SNMP-Nachricht enthält, abhängig von der SNMP-Version, verschiedene Felder, die unter anderem der Authentifikation dienen. Das Kernelement einer SNMP-Nachricht ist die enthaltene Protocol Data Unit (PDU). Die PDU definiert, welche Protokoll-Operation vom Empfänger ausgeführt werden soll. Außerdem enthält sie die dazu notwendigen Informationen, wie zum Beispiel die OID eines angefragten Objekt-Wertes. Für SNMPv3 definiert RFC 3416 [Pre02b] die möglichen Protokoll-Operationen und die entsprechenden PDU-Typen. Im folgenden Abschnitt werden die standardisierten PDU-Typen behandelt.

Die *GetRequest-PDU* wird von einem SNMP Manager an einen SNMP Agent gesendet. Sie kann eine oder mehrere OIDs beinhalten, deren Werte der SNMP Manager abfragen möchte. Die *GetBulkRequest-PDU* dient der effizienten Abfrage von mehreren Instanzen eines Objektes. Dies findet im Kontext von Tabellen Anwendung. So kann eine bestimmte Anzahl an Einträgen aus einer Spalte einer Tabelle abgefragt werden, ohne die OID jedes einzelnen Eintrages angeben zu müssen. Stattdessen wird die Spalte, die Startreihe und die Anzahl nachfolgender Instanzen, die aus der Tabelle abgefragt werden sollen, angegeben. Die *GetNextRequest-PDU* spezifiziert eine OID und erwartet als Antwort die in der OID-Hierarchie nachfolgende OID und, wenn es ein Blattknoten ist, den Wert des damit adressierten Objektes. Dies kann genutzt werden, um eintragsweise durch Tabellen zu navigieren, oder um die OID-Struktur der MIB herauszufinden. Die *SetRequest-PDU* dient nicht dem Abfragen, sondern dem Setzen eines Wertes. So enthält die PDU die OID des zu setzenden Wertes und den neuen Wert. Die *SNMPv2-Trap-PDU* dient der Zustellung von SNMP-Traps von einem SNMP Agent zu einem SNMP Manager. Sie findet auch in SNMPv3 Anwendung. Der Agent erwartet keine Antwort auf eine solche PDU, daher kann ein möglicher Verlust des Pakets nicht entdeckt werden. Dieses Problem löst die *InformRequest-PDU*. Wie auch die *SNMPv2-Trap-PDU* dient sie der Zustellung von Traps. Jedoch erwartet der Agent auf eine *InformRequest-PDU* eine Bestätigung des Managers. Wird diese Bestätigung nicht erhalten, kann die Trap vom Agent erneut gesendet werden. Mit Ausnahme der *SNMPv2-Trap-PDU* erwartet der Sender der Nachricht bei allen PDU-Typen eine Antwort vom Empfänger. Diese Antwort wird mittels einer *Response-PDU* gesendet. Sie beinhaltet etwaige angefragte OIDs und Werte. Außerdem besitzt sie mit dem *error-status*- und dem *error-index*-Feld die Möglichkeit, Informationen über bei der Ausführung der Anfrage aufgetretene Fehler zu melden.

2.1.4 SNMPv3-Sicherheitsfeatures

Das primäre Vorteil von SNMPv3 gegenüber älteren SNMP-Versionen sind neue Sicherheitsfeatures. Frühere Versionen von SNMP kommunizieren unverschlüsselt mit einfachen *community strings* zur Authentifikation [CMRW96]. SNMPv3 hingegen unterstützt verschlüsselte Kommunikation mit einem nutzerbasierten Authentifikationsmodell.

Das in RFC 3414 [BW02] standardisierte, nutzerbasierte Authentifikationsmodell bietet zusammen mit dem ansichtsbasierten Sichtbarkeitsmodell aus RFC 3415 [MWP02] die Möglichkeit, verschiedenen Nutzern granularen Lese- oder Schreibzugriff auf bestimmte MIBs oder auch Teilbäume von MIBs zu gewähren. Zur Authentifikation und Integritätsprüfung von SNMP-Nachrichten werden gemäß RFC 3414 [BW02] die Algorithmen Message-Digest 5 (MD5) [Riv92] sowie Secure Hash Algorithm 1 (SHA1) [DJ01] unterstützt. Neben der Authentifikation und Integritätsprüfung bietet SNMPv3 auch Mechanismen zur Verschlüsselung der SNMP-Kommunikation an, um zu verhindern, dass der Nachrichtenverkehr von Dritten abgehört werden kann. Unterstützt wird hierzu gemäß RFC 3414 der Data Encryption Standard (DES) [Nat88]. Zudem wird nach RFC 3826 [MBM04] auch der Advanced Encryption Standard (AES) [DBN+01] als Verschlüsselungsalgorithmus unterstützt.

2.2 Network Configuration Protocol (NETCONF)

Das Network Configuration Protocol (NETCONF) ist ein in RFC 6241 [EBBS11] standardisiertes Netzwerkkonfigurationsprotokoll, das alternativ zu SNMP eingesetzt werden kann.

Der Funktionsumfang von NETCONF ähnelt dem von SNMP. So unterstützen beide Protokolle Request/Response-Anfragen, bei denen der Controller Anfragen an Geräte sendet und entsprechende Antworten erhält. Ebenfalls unterstützen beide Protokolle ein System für Pub/Sub-Benachrichtigungen. Bei SNMP sind dies die in Kapitel 2.1.1 eingeführten Traps. Bei NETCONF gibt es einen ähnlichen Mechanismus für sogenannte Notifications. Eine weitere Ähnlichkeit beider Protokolle ist, dass Verbindungen zwischen Controller und verwaltetem Gerät vom Controller aufgebaut werden.

Jedoch gibt es auch einige markante Unterschiede zwischen den beiden Protokollen. Zum einen basiert NETCONF nicht auf MIBs, sondern nutzt zum Lesen und Setzen von Werten Remote Procedure Call (RPC)-Funktionen, die mittels in YANG-Modellen definierte XML-Dateien zwischen den Geräten ausgetauscht werden. Verschieden ist auch die Realisierung des Nachrichtenaustauschs zwischen Controller und verwaltetem Gerät. Während SNMP primär auf UDP und TCP setzt, wird NETCONF-Kommunikation häufig über eine Secure Shell Protocol (SSH)-Verbindung zwischen Controller und verwaltetem Gerät umgesetzt. Damit ist NETCONF-Kommunikation über SSH inhärent mit den umfangreichen Sicherheitsfeatures von SSH ausgestattet, während SNMP – insbesondere SNMP-Versionen vor SNMPv3 – hier große Mängel aufweist.

2.3 Time-Sensitive Networking (TSN)

TSN bezeichnet eine Reihe von Standards, die von der IEEE Time-Sensitive Networking (TSN) Task Group [IEEE24] entwickelt werden. Diese Standards beschreiben Erweiterungen für IEEE 802.3 (Ethernet) [IEEE22a], die deterministische Kommunikation über Ethernet-Netzwerke realisieren. Damit ist es möglich, Echtzeitgarantien für die Nachrichtenzustellung zu gewährleisten. Viele dieser Standards sind in IEEE 802.1Q (Bridges and Bridged Networks) [IEEE22b] eingebracht worden. Die wichtigsten dieser Standards werden im Folgenden betrachtet, insbesondere IEEE 802.1Qbv (Enhancements for Scheduled Traffic) [IEEE16b], auf dem der Fokus dieser Arbeit liegt.

2.3.1 Portarchitektur eines TSN-Switches

Ein Netzwerkswitch, der TSN unterstützt, modifiziert die Architektur der TSN-fähigen Ports. In regulären Switches ohne TSN-Unterstützung werden die Pakete eines Ports an einem Netzwerkswitch in einer *Queue* abgelegt, aus der das letzte Element zugestellt wird, sobald der Port nicht länger durch das vorausgehende Paket blockiert ist. Diese *Queue* kann als einfache First In First Out (FIFO)-Queue, oder auch als komplexere gewichtete Queue oder Prioritäts-Queue umgesetzt werden.

Ein Switch mit TSN-Unterstützung nutzt an den entsprechenden Ports nicht nur eine, sondern acht *Queues* für die Pufferung eintreffender Pakete. Eingehende Pakete werden abhängig von ihrer Prioritäts-Klassifikation in diese *Queues* einsortiert. Die Priorität ist definiert als der Wert ihres Priority Code Point (PCP)-Feldes. Das PCP-Feld ist ein drei-Bit-Feld des in IEEE 802.1Q definierten VLAN-Tags. Das Feld kann Werte von null bis sieben annehmen, entsprechend gibt es eine *Queue* pro möglichem Wert.

Wenn ein Paket vollständig übertragen wurde, wird im Anschluss das nächste Paket zugestellt. Da es in einem TSN-fähigen Port statt einer einzelnen Queue nun acht Queues gibt, aus denen potentiell das nächste Paket zugestellt werden kann, muss eine Auswahl des nächsten zuzustellenden Pakets erfolgen. Diese Aufgabe übernimmt die *Transmission Selection*. Diese kann auf Basis von Prioritäten der Klassen, Quality-Of-Service-Anforderungen und anderen Merkmalen bestimmen, welcher Queue der nächste zu übertragende Frame entnommen wird. Die Auswahlfunktion der *Transmission Selection* wird von verschiedenen TSN-Standards wie IEEE 802.1Qbv genutzt, um Echtzeitkommunikation herzustellen.

2.3.2 Time-Aware Shaper (TAS)

Der in IEEE 802.1Qbv [IEEE16b] standardisierte Time-Aware Shaper (TAS) garantiert den verschiedenen Prioritätsklassen des Netzwerkverkehrs vordefinierte Zeiträume zur Kommunikation über das Netzwerk. So werden wiederkehrende Zeitzyklen in Zeitscheiben eingeteilt, die nur bestimmten Prioritätsklassen die Übertragung erlauben.

Um dies zu erreichen, ist jede der acht *Queues* mit einem *Transmission Gate* verknüpft. Das *Transmission Gate* kann einen von zwei Zuständen annehmen: *offen* oder *geschlossen*. Bei der Auswahl des nächsten zu übertragenden Pakets durch die *Transmission Selection* werden nur *Queues* in Betracht gezogen, deren *Transmission Gate* im Zustand *offen* ist. Zur Bestimmung, welche *Gates* zu welchem Zeitpunkt offen bzw. geschlossen sind, wird eine GCL genutzt. Die Einträge der GCL sind Tupel, die aus acht Binärwerten, die den Zustand der *Gates* angeben, sowie einer Zeitdauer, die angibt, wie lange der Eintrag aktiv ist, bestehen. Die *Gates* nehmen demnach für die Dauer des aktuell aktiven GCL-Eintrags die im Eintrag spezifizierten Zustände an. Ist das Zeitfenster des Eintrags abgelaufen, wird der nächste Eintrag aktiviert. Nach einer vordefinierten Zykluszeit seit Beginn des letzten Zyklus wird der erste GCL-Eintrag erneut aktiviert, und es beginnt ein neuer Zyklus. Damit werden die Zeitzyklen in Zeitscheiben eingeteilt, die nur bestimmten Prioritätsklassen die Übertragung erlauben. Die Architektur eines solchen Netzwerkswitch-Ports ist in Abbildung 2.3 dargestellt.

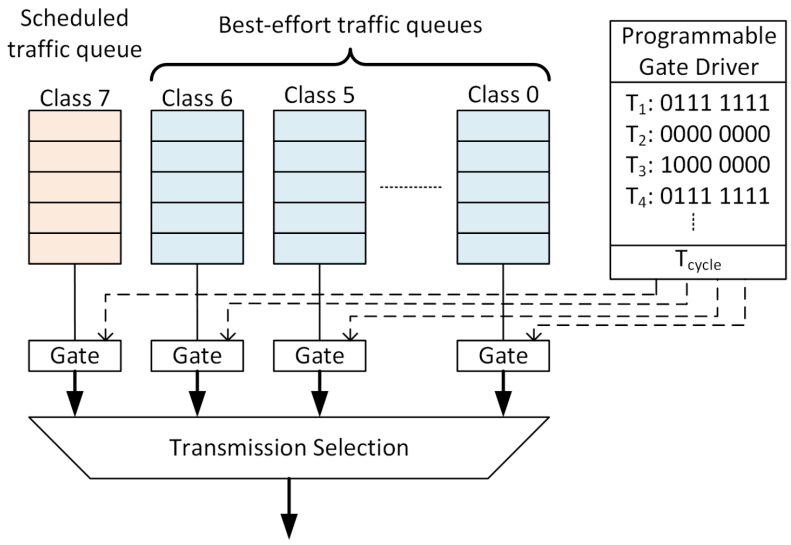

Abbildung 2.3: Architektur eines TAS-Ports [DN16]

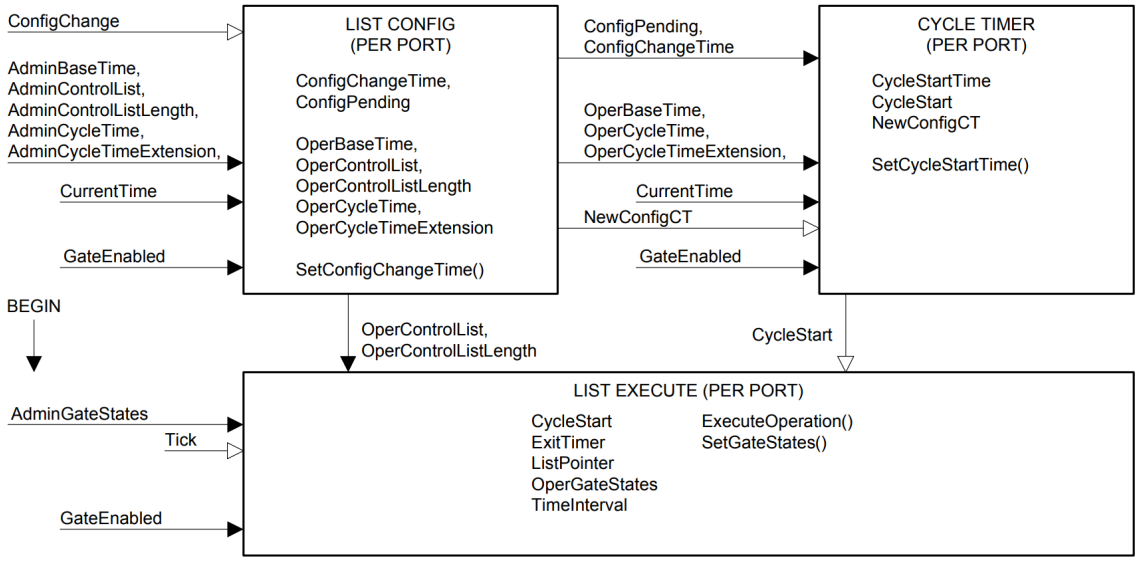

Abbildung 2.4: Zustandsmaschinen des TAS [IEEE16b]

2.3.3 TAS-Zustandsmaschinen

Verwaltet und ausgeführt wird der Wechsel der GCL-Einträge durch mehrere Zustandsmaschinen. Da der Fokus dieser Arbeit auf der Präsentation und Konfiguration der in den Zustandsmaschinen verwendeten Parameter liegt, werden diese im Folgenden genauer betrachtet. Dargestellt sind die Zustandsmaschinen und ihr Zusammenhang in Abbildung 2.4.

Um die Zustandsmaschinen zu verstehen, müssen zunächst die wichtigsten Parameter beleuchtet werden. Einige dieser Parameter gibt es sowohl als *Admin*- als auch als *Oper*-Parameter. Die *Admin*-Parameter beschreiben nicht die aktuell aktive Konfiguration des TAS, sondern werden

erst beim nächsten Konfigurationswechsel übernommen. Die *Oper*-Parameter hingegen definieren die aktuelle Konfiguration des TAS. Zu diesen Parametern zählt die *BaseTime*, die angibt, zu welchem Zeitpunkt die Konfiguration gestartet werden soll bzw. wurde. Auch dazu zählen die *ControlList* sowie *ControlListLength*-Parameter, die die GCL bzw. deren Länge beschreiben. Außerdem gibt es auch die *CycleTime* sowie die *CycleTimeExtension* als *Admin*- wie auch *Oper*-Parameter. Die *CycleTime* gibt die Zykluszeit bis zum Neustart der GCL beim ersten Eintrag an. Die *CycleTimeExtension* beschreibt eine Pufferzeit, um welche der zuletzt aktive Zyklus bei einem Wechsel der TAS-Konfiguration verlängert werden kann, um nicht mitten in einem Zyklus den Konfigurationswechsel zu vollziehen.

Die nachfolgenden Parameter gibt es nicht als *Admin*- bzw. *Oper*-Parameter, sondern liegen nur in einer Version vor. Der *ConfigChange*-Parameter gibt an, dass die *Admin*-Parameter als neue Konfiguration übernommen werden sollen, wenn er auf *true* gesetzt wird. Nach vollzogenem Konfigurationswechsel wird *ConfigChange* auf *false* gesetzt. Der *GateEnabled*-Parameter bestimmt, ob der TAS auf dem entsprechenden Port aktiv ist. Dazu können die Werte *true* bzw. *false* gesetzt werden. In IEEE 802.1Qbv sind für diese und weitere Parameter auch MIB-Einträge zur Verwaltung des TAS mittels SNMP standardisiert.

Genutzt werden diese Parameter von drei Zustandsmaschinen: Die *List Config State Machine (SM)*, die *Cycle Timer SM* und die *List Execute SM*. Die *List Config SM* nimmt die beschriebenen Parameter entgegen, wandelt sie in *Oper*-Parameter um und berechnet die Zeit des Konfigurationswechsels. Zum Zeitpunkt des Konfigurationswechsels werden die Zeitparameter an die *Cycle Timer SM* und die GCL-Parameter an die *List Execute SM* übergeben. Die *Cycle Timer SM* bestimmt auf Basis der erhaltenen Zeitparameter, wann ein neuer GCL-Zyklus startet, und benachrichtigt zu diesen Zeitpunkten die *List Execute SM* darüber. Diese führt auf Basis der erhaltenen GCL-Parameter die Zustandswechsel der *Transmission Gates* durch und startet einen neuen Zyklus, wenn sie von der *Cycle Timer SM* darüber benachrichtigt wird.

2.3.4 Zentralisiertes Konfiguratonsmodell

Der TAS wird lokal auf einzelnen Netzwerkswitches implementiert. Um mit dem TAS in komplexen Netzwerken mit mehreren TSN-Switches deterministische Echtzeit-Kommunikation gewährleisten zu können, ist es notwendig, die TSN-Konfigurationen der einzelnen Netzwerkswitches aufeinander abzustimmen. Ein zentralisierter Ansatz, um dies zu erreichen, ist der Einsatz eines CNC. Ein solcher CNC besitzt Informationen über die gesamte Netzwerktopologie und kann die TSN-Konfiguration der einzelnen Netzwerkswitches über Netzwerkkonfigurationsprotokolle wie SNMP oder NETCONF abfragen und manipulieren. Damit ist der CNC in der Lage, Konfigurationen der einzelnen Netzwerkswitches zu bestimmen, die gemeinsam die erforderliche Echtzeitkommunikation im gesamten Netzwerk ermöglichen.

Damit eine solche Konfiguration des Netzwerks die gestellten Echtzeitanforderungen tatsächlich erfüllen kann, ist es notwendig, dass die Start- und Endzeitpunkte der Zeitzyklen und -scheiben mehrerer Switches genau aufeinander abgestimmt werden können. Um dies zu erreichen, muss die Systemzeit der Switches im Netzwerk synchronisiert werden. Hierzu kann unter anderem das in IEEE 1588 [IEEE20] definierte Precision Time Protocol (PTP) genutzt werden.

Abbildung 2.5: Aufbau eines LLDP-Pakets für IEEE 802.3 Ethernet-Netzwerke [IEEE16a]

2.3.5 Credit-Based Shaper (CBS)

Neben dem TAS, auf dem der Fokus dieser Arbeit liegt, gibt es noch weitere Standards, die sich mit anderen Aspekten von TSN und anderen Ansätzen zur Realisierung von deterministischer Echtzeitkommunikation beschäftigen. Der in IEEE 802.1Qav (Forwarding and Queuing Enhancements for Time-Sensitive Streams) [IEEE10] vorgestellte Credit-Based Shaper (CBS) stellt eine creditbasierte Alternative zu dem zeitscheibenbasierten TAS aus IEEE 802.1Qbv dar.

Der CBS weist die Prioritätsklassen des PCP-Feldes einer von mehreren Stream Reservation (SR)-Klassen zu. Jede dieser SR-Klassen besitzt einen Credit-Wert. Wenn die *Transmission Selection* das nächste Paket zur Übertragung auswählt, werden nur Pakete aus den Klassen zur Übertragung akzeptiert, deren Credit-Wert positiv ist. Wird ein Paket einer SR-Klasse übertragen, so werden die Credits dieser Klasse reduziert. Wartet eine Klasse auf die Übertragung von Paketen anderer Klassen, werden ihre Credits erhöht. Die Anzahl Credits, die einer Klasse abgezogen bzw. hinzugefügt werden, sind konfigurierbar. Damit kann bestimmt werden, wie oft Pakete einer Klasse übertragen werden können. Im Ergebnis kann so festgelegt werden, welcher Anteil der insgesamt verfügbaren Bandbreite für die einzelnen SR-Klassen reserviert wird. Um deterministische Echtzeit-Kommunikation in komplexen Netzwerken zu ermöglichen, wird auch beim CBS eine gemeinsame Konfiguration der einzelnen Netzwerkswitches auf Basis der gesamten Netzwerktopologie benötigt.

2.4 Link Layer Discovery Protocol (LLDP)

Das Link-Layer Discovery Protocol (LLDP) ist ein in IEEE 802.1AB [IEEE16a] standardisiertes Protokoll, das es benachbarten Geräten in einem IEEE 802 Local Area Network (LAN) ermöglicht, Informationen über sich und die Fähigkeiten des Geräts auszutauschen. Damit arbeitet LLDP auf Schicht zwei des OSI-Modells. Es werden in regelmäßigen Abständen LLDP-Pakete zwischen benachbarten Geräten ausgetauscht, die einem standardisierten Format mit pflicht- und optionalen Feldern folgen. In IEEE 802.3 (Ethernet)-Netzwerken [IEEE22a] bestehen diese Pakete aus drei Abschnitten: Dem *Header*, der *Link Layer Discovery Protocol Data Unit (LLDPDU)* und einer *Frame Check Sequence (FCS)*. Dargestellt ist der Aufbau eines solchen LLDP-Pakets in Abbildung 2.5.

Der *Header* dient der Zustellung des LLDP-Pakets über ein IEEE 802 LAN. Die Informationen, die mittels LLDP ausgetauscht werden, befinden sich in der LLDPDU. Die Felder der LLDPDU werden als Type-Length Value (TLV) dargestellt. Sie beginnt mit 3 zwingend vorhandenen Feldern: Der *Chassis ID*, der *Port ID* und der *Time To Live*. Die *Chassis ID* ist ein einzigartiger Identifikator für das Netzwerkgerät. Die *Port ID* hingegen ist ein einzigartiger Identifikator für den Port des Geräts, von dem aus das LLDP-Paket gesendet wurde. Dies ist oft die Media Access Control (MAC)-Adresse. Die *Time To Live* gibt an, wie lange das empfangende Gerät die in diesem Paket gesendeten Informationen als gültig betrachten soll. Anschließend können weitere, optionale TLVs

eingebracht werden, die zum Beispiel den Systemnamen und die Fähigkeiten des sendenden Geräts angeben. Das Ende der LLDPDU wird durch eine spezielle *End of LLDPDU*-TLV angegeben. Im Anschluss an die LLDPDU folgt noch eine FCS, mit deren Hilfe die Integrität des LLDP-Pakets überprüft werden kann.

Der IEEE 802.1AB-Standard definiert außerdem MIB-Einträge, in denen die mittels LLDP erhaltenen Informationen über Nachbargeräte gespeichert und über SNMP abgefragt werden können. Damit ist es möglich, durch Abfragen der LLDP-MIBs der Geräte eines Netzwerkes die Topologie des Netzwerks herauszufinden.

2.5 Hypertext Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll zum Austausch von (Hyper-)Text-Nachrichten zwischen Geräten in einem Netzwerk. HTTP wird unter anderem eingesetzt, um die Kommunikation zwischen Webbrowsern und Webservern zu realisieren. Damit ist HTTP ein Kernelement jeder Web-Anwendung. Die neueste Version, Hypertext Transfer Protocol version 3 (HTTP/3), wurde im Juni 2022 in RFC 9114 [Bis22] standardisiert. Da HTTP/3 noch keine 2 Jahre alt ist, und die für den Datenaustausch zwischen Browser und Server verwendete HTTP-Version von Browser und Server unterstützt werden muss, sind ältere Versionen von HTTP noch weit verbreitet. Im Mai 2015 wurde Hypertext Transfer Protocol version 2 (HTTP/2) in RFC 7540 [BPT15] standardisiert. Während HTTP/2 hauptsächlich semantische Änderungen gegenüber Hypertext Transfer Protocol version 1.1 (HTTP/1.1) einführt, ist HTTP/3 eine größere Überarbeitung des Protokolls. Frühere Versionen bauten zur Kommunikation zwischen Client und Server zunächst eine TCP-Verbindung auf und tauschten über diese Verbindung HTTP-Nachrichten aus, HTTP/3 nutzt hierfür jedoch QUIC auf Basis des UDP. Gemäß Daten von Cloudflare wurden im April 2023 rund 64% aller HTTP-Anfragen mit HTTP/2 durchgeführt [BP23]. Da auch das in dieser Arbeit vorgestellte Backend des CNC-Prototypen bis zu HTTP/2 unterstützt, basiert der HTTP-Abschnitt dieses Kapitels auf RFC 7540 [BPT15].

2.5.1 HTTP-Nachrichten

Grundsätzlich folgt HTTP dem Client/Server-Modell: Ein Client stellt eine Anfrage bei einem Server, und bekommt daraufhin eine Antwort zurückgeschickt. HTTP-Nachrichten bestehen aus 3 Abschnitten: Der *Request*- bzw. *Status-Zeile*, dem *Header* und dem *Body*.

Bei einer HTTP-Anfrage wird in der *Request-Zeile* angegeben, welcher Endpunkt auf dem Server mit welcher HTTP-Operation angesprochen werden soll. Außerdem wird die verwendete HTTP-Version angegeben. Die *Header*-Sektion enthält vordefinierte Variablen und deren Werte. Hier gibt es einige standardisierte HTTP-Header, eine Web-Anwendung kann jedoch auch eigene Header definieren und nutzen, solange sie von Backend und Frontend verstanden werden. So nutzt der in dieser Arbeit vorgestellte Prototyp beispielsweise einen *'webcnc-password'*-Header, um zu verifizieren, dass der Frontend-Client auf bestimmte Operationen im Backend zugreifen darf. Die *Body*-Sektion enthält die eigentlichen zu transferierenden Daten. Dabei kann es sich um ganze Websites mit HyperText Markup Language (HTML), Cascading Style Sheet (CSS) und JavaScript-Code handeln, es können aber auch andere Nutzdaten wie eine Objektstruktur im JavaScript Object Notation (JSON)-Format oder andere Datentypen versendet werden.

Die Struktur von HTTP-Antworten entspricht weitgehend der von Anfragen, nur die *Request-Zeile* wird durch eine *Status-Zeile* ersetzt. Die *Status-Zeile* enthält ebenfalls die verwendete HTTP-Version, statt Endpunkt und Operation beinhaltet sie jedoch einen Statuscode, der angibt, ob die Operation erfolgreich durchgeführt werden konnte. Durch verschiedene Statuscodes kann im Fehlerfall die Art des aufgetretenen Fehlers angegeben werden.

2.5.2 HTTP-Endpunkte

Ein Webserver stellt für die HTTP-Kommunikation Endpunkte bereit, die über einen hierarchisch strukturierten Pfad angesprochen werden. So besitzt der in dieser Arbeit vorgestellte Prototyp unter anderem die Endpunkte *'/webcnc/api/getswitches'* sowie *'/webcnc/api/saveport'*. Ein Endpunkt kann durch die optionale Unterstützung mehrerer – mindestens jedoch einer – HTTP-Operationen verschiedene Funktionalitäten anbieten. HTTP/2 übernimmt die in RFC 7231 [FR14] für HTTP/1.1 definierten Operationen. Die wichtigsten dieser Operationen sind *GET*, *HEAD*, *POST*, *PUT* und *DELETE*.

Die *GET*-Operation wird benutzt, um eine Ressource vom Webserver abzufragen. So kann beispielsweise ein statische Webseite oder dynamisch generierte Daten vom Server abgefragt werden. Dafür enthält die Anfrage nur *Header* und keinen *Body*. Die *HEAD*-Operation ähnelt der *GET*-Operation, erwartet jedoch nur *Header* als Antwort und keinen *Body*. Eine *POST*-Anfrage beinhaltet neben den *Headern* auch einen *Body* mit Daten. Diese Daten sollen vom empfangenden Webserver verarbeitet werden und an eine bestehende Ressource angefügt oder als neue Ressource angelegt werden. Hierbei sollen keine Daten überschrieben werden. Die *PUT*-Operation ähnelt der *POST*-Operation, hier können jedoch bereits existierende Daten auf dem Webserver überschrieben werden. Die *DELETE*-Operation zielt darauf ab, eine Ressource vom Webserver zu entfernen, und besitzt dafür nur *Header* und keinen *Body*.

2.5.3 HTTP-Sicherheit

HTTP-Nachrichten sind menschenlesbare Klartextdateien. Damit können HTTP-Nachrichten von anderen Teilnehmern des Netzwerks gelesen und manipuliert werden. Um die HTTP-Kommunikation abzusichern, wird Hypertext Transfer Protocol Secure (HTTPS) eingesetzt. HTTPS beschreibt die Nutzung von HTTP zusammen mit dem Transport Layer Security (TLS)-Protokoll. HTTP/2 erfordert gemäß RFC 7540 [BPT15] mindestens TLS version 1.2 [RD08]. TLS ermöglicht eine verschlüsselte, authentifizierte Kommunikation mit Integritätsprüfung, um zu verhindern, dass Nachrichten abgehört, gefälscht oder verändert werden. HTTP-Kommunikation ohne die Nutzung von TLS ist als unsicher anzusehen und entspricht nicht den modernen Anforderungen an Web-Anwendungen. Daher wird HTTPS auch vom in dieser Arbeit vorgestellten Prototypen unterstützt.

Um TLS einzusetzen, benötigt der Server, der die HTTP-Endpunkte bereitstellt, ein TLS-Zertifikat. Dieses Zertifikat wird von einer Certificate Authority (CA) ausgestellt, und dient als Nachweis, dass der Kommunikationspartner des Clients tatsächlich der Server ist, der er vorgibt zu sein. Erreicht wird dies über ein asymmetrisches Kryptografie-Verfahren, bei dem der Client durch das Zertifikat einen öffentlichen Schlüssel erhält, mit dem Nachrichten, die an den Server gesendet werden, verschlüsselt werden. Mithilfe eines privaten Schlüssels, der nicht effizient aus dem öffentlichen Schlüssel berechnet werden kann, und der nur dem Server bekannt ist, können

serverseitig diese Nachrichten wieder entschlüsselt werden. Damit ein Angreifer dem Client nicht mit einem gefälschten Zertifikat vorgeben kann, der echte Server zu sein, muss der Client das Zertifikat bei der ausstellenden CA oder einer anderen, der ausstellenden CA übergeordneten CA, validieren. Vertraut der Client der CA, wird das Zertifikat als gültig angesehen und es kann mithilfe des asymmetrischen Verschlüsselungsverfahren eine sichere Kommunikation aufgebaut werden. Häufig wird mithilfe des asymmetrischen Verschlüsselungsverfahrens ein symmetrischer Schlüssel ausgehandelt, um Ver- und Entschlüsselung großer Datenmengen auf beiden Seiten effizienter zu gestalten.

3 Verwandte Arbeiten

In diesem Kapitel werden andere Publikationen und CNCs vorgestellt, die mit dieser Arbeit verwandt sind. So werden im ersten Abschnitt einige bereits existierende TSN-CNCs betrachtet. Anschließend werden SDN, das OpenFlow-Protokoll und freie CNCs dafür präsentiert.

3.1 TSN-Network-Controller

Es gibt bereits eine Reihe an verfügbaren TSN-CNCs. Diese CNCs und ihre Unterschiede zum WebCNC werden in diesem Abschnitt diskutiert.

3.1.1 CNC-Prototyp von Álvarez et al.

In der Publikation "Implementing a First CNC for Scheduling and Configuring TSN Networks" [ÁSP+22] stellen Álvarez et al. einen Prototypen eines TSN-CNC, der die TAS von Switches konfigurieren kann, vor. Im Gegensatz zu dieser Arbeit, bei der die TAS-Konfiguration manuell eingegeben wird, nimmt die Arbeit von Álvarez et al. Netzwerkverkehrs-Anforderungen von einem CUC oder anderen Quellen entgegen und berechnet daraus die zum Erreichen dieser Anforderungen notwendigen TAS-Konfigurationspläne. Die Architektur des CNC von 'Alvarez et al. ist in Abbildung 3.1 dargestellt.

Die Berechnung der Konfigurationspläne wird nicht von dem CNC selbst implementiert, sondern von dem bereits existierenden *HERMES*-Scheduler [BAP+22] übernommen. Der von *HERMES* generierte Konfigurationsplan kann nicht direkt als Parameter auf den Netzwerkswitches eingesetzt werden, sondern muss zunächst in gemäß IEEE 802.1Qbv [IEEE16b] kompatible TAS-Parameter umgewandelt werden. Dafür wird eine *GCL converter*-Komponente eingesetzt, welche die produzierte GCL im JSON-Format ausgibt. Diese GCL wird an einen *Configuration Agent* weitergeleitet.

Der *Configuration Agent* beinhaltet die restlichen Komponenten der CNC und läuft zur Verbesserung der Portabilität in einem Docker-Container. Eintreffende GCL-JSONs werden hier hier von einem *Schedule Parser* angenommen. Der *Schedule Parser* ließt die GCL-JSON ein, interpretiert sie und speichert die darin enthaltenen Informationen in Programm-internen Datenstrukturen ab. Diese werden dann an den *YANG instance generator* geliefert, der auf Basis einen YANG-Modells aus den GCL-Informationen für jeden Switch eine XML-Datei zur Konfiguration der Switches mittels NETCONF generiert. Ist die XML-Datei zur Konfiguration eines Switches generiert, wird sie an einen *NETCONF Client* geleitet, der die NETCONF-Kommunikation mit den Switches implementiert und damit die gewünschte TAS-Konfiguration auf den Switches setzt.

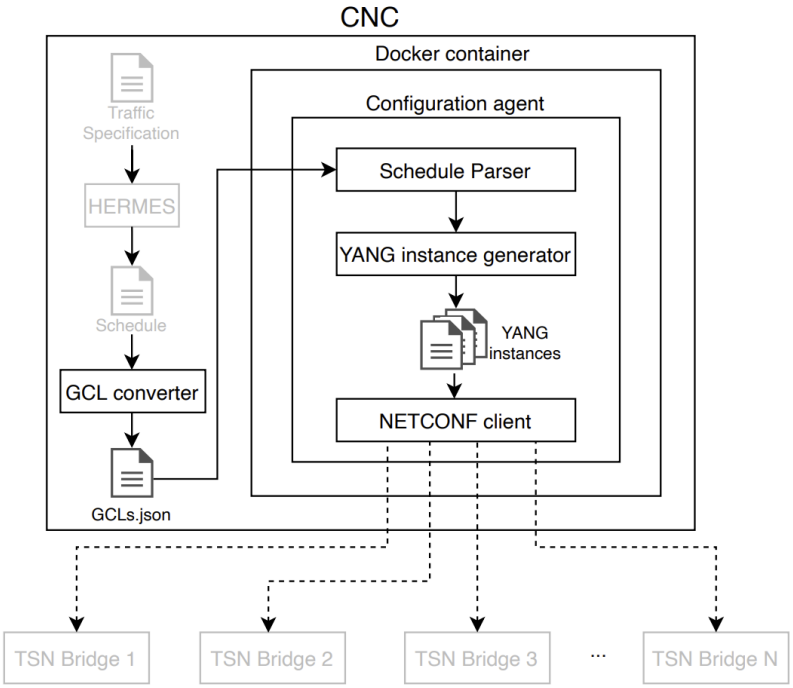

Abbildung 3.1: CNC-Architektur [ÁSP+22]

Grundsätzlich unterscheidet sich die von Álvarez et al. vorgestellte CNC-Architektur von dieser Arbeit in zwei Kernpunkten. Zum einen ist die Wahl der Eingabedaten verschieden: Während der CNC von Álvarez et al. Netzwerkverkehrsanforderungen von einem anderen System annimmt, und diese mittels des *HERMES*-Schedulers in TAS-Konfigurationen umwandelt, stellt der in dieser Arbeit vorgestellte CNC ein Webinterface zur manuellen Konfiguration der TAS bereit. Außerdem werden durch die unterschiedliche Wahl des Netzwerkkonfigurationsprotokolls – NETCONF statt SNMP – unterschiedliche Netzwerkswitches unterstützt, da viele gängige Netzwerkswitches nicht beide Konfigurationsprotokolle unterstützen. Insbesondere ältere Switch-Modelle unterstützen NETCONF häufig nicht, da NETCONF deutlich später als SNMP standardisiert wurde.

3.1.2 OpenCNC

Ein frei verfügbarer TSN-CNC ist der OpenCNC von Hamza Chahed et al. [CJ]. Dieser CNC ist in Go, einer von Google entwickelten Programmiersprache, implementiert, und basiert auf einer Microservice-Architektur. Bei einer Microservice-Architektur werden mehrere voneinander getrennte Services eingesetzt, die genau eine Aufgabe in der Gesamtanwendung implementieren und mit anderen Microservices der Anwendung über den Austausch von Nachrichten kommunizieren. Der OpenCNC setzt für die Kommunikation zwischen Microservices das gRPC Network Management Interface (gNMI) [SSBM22] ein, ein von Google entwickeltes RPC-Interface, das auf dem ebenfalls von Google entwickelten gRPC [Goob] RPC-Protokoll basiert. gNMI kann zur Konfiguration und Verwaltung von Systemen in einem Netzwerk genutzt werden, und ist damit eine Alternative zu

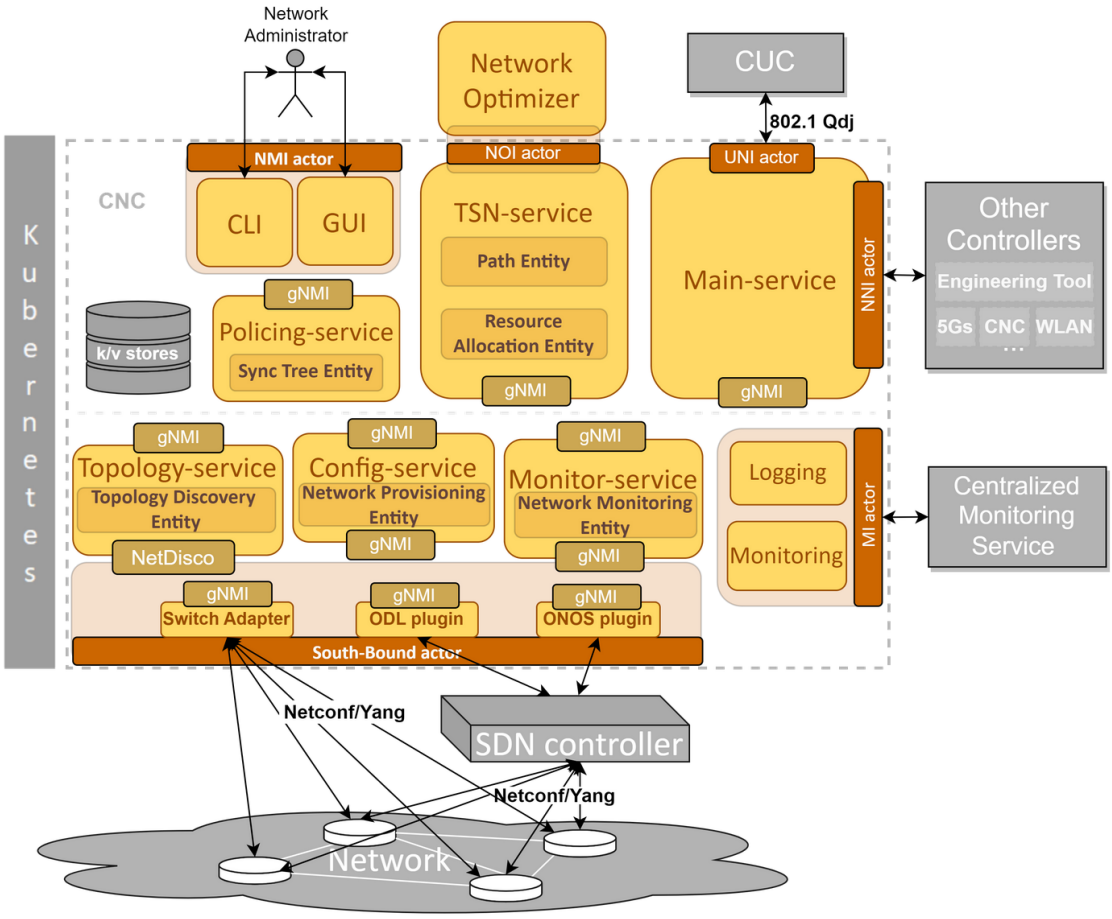

Abbildung 3.2: Microservice-Architektur der OpenCNC [CJ]

anderen Netzwerkmanagementprotokollen wie NETCONF oder SNMP. Für die Kommunikation mit den Netzwerkswitches setzt der OpenCNC ebenso wie der CNC von Álvarez et al. auf NETCONF.

Ein Überblick über die Architektur des OpenCNC ist in Abbildung 3.2 dargestellt. Die als Microservices implementierten Funktionen des OpenCNC umfassen den *Main-service*, der Kommunikation mit einem CUC oder anderen Controllern ermöglicht, um das Netzwerk dynamisch auf Basis von Netzwerkverkehrsanforderungen zu konfigurieren. Außerdem stellt der OpenCNC ein Command Line Interface (CLI) und ein Graphical User Interface (GUI) bereit, die ein manuelles Einsehen und Konfigurieren des Netzwerks erlauben. Um aus erhaltenen Datenflussanforderungen die Konfigurationspläne und TSN-Konfigurationen der einzelnen Switches zu berechnen, gibt es den *TSN-service*. Die Verwaltung der von den TSN-Switches verwendeten YANG-Modelle übernimmt der *Config-service*-Microservice. Dieser ist auch verantwortlich für die Koordination des *Switch Adapter*, der die NETCONF-Kommunikation mit den Switches implementiert. Der OpenCNC stellt außerdem mit dem *Topology-Service* einen Microservice zum Erkennen der Netzwerktopologie bereit, und der *Monitor-Service* ist zuständig dafür, das Netzwerk auf Änderungen außerhalb des OpenCNC zu überwachen, um einen konsistenten Zustand der internen Netzwerkrepräsentation mit dem realen Netzwerk zu gewährleisten. Außerdem stellt der OpenCNC Logging-Funktionen, ein

Interface zu einem Netzwerk-Optimierer sowie Interfaces zur Kommunikation mit SDN-Controllern bereit. Zum Deployment des OpenCNC wird ein Kubernetes-Cluster mit Docker-Containern für die Microservices genutzt.

Das von dem OpenCNC bereitgestellte Funktionsspektrum ist damit sehr umfassend. Die Umsetzung des CNC in einem Kubernetes-Cluster mittels Docker-Containern und weiteren Abhängigkeiten sorgt jedoch auch für erhöhte Infrastrukturanforderungen. Die Abwägung von Benutzerfreundlichkeit vs. Mächtigkeit ist damit ein Kernunterschied zwischen dem OpenCNC und dieser Arbeit. Weitere Kernunterschiede sind die Wahl des Netzwerkkonfigurationsprotokolls – der OpenCNC setzt hier auf NETCONF – sowie die Implementierung des OpenCNC als Microservice-Architektur.

3.1.3 SlateXNS

Auch im Bereich kommerzieller Closed-Source-Produkte sind TSN-CNCs vertreten. Das von TTTech angebotene SlateXNS [TTT] stellt ein solches Produkt dar.

SlateXNS ist ein GUI-basierter CNC, ähnlich des in dieser Arbeit vorgestellten WebCNC. Die Art der vom Nutzer eingegebenen Daten ist jedoch verschieden. So wird bei SlateXNS nicht die TSN-Konfiguration der Switches vom Nutzer eingegeben. Stattdessen gibt der Nutzer Datenflussanforderungen ein, die über das Netzwerk zwischen Endgeräten benötigt werden. Daraus berechnet SlateXNS selbstständig ein Scheduling mit den entsprechenden TSN-Konfigurationen der Switches, und setzt diese neuen Konfigurationen anschließend auf den Switches.

Weitere Unterschiede lassen sich in der Wahl des Netzwerkkonfigurationsprotokolls finden. Für die Kommunikation mit den TSN-Switches setzt SlateXNS im Gegensatz zum WebCNC-Prototypen auf NETCONF. Außerdem kann in SlateXNS die Modellierung der gesamten Netzwerktopologie manuell vom Nutzer über die GUI vorgenommen werden. Ob SlateXNS über eine Option zur (teil-)automatischen Erkennung der Netzwerktopologie ähnlich dem WebCNC verfügt, ist auf Basis der veröffentlichen Materialien nicht klar erkennbar.

3.2 SDN-Network-Controller

Neben TSN gibt es noch weitere Technologien zur Netzwerksteuerung, die zentral über einen CNC gesteuert werden können. SDN bezeichnet eine weiterer Netzwerkmanagement-Technologie, bei der die Kontrollebene von der Datenebene getrennt wird. Während sich TSN vor allem auf das Scheduling von Netzwerkverkehr fokussiert, liegt bei SDN der Fokus auf der Steuerung der Paketweiterleitung. So können zum Beispiel Routing-Entscheidungen von einer zentralen Stelle aus beeinflusst werden, um eine effizientere Nutzung der Netzwerkressourcen zu erreichen.

Da einzelne Switches und Router nur eine begrenzte Kenntnis über die gesamte Netzwerkstruktur besitzen, müssen ohne zentrale Steuerung Routingentscheidungen mit unvollständigem Wissen getroffen werden, die zu einem suboptimalen Datenfluss führen. Um dieses Problem zu vermeiden, ist die Steuerung eines SDN-Netzwerks zentralisiert – erreicht wird dies in der Regel durch den Einsatz eines SDN-CNC.

Ein Protokoll zur Umsetzung eines SDN-CNC ist das von der Open Networking Foundation standardisierte OpenFlow-Protokoll [Ope15]. Im Gegensatz zu anderen Protokollen wie SNMP oder NETCONF, mit denen beliebige Konfigurationen vorgenommen werden können, liegt der Fokus von OpenFlow speziell auf der Konfiguration von Parametern, welche die Paketweiterleitung in Netzwerken regulieren. Dieser Abschnitt basiert auf Version 1.5.1 des OpenFlow-Standards, der neuesten frei verfügbaren Version.

Switches und Router, die OpenFlow implementieren, können von einem OpenFlow-CNC gesteuert werden. Hierzu setzt OpenFlow *Flow Tables* ein, die *Flow Entries* enthalten. Diese bestimmen, wie eintreffende Pakete zu behandeln sind. So kann regelbasiert bestimmt werden, wie und ob Pakete weitergeleitet oder modifiziert werden. Um dies umzusetzen, beinhaltet ein *Flow Entry* verschiedene Felder, darunter die *match fields*, *priority* und *instructions*-Felder. Die *match fields* werden genutzt, um zu bestimmen, ob der entsprechende *flow entry* auf ein eintreffendes Paket anzuwenden ist. Das *priority*-Feld wird genutzt, um zu bestimmen, in welcher Reihenfolge die auf ein Paket anzuwendenden *Flow Entries* einzusetzen sind. Das *instructions*-Feld beinhaltet die Handlungen, die gemäß des *flow entry* auf das Paket anzuwenden sind. Hiermit wird unter anderem definiert, ob und auf welchem Port das Paket weitergeleitet wird oder wie es zu modifizieren ist. Neben diesen Feldern besitzt ein *Flow Entry* noch weitere Felder zur Nutzung durch den Controller, wie zum Beispiel ein *counter*-Feld, das angibt, auf wie viele Pakete der *Flow Entry* bereits angewendet wurde.

Für OpenFlow existieren verschiedene Open-Source-Projekte, die OpenFlow-CNCs bereitstellen. Darunter fallen unter anderen der Java-basierte Floodlight-Controller [MQJ+], der ebenfalls Java-basierte OpenDaylight-Controller [LF] sowie die Python-basierten Ryu- [Ryu] und Pox-Controller [MSz+]. Mit diesen und weiteren, weniger bekannten OpenFlow-Controllern, gibt es im Vergleich zu TSN eine Vielzahl an CNCs für SDN, die das OpenFlow-Protokoll implementieren.

4 Systemmodell und Problemstellung

Dieses Kapitel legt die Grundlagen für die weitere Erläuterung der in dieser Arbeit erzielten Ergebnisse. Zunächst wird das zum Entwurf des CNC genutzte Systemmodell anhand eines Modells präsentiert. Anschließend werden die Anforderungen, die der in dieser Arbeit entworfene CNC erfüllen soll, präsentiert.

4.1 Systemmodell

Das zum Entwurf des WebCNC verwendete Systemmodell ist in Abbildung 4.1 dargestellt. In diesem Systemmodell können beliebig viele Netzwerkswitches und Endgeräte vorhanden sein. Die Switches sind TSN-fähig, insbesondere unterstützen sie den in IEEE 802.1Qbv [IEEE16b] standardisierten TAS. Außerdem besitzen die Switches einen SNMP Agent zur Verwaltung der Switches und stellen MIBs bereit, aus denen von den Switches gesammelte LLDP-Daten abgefragt werden können und die TAS-Parameter ausgelesen und angepasst werden können. Die Switches können über Ethernet-Verbindungen mit anderen Switches oder Endgeräten verbunden sein. Auf einem NMS im Netzwerk läuft der CNC-Prototyp, der die TSN-Konfiguration der Netzwerkswitches steuert und eine GUI zur Interaktion mit dem Nutzer bereitstellt. Von dem NMS muss ein Kommunikationspfad zu allen verwalteten Switches bestehen. Sowohl die Endgeräte als auch die Switches senden LLDP-Pakete an

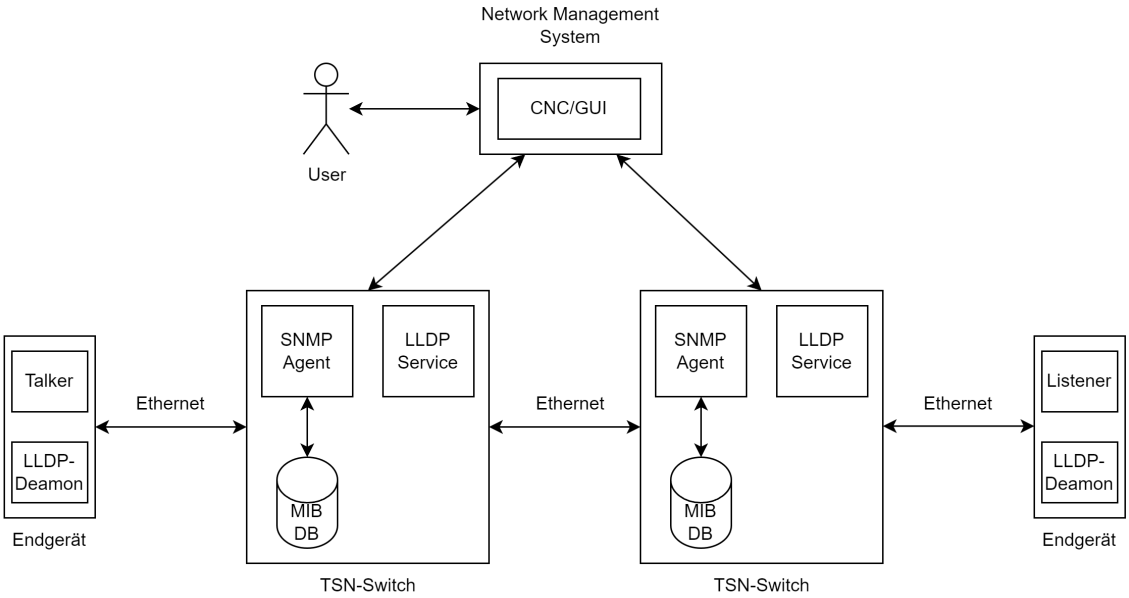

Abbildung 4.1: Systemmodell

ihre Nachbargeräte, um eine Erkennung der Netzwerktopologie zu ermöglichen. Auf den Endgeräten können Talker und Listener laufen, die Kommunikation mit Echtzeitanforderungen zwischen den Geräten durchführen.

4.2 Problemstellung

Um TSN-Netzwerke manuell zu konfigurieren, muss jeder TSN-Switch im Netzwerk separat konfiguriert werden. Ohne den Einsatz eines CNC erfolgt dies meist durch die Nutzung von Konfigurationswebinterfaces oder einer CLI der einzelnen Switches. Dies ist ein sehr aufwändiger Prozess, da für jeden Switch im Falle einer Webinterfaces das Konfigurationswebinterface geladen werden muss, eine Anmeldung erfolgen muss und dann im Webinterface zur TSN-Konfiguration navigiert werden muss. Verschiedene Switches können zudem verschiedene Weboberflächen bereitstellen, was den manuellen Konfigurationsprozess weiter erschwert. CLIs hingegen bieten darstellungsbedingt nur wenige Möglichkeiten zur Visualisierung. Außerdem ist die Nutzung einer CLI mangels grafischer Elemente weniger intuitiv als eine GUI.

Außerdem liegt in den Interfaces einzelner Switches in der Regel keine globale Sicht auf das Netzwerk – und damit keine Darstellung der gesamten Netzwerktopologie – vor, was die Identifikation der korrekten zu setzenden Konfiguration erschwert.

Das allgemeine Ziel dieser Arbeit ist es, diese beim manuellen Konfigurieren von TSN-Netzwerk bestehenden Probleme mittels eines CNCs zu lösen. Die funktionalen Anforderungen an den CNC sind daher die korrekte Darstellung der aktuellen TAS-Konfigurationen mit den in Abschnitt 2.3.3 vorgestellten Parametern der Netzwerkswitches sowie die Möglichkeit, diese Konfigurationen zu ändern. Außerdem soll die Webanwendung Informationen zur Netzwerktopologie liefern, die es ermöglichen, die TSN-Konfiguration des Netzwerks ohne a-priori-Wissen über die Netzwerkstruktur vorzunehmen. Dazu soll eine Web-Anwendung bereitgestellt werden, die über einen normalen Webbrowser mit JavaScript-Unterstützung bedient werden kann.

Zusätzlich zu den bisher genannten funktionalen Anforderungen gibt es eine Reihe nicht-funktionaler Anforderungen, die vom in dieser Arbeit entworfenen CNC erfüllt werden sollen. So soll der CNC modular aufgebaut und erweiterbar sein, damit in Zukunft die Implementierung weitere Funktionalitäten wie beispielsweise Unterstützung für NETCONF mit geringstmöglichem Aufwand möglich sind. Auch soll der CNC portabel sein und damit auf allen gängigen Systemen plattformunabhängig bereitgestellt werden können. Als drittes nichtfunktionales Ziel soll der CNC zudem benutzerfreundlich sein. Er soll mit geringem Aufwand auf einem Server installiert werden können und anschließend ohne lokale Installationen über reguläre Webbrowser bedient werden können.

Zusammenfassend soll der in dieser Arbeit zu entwerfende CNC ein portabler, benutzerfreundlicher und erweiterbarer CNC sein, der über einen Webbrowser die IEEE 802.1Qbv-Scheduler-Konfiguration von Netzwerkswitches präsentiert und eine Manipulation dieser Konfiguration ermöglicht. Zudem sollen Informationen über die Netzwerktopologie bereitgestellt werden.

5 Architektur

Im vorherigen Kapitel wurden die Anforderungen, die der CNC erfüllen soll, vorgestellt. Dieses Kapitel widmet sich dem Entwurf einer Architektur für den CNC, mit der das Erfüllen der Anforderungen ermöglicht wird.

5.1 Überblick

Wie in Abschnitt 4.2 festgelegt, soll die Interaktion mit dem WebCNC in Form einer Webanwendung realisiert werden. Um dies umzusetzen, wird eine adaptierte Drei-Schichten-Architektur auf Basis des Server/Client-Modells eingesetzt. Beim Server/Client-Modell gibt es einen Client - hier das Frontend, das im Browser des Nutzers läuft - der Anfragen an ein Backend stellt und von diesem entsprechende Antworten erhält. Eine Übersicht über die Architektur des WebCNC ist in Abbildung 5.1 dargestellt.

Die in der internen Komponentenstruktur des CNC verfolgte Adaption der Drei-Schichten-Architektur modifiziert die Zugehörigkeit der Komponenten zu den Schichten. Bei einer Drei-Schichten-Architektur teilt sich die Anwendung in drei Schichten – die Präsentationsschicht, die Anwendungsschicht und die Datenschicht – auf. Die Präsentationsschicht setzt die Kommunikation mit den Nutzern um und kommuniziert dazu mit der Anwendungsschicht. Sie befindet sich auf der Clientseite des Server/Client-Modells.

Die restlichen Schichten befinden sich hingegen auf der Serverseite. Die Anwendungsschicht setzt die Kernlogik der Anwendung um und bestimmt, wie Anfragen von der Präsentationsschicht verarbeitet werden. Dazu kommuniziert die Anwendungsschicht auch mit der Datenschicht. Die Datenschicht ist dafür verantwortlich, auf Anfrage von der Anwendungsschicht Operationen auf den gehaltenen Daten vorzunehmen und diese abzufragen und zu manipulieren. Damit greift nur die Datenschicht direkt auf die vorhandenen Daten zu.

Die hier vorgenommene Adaption der Drei-Schichten-Architektur besagt, dass Komponenten der Datenschicht nicht zwangsläufig die Daten, auf die sie Zugriff gewähren, selbst halten müssen. Damit sind Komponenten, die auf externe Datenstrukturen wie Datenbanken zugreifen und diese dem Rest des CNC über ein internes Interface bereitstellen, ebenfalls Teil der Datenschicht. Diese Adaption wird benötigt, da die Architektur den Komponenten generell nicht vorgibt, ob externe Datenbanken zu verwenden sind oder ob die Komponenten die gehaltenen Daten intern verwalten. Dies würde ohne die vorgenommene Adaption dazu führen, dass Komponenten abhängig von Implementierungsdetails verschiedenen Schichten zugeordnet werden müssten, was nicht erwünscht ist.

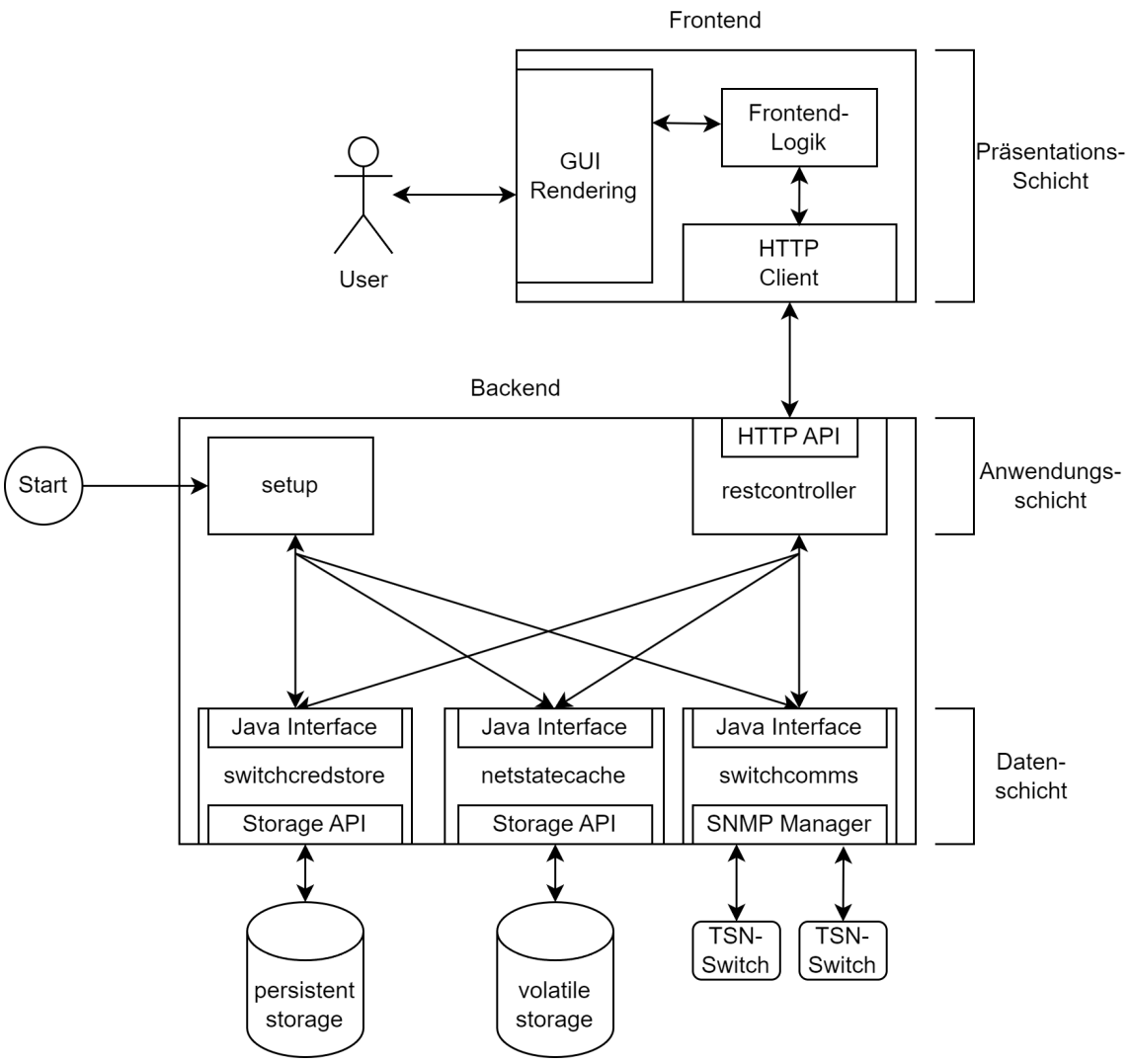

Abbildung 5.1: Architektur-Übersicht

5.2 Präsentationsschicht

Zur Nutzung der Webanwendung wird zunächst ein Frontend, das die Präsentationsschicht übernimmt, von einem Webserver in den Browser des Nutzers geladen. Das Frontend stellt dann eine GUI bereit, die es dem Nutzer ermöglicht, die aktuellen TSN-Konfigurationen der Switches einzusehen und zu manipulieren. Außerdem stellt sie Informationen zur Netzwerktopologie in der GUI bereit und ermöglicht es, neue Netzwerkswitches und ihre Zugangsdaten im CNC zu registrieren. Um Daten über den Netzwerkzustand zu erhalten und diesen zu manipulieren, kommuniziert das Frontend entsprechend der Drei-Schichten-Architektur nicht direkt mit den Netzwerkswitches, sondern nur mit der Anwendungsschicht im Backend.

5.3 Anwendungsschicht

Sämtliche Kommunikation des Frontends findet nur mit der Anwendungsschicht des Backends statt. Dafür besitzt das Backend eine *restcontroller*-Komponente, die eine HTTP- Application Programming Interface (API) bereitstellt, auf die das Frontend zugreifen kann. Über diese HTTP-API können der aktuelle Netzwerkzustand abgefragt werden, neue zu setzende TAS-Konfigurationen an das Backend gesendet werden und die Zugangsdaten von neuen Switches zur Speicherung im Backend übermittelt werden. Eintreffende HTTP-Anfragen werden von einem Webserver angenommen und von den Controllern der Komponente mittels Zugriffen auf Komponenten der Datenschicht verarbeitet. Anschließend werden entsprechende Antworten an das Frontend zurückgesendet.

Neben der *restcontroller*-Komponente besitzt die Anwendungsschicht mit der *setup*-Komponente auch eine Komponente, die sich um die auszuführende CNC-Logik beim Start des WebCNC kümmert. Die *setup*-Komponente wird daher nicht über ein Interface angesprochen, sondern wird automatisch beim Start des CNC ausgeführt. Sie steuert die nötigen Aktionen zum Herstellen eines Zustands der WebCNC-Komponenten, welcher die Bearbeitung der Anfragen von Clients über die *restcontroller*-Komponenten erlaubt. So muss der persistente Speicher über Interfaces der Datenschicht ausgelesen werden, um die gespeicherten Zugangsdaten der Switches zu erhalten. Anschließend müssen TAS-Konfigurationen und LLDP-Daten der Switches abgefragt werden und die erhaltenen Daten in einer weiteren Datenschicht-Komponente abgespeichert werden. Dies beim CNC-Start zu koordinieren ist Aufgabe der *setup*-Komponente.

Die Initialisierung der Anwendungsschicht-Komponenten und verwendeter Dienste wie einem Logger oder der für die Annahme von HTTP-Anfragen genutzte Webserver ist ausdrücklich nicht Aufgabe der *setup*-Komponente. Diese Initialisierungsschritte werden vom verwendeten Backend-Framework übernommen und sind nicht Teil einer explizit definierten Komponente. Komponenten wie die *setup*- und *restcontroller*-Komponente, die beim Start bzw. bei eingehenden HTTP-Anfragen aufgerufen werden, müssen daher beim Backend-Framework entsprechend ihrer Rolle in der Architektur registriert werden.

5.4 Datenschicht

Während auch Komponenten der Anwendungsschicht komponenteninterne Daten selbst verwalten können, zeichnen sich Komponenten der Datenschicht dadurch aus, dass ihre Hauptaufgabe die Bereitstellung von komponentenübergreifend relevanten Daten sowie Operationen auf diesen für andere Komponenten ist. Dies betrifft gemäß der zuvor diskutierten Adaption der Datenschicht sowohl lokal in dem WebCNC gehaltene Daten als auch von Switches in Switch-internen Datenbanken gespeicherte Daten. Dafür implementieren die Komponenten der Datenschicht vordefinierte Java-Interfaces.

Damit nicht bei jedem CNC-Neustart sämtliche Switches bei dem WebCNC neu registriert werden müssen, gibt es einen persistenten Speicher, der die bereits registrierten Switches und ihre Zugangsdaten speichert. Zugriffe auf diesen persistenten Speicher werden von der *switchcredstore*-Komponente realisiert. Um unabhängig von Implementierungsdetails der Komponente von der

Anwendungsschicht aus angesprochen werden zu können, implementiert die *switchcredstore*-Komponente ein vordefiniertes Java-Interface, das Operationen zum Abfragen der gespeicherten Switch-Daten, zum Hinzufügen neuer Switches in den persistenten Speicher sowie zum dauerhaften Entfernen von Switchdaten bereitstellt. Die Art des verwendeten persistenten Speichers ist nicht definiert und lediglich ein Implementierungsdetail der Komponente. So ist die Verwendung von einfachen Text-Dateien, Remote-Datenbanken oder anderen persistenten Speicherlösungen möglich.

Um auf Anfragen des Frontends schnell reagieren zu können und eine angenehme Nutzererfahrung ohne lange Wartezeiten zu ermöglichen, müssen in der Datenschicht Informationen über den Netzwerkzustand vorgehalten werden. Dies wird von der *netstatecache*-Komponente realisiert, die Informationen zum Zustand der einzelnen Netzwerkswitches vorhält. Hier werden vor allem die zuletzt bekannten TAS-Konfigurationen und LLDP-Informationen der Switches gespeichert, damit bei Frontend-Anfragen nicht jeder Switch erneut per SNMP abgefragt werden muss. Um der Anwendungsschicht Zugriff auf diese vorgehaltenen Informationen zu ermöglichen, implementiert die *netstatecache*-Komponente ebenfalls ein Java-Interface, das Operationen zum Abfragen, Hinzufügen und Entfernen von Informationen aus dem *netstatecache* ermöglicht. Im Gegensatz zur *switchCredStore*-Komponente müssen die im *netstatecache* gehaltenen Daten nicht persistent gespeichert werden, sondern können beim CNC-Neustart verloren gehen, da sie im Rahmen der *setup*-Prozedur ohnehin neu abgefragt werden.

Als dritte und letzte Komponente der Datenschicht implementiert die *switchComms*-Komponente die Kommunikation mit den Netzwerkswitches. Auch sie implementiert ein Java-Interface, das von der Anwendungsschicht aus aufgerufen werden kann. Dabei stellt die *switchcomms*-Komponente Operationen zum Abfragen von TSN- und LLDP-Daten von Switches sowie das Setzen und Aktivieren neuer TAS-Konfigurationen auf Switches bereit. Unter Beachtung der beschriebenen Adaption der Datenschicht wird auch die *switchComms*-Komponente dieser Schicht zugeordnet, da sie den Zugriff auf in Switches gespeicherte Werte und Parameter realisiert und diese Funktionen den restlichen Komponenten über ein internes Interface bereitstellt. Dies entspricht einem Zugriff auf eine externe, in den Netzwerkswitches vorhandene Datenbank über ein Kommunikationsprotokoll und ist damit Teil der Datenschicht.

6 Entwurf und Implementierung des Backends

In diesem Kapitel werden Details zum Entwurf der in Abbildung 5.1 dargestellten Backend-Komponenten erläutert. Dabei werden insbesondere die von den Komponenten bereitgestellten Interfaces vorgestellt. Außerdem wird die Implementierung des Backends des CNC-Prototypen betrachtet.

6.1 Verwendete Technologien und Bibliotheken

Der Software-Prototyp des Backends basiert auf einer Reihe von Bibliotheken und Frameworks, die die Entwicklung eines Backends für Webanwendungen unterstützen. Diese Abhängigkeiten werden in diesem Abschnitt kurz erklärt.

Als Programmiersprache wurde für das Backend Java [GJS+21] ausgewählt. Java ist eine geeignete Programmiersprache für Anwendungen, bei denen die Portabilität eine wichtige Anforderung ist. So wird Java-Code nicht, wie etwa bei C, direkt zu Maschinencode kompiliert, sondern zu Java Bytecode. Dieser ist plattformagnostisch und damit nicht auf ein spezielles Betriebssystem oder einen speziellen Central Processing Unit (CPU)-Befehlssatz angepasst. Ausgeführt wird der Java Bytecode von der Java Virtual Machine (JVM) [LYB+21], die den Bytecode zur Laufzeit in Betriebssystem- und CPU-Befehlssatz-spezifischen Code umwandelt und ausführt. Die JVM ist das Kernelement des Java Runtime Environment (JRE), das neben der JVM noch weitere Standard-Bibliotheken enthält. Damit ist zu Java Bytecode kompilierter Java-Code auf jedem System, für das ein JRE existiert, ausführbar, was eine hohe Portabilität der Anwendung ermöglicht.

Als Basis-Framework für das Backend wird Spring Boot [WNW+] eingesetzt. Spring Boot ist ein Open-Source-Framework zur Umsetzung von Java-basierten Webanwendungen. Es vereinfacht die Integration von eigenen Software-Komponenten mit anderen Diensten wie Webservern zur Bereitstellung einer HTTP-API für Web-Anwendungen. Außerdem wird die Integration von weiteren Abhängigkeiten wie Logging-Frameworks unterstützt.

Als integrierter Webserver wird der von Spring Boot bereitgestellte Open-Source Apache Tomcat-Server [TASFc] verwendet. Dieser stellt die HTTP-API des Backends bereit, die sämtliche Anfragen vom Frontend bearbeitet. Der Tomcat-Webserver empfängt und sendet HTTP-Nachrichten, deren Nutzlast hauptsächlich aus JSON-Dateien besteht.

Zur Konvertierung von JSON-Dateien in Java-Objekte sowie zur Generierung der JSON-Dateien aus Java-Objekten wird die von Spring Boot bereitgestellte Jackson-Bibliothek [SHK+24] genutzt.

Als Logging-Framework wird Apache Log4j [TASFa] verwendet. Log4j ist ein weit verbreitetes Logging-Framework für Java-Anwendungen, das auch von Spring Boot nativ unterstützt wird.

Zur Kommunikation mit den Netzwerkswitches setzt der WebCNC-Prototyp auf SNMPv3. Zur Implementierung des SNMP Managers wird die SNMP4J-Bibliothek [Foc] genutzt. SNMP wurde als erstes unterstütztes Netzwerkkonfigurationsprotokoll gewählt, da es im Gegensatz zu dem später standardisierten NETCONF auch von vielen älteren Switch-Modellen unterstützt wird. Eine Erweiterung des WebCNC auf die Unterstützung weiterer Protokolle wie NETCONF ist jedoch möglich.

Als Build-Tool für das gesamte Backend-Projekt wird Apache Maven [TASFb] eingesetzt. Maven vereinfacht den Build-Prozess durch eine XML-basierte Konfiguration und ermöglicht das automatische Herunterladen und Einbinden von Abhängigkeiten aus Maven-Repositories.

Zur Dokumentation des Projektes wird Javadoc [Ora] eingesetzt. Javadoc ist eine von Oracle entwickelte Technologie für Java-Projekte, die es erlaubt, aus speziellen Kommentaren im Quellcode umfangreiche Dokumentationen im HTML-Format zu generieren.

Insgesamt setzt der Backend-Prototyp damit auf bewährte Technologien sowie Bibliotheken und Frameworks, die es ermöglichen, den heutigen technischen Ansprüchen an eine Web-Anwendung gerecht zu werden.

6.2 Typ-Definitionen

Das Backend ist für die Verwaltung des Netzwerkzustandes zuständig. Um die vom Backend bereitgestellten Funktionalitäten im Detail zu verstehen, ist es daher notwendig, zunächst die vom Backend verwendeten Datenstrukturen zu betrachten. Dieser Abschnitt widmet sich der Darstellung der im Backend verwendeten Datenstrukturen. Das Backend wird gemäß dem objektorientierten Programmierparadigma entwickelt. Es werden in diesem Abschnitt also Klassen betrachtet, deren instanziierte Objekte der internen Datenhaltung und -verwaltung dienen. Außerdem werden einige Hilfsklassen beschrieben.

6.2.1 Datenstrukturen für Switch-Daten

Zunächst werden die Klassen beschrieben, deren Objekt-Instanzen zur Darstellung des Netzwerkzustandes genutzt werden. Hierbei handelt es sich um drei hierarchisch aufgebaute Klassen, die in verschiedenen Granularitäten die über einzelne Netzwerkswitches gesammelten Informationen beschreiben. Außerdem gibt es eine spezielle Klasse, deren Instanzen die Kontaktinformationen von noch nicht kontaktierten Switches repräsentieren. Dargestellt sind diese Klassen als Unified Modeling Language (UML)-Klassendiagramm in Abbildung 6.1.

Diese Aufteilung der Datenstruktur in hierarchische Klassen basiert auf mehreren Überlegungen. Zunächst ist die Aufteilung in separate *Switch*-, *Port*- und *GCLEntry*-Klassen eine logische Folge der objektorientierten Programmierung. So besitzen reale TSN-Switches in der Regel mehrere gleichartige, TSN-fähige Ports. Diese Ports können gemäß IEEE 802.1Qbv [IEEE16b] als Teil ihrer TAS-Konfiguration beliebig viele, gleichartige Einträge in ihrer GCL besitzen. Diesen fundamentalen Aufbau von TSN-Switches in den Datenstrukturen des WebCNC zu repräsentieren ist ein Grund für die Aufteilung der Klassen.

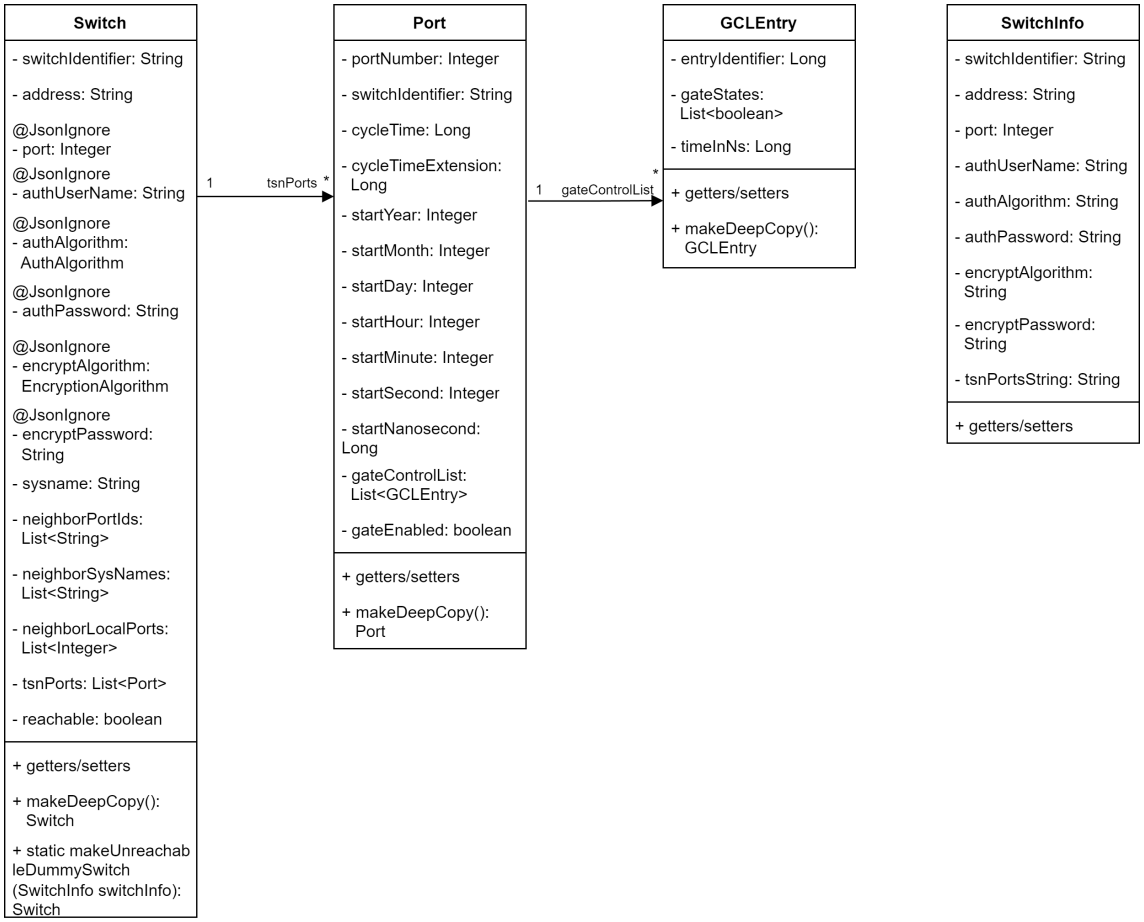

Abbildung 6.1: Klassendiagramm der Backend-Klassen für Switch-Daten

Außerdem spielt auch die Erweiterbarkeit auf andere Netzwerkkonfigurationsprotokolle wie NETCONF eine Rolle in der Entscheidungsfindung. So sind die Felder der Port- und GCL-Klassen – neben einigen ebenfalls unabhängigen Feldern zur internen Verwaltung – protokollunabhängige Repräsentationen der TAS-Konfiguration eines TSN-Ports, das es erlaubt, diese Datenstrukturen bei einer Erweiterung des WebCNC-Prototypen auf weitere Protokolle nicht modifizieren zu müssen. So müssen lediglich die *Switch*- sowie *SwitchInfo*-Klassen um weitere Felder ergänzt werden, um protokollspezifische Daten zu speichern.

Switch-Klasse

An der Spitze der Hierarchie steht die **Switch**-Klasse. **Switch**-Instanzen repräsentieren einzelne Netzwerkswitches und enthalten Informationen über Attribute, die den gesamten Switch in seiner Netzwerkumgebung beschreiben.

So gibt es einige Felder, die generelle Informationen über den Switch beinhalten. So wird das *switchIdentifier*-Feld als eindeutiger Identifikator für einen Switch verwendet. Das *address*-Feld gibt zudem die IP-Adresse an, unter welcher der Switch im Netzwerk erreichbar ist. Das *reachable*-Feld gibt an, ob der Switch beim letzten Kommunikationsversuch erreicht werden konnte oder nicht.

Die Klasse besitzt außerdem einige Felder, die Konfigurationsprotokoll-spezifische Informationen über den Switch enthalten. Da der WebCNC-Prototyp SNMPv3 unterstützt, sind dies Felder, welche die nötigen Informationen für eine SNMPv3-Verbindung beinhalten. Werden weitere Protokolle unterstützt, sind diesen Protokollen entsprechende Felder zu ergänzen. Das *port*-Feld wird genutzt, um den für SNMP-Kommunikation verwendeten UDP-Port des Switches zu speichern. Die *authUserName*-, *authAlgorithm*-, *authPassword*-, *encryptAlgorithm*- sowie *encryptPassword*-Felder geben die SNMPv3-Zugangsdaten für den Switch an. Die @JsonIgnore-Annotation gibt an, dass diese Felder bei der Konvertierung in eine JSON-Datei durch die Jackson-Bibliothek ignoriert werden sollen. Da diese Informationen im Frontend nicht benötigt werden, verbleiben die Zugangsdaten registrierter Switches im Backend. Als Authentifikationsalgorithmen werden MD5 und SHA1 unterstützt, zur Verschlüsselung können DES und AES-128 verwendet werden.

Zudem gibt es einige Felder, die Informationen zur Topologieerkennung beinhalten. Das *sysname*-Feld gibt den Systemnamen des Switches an. Die *neighborPortIds*-, *neighborSysNames*- und *neighborLocalPorts*-Listen werden genutzt, um LLDP-Informationen über benachbarte Geräte abzuspeichern. *neighborPortIds* enthält die PortIDs der benachbarten Geräte, *neighborSysNames* die Systemnamen der benachbarten Geräte und *neighborLocalPorts* die Portnummer des Switches, an dem das benachbarte Gerät angeschlossen ist. Die Einträge dieser drei Listen an Position n gehören alle zum n-ten Nachbargerät.

Um dem Switch seine zugehörigen Ports zuzuweisen, gibt es außerdem das *tsnPorts*-Feld. Dieses beinhaltet eine Liste aller TSN-fähigen Ports des Switches. Hier werden komplexe Objekte vom *Port*-Typ gespeichert.

Außerdem verfügt die *Switch*-Klasse über getter und setter für die privaten Felder. Objekte des *Switch*-Typs bieten zudem mit der *makeDeepCopy*-Methode eine Funktion an, die eine Deepcopy des Objektes anlegt. Außerdem gibt es eine statische *makeUnreachableDummySwitch*-Methode, die mit den im *SwitchInfo*-Objekt vorliegenden Switch-Zugangsdaten ein *Switch*-Objekt erstellt, dessen *reachable*-Feld auf false gesetzt ist. Diese Methode wird genutzt, um Switch-Objekte von persistent gespeicherten, aber aktuell nicht ansprechbaren Switches zu erstellen, deren TSN-Konfiguration und LLDP-Daten deshalb nicht zur Verfügung stehen.

Port-Klasse

Die im *tsnPorts*-Feld der *Switch*-Klasse gespeicherten *Port*-Objekte stellen die nächste Stufe der Klassenhierarchie dar. Instanzen der *Port*-Klasse repräsentieren einen bestimmten TSN-Port eines Switches mit allen Port-weiten TAS-Konfigurationsparametern.

Die Klasse enthält einige Felder zur internen Verwaltung. Als eindeutige Identifikatoren für ein *Port*-Objekt dienen die *portNumber*- und *switchIdentifier*-Felder, die angeben, welcher Port an welchem Switch von der Instanz repräsentiert wird.

Die restlichen Felder werden zur generischen Repräsentation der TAS-Konfiguration genutzt. Sie geben die entsprechenden TAS-Parameter des Ports an, die in Kapitel 2.3.3 beschrieben wurden. Manche dieser Parameter – beispielsweise die *BaseTime* – werden in den internen Datenstrukturen anders, aber mit gleichem Informationsgehalt dargestellt, um die Darstellung im Frontend zu erleichtern.

Die *cycleTime*- und *cycleTimeExtension*-Felder stellen die gleichnamigen TAS-Parameter dar. Die *startYear*- bis *startNanosecond*-Felder repräsentieren den *BaseTime*-Parameter des TAS. Die Speicherung als separate Parameter für Jahr, Monat, Tag, Stunde, Minute, Sekunde und Nanosekunde wurde gewählt, um die Darstellung und Manipulation der Werte durch einen Nutzer im Frontend zu vereinfachen. Das *gateControlList*-Feld ist eine Liste von Objekten des *GCLEntry*-Typs, die zusammen die TAS-GCL auf diesem Port repräsentieren. Das *gateEnabled*-Feld gibt an, ob der TAS auf diesem Port aktiv ist.

Auch besitzt die *Port*-Klasse Setter und Getter für die Felder und eine *makeDeepCopy*-Methode zum Generieren einer Deepcopy der Instanzen.

GCLEntry-Klasse

Die niedrigste Stufe in der Klassen-Hierarchie wird von der *GCLEntry*-Klasse belegt. Diese repräsentiert einen einzelnen GCL-Eintrag auf einem Port eines Switches. Identifiziert werden Instanzen dieser Klasse durch das *entryIdentifier*-Feld. Außerdem besitzen sie ein *gateStates*-Feld, das eine Liste aus acht booleschen Werten beinhaltet, die den Aktivierungszustand der Gates sieben bis null angeben. Das *timeInNs*-Feld gibt an, wie lange der GCL-Eintrag aktiv ist.

Außerdem gibt es getter und setter für die beschriebenen Felder sowie eine *makeDeepCopy*-Methode zur Erstellung von DeepCopies des *GCLEntry*-Objektes. Im Zusammenspiel beschreiben diese drei Klassen den gesamten durch den WebCNC erfassten Zustand eines Netzwerkswitches.

SwitchInfo-Klasse

Eine besondere Rolle kommt der *SwitchInfo*-Klasse zu. Sie erfasst die nötigen Informationen, um einen noch nicht kontaktierten Switch über die unterstützten Konfigurationsprotokolle zu kontaktieren und auf Basis der Antworten ein Objekt der *Switch*-Klasse zu erstellen. Auch hier sind für die Unterstützung weiterer Protokolle entsprechende Felder zu ergänzen.

Zunächst enthält die Klasse einige Felder zur internen Verwaltung. Das *switchIdentifier*-Feld enthält den eindeutigen Identifikator des anzulegenden Switches. Das *address*-Feld gibt die IP-Adresse des Switches an. Im *tsnPortsString*-Feld wird mithilfe eines speziell formatierten Strings angegeben, welche Ports des Switches TSN unterstützen. Diese werden anhand ihrer Portnummer identifiziert und durch Kommata getrennt.

Außerdem enthält die Klasse, da der WebCNC-Prototyp SNMPv3 unterstützt, einige Felder, in denen die für SNMPv3-Verbindungen benötigten Informationen abgelegt werden. Das *port*-Felder gibt den für SNMP genutzten UDP-Port an. Die *authUserName*-, *authAlgorithm*-, *authPassword*-, *encryptAlgorithm*- und *encryptPassword*-Felder speichern die SNMPv3-Zugangsdaten des Switches.

Mit diesen Informationen ist es möglich, sämtliche Informationen, die in den eingangs beschriebenen drei Klassen gespeichert werden, vom Netzwerkswitch abzufragen. *SwitchInfo*-Objekte werden beim Auslesen der persistent gespeicherten Switch-Daten sowie der Eingabe neuer Switches durch den Nutzer eingesetzt.

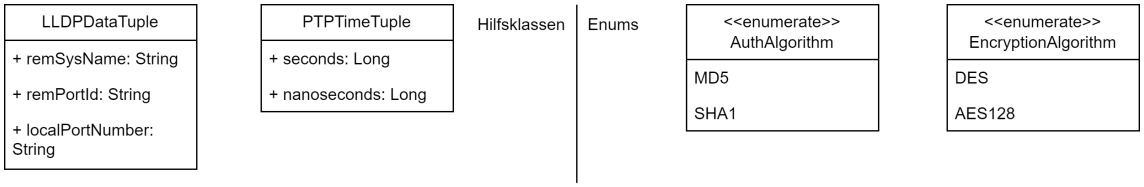

Abbildung 6.2: Hilfsklassen und Enums des Backends

6.2.2 Hilfsklassen und Enmus

Es wird eine Reihe von Hilfsklassen und Enums verwendet, um Parameter und Rückgabetypen zu organisieren. Dargestellt sind diese in Abbildung 6.2. Die *LLDPDataTuple*-Hilfsklasse besitzt Felder, in denen die über ein Nachbargerät gesammelten *lldp*-Daten gesammelt werden. Dieser wird als Rückgabetyp einer SwitchComms-Interface-Methode verwendet, die LLDP-Daten von den Switches einholt.

Die *PTPTimeTuple*-Hilfsklasse wird genutzt, um Rückgabewerte, die aus einem Sekundenwert und einem Nanosekundenwert bestehen, darzustellen. Dies wird im Kontext von PTP-Werten genutzt, welche die aktuelle Zeit in Sekunden und Nanosekunden seit dem 1. Januar 1970 angeben.

Der *AuthAlgorithm*-Enumerate stellt die unterstützten SNMPv3-Authentifikationsalgorithmen dar. Der CNC-Prototyp unterstützt hier MD5 und SHA1. Der *EncryptionAlgorithm*-Enumerate stellt entsprechend die unterstützten SNMPv3-Verschlüsselungsalgorithmen dar. Vom CNC-Prototyp werden DES und AES128 unterstützt.

6.3 netStateCache-Komponente

Die netStateCache-Komponente ist in der Datenschicht des Backends verortet. Ihre Aufgabe ist es, den zuletzt bekannten Zustand der Netzwerkswitches mit allen gesammelten Informationen zu speichern und auf Anfrage bereitzustellen. Dafür werden *Switch*-Objekte, welche die realen Netzwerkswitches mit ihrem zuletzt bekannten Zustand repräsentieren, vorgehalten. Diese Zustände vorzuhalten, ist notwendig, um Nutzern einen schnellen Zugriff auf Informationen über das Netzwerk zu gewährleisten. Müssten die Daten sämtlicher Switches bei jeder Nutzeranfrage bei den Switches neu angefragt werden, wären die Nutzer beim Öffnen der Webanwendung einer langen Wartezeit ausgesetzt. Um dies zu verhindern, werden die zuletzt bekannten Daten im netStateCache gespeichert und bei Anfrage durch den Nutzer ausgelesen.

Obwohl der netStateCache notwendig für eine angenehme Nutzererfahrung ist, kann er auch dafür sorgen, dass dem Nutzer veraltete Informationen präsentiert werden. Tritt eine Änderung des Netzwerkzustandes auf, die nicht durch den WebCNC veranlasst wurde, sind die Daten des netStateCache inkonsistent mit dem tatsächlichen Netzwerkzustand. Dieses Problem ist generell Caching-Verfahren inhärent und lässt sich unter Wahrung der Caching-Funktionalität nicht vermeiden. Es gibt jedoch einige mögliche Erweiterungen des WebCNC, die dieses Problem abmildern können.

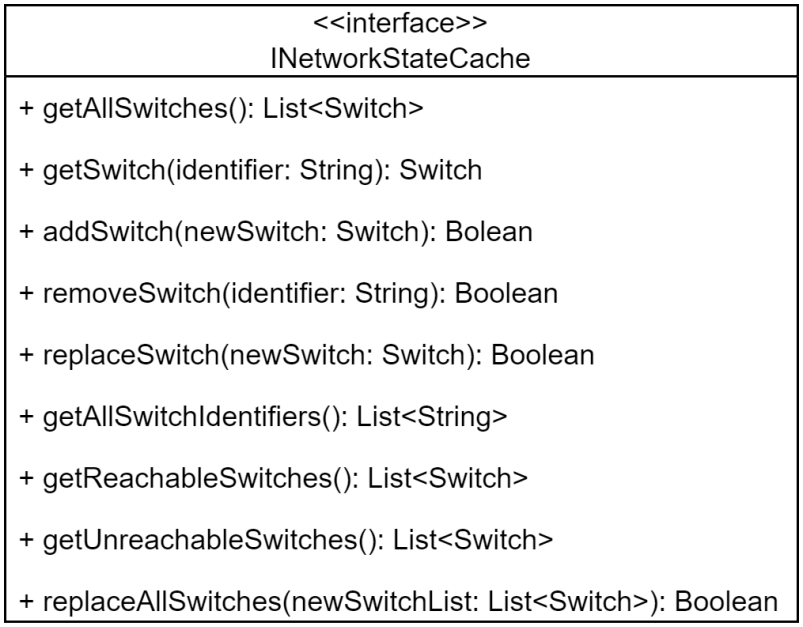

Abbildung 6.3: Interface der netStateCache-Komponente

So ließe sich einerseits eine Monitoring-Komponente entwickeln, welche durch periodisches Abfragen der Switch-Zustände die im netStateCache vorgehaltenen Switch-Daten aktualisiert. Damit wäre ein inkonsistenter Zustand maximal für die Zeitdauer des Intervalls, nach dem die Switch-Daten erneut abgefragt werden, vorhanden. Eine weitere Möglichkeit, eine solche Monitoring-Komponente umzusetzen, ist die Realisierung eines Pub/Sub-Systems, bei dem die Switches nach einer Änderung ihrer Konfiguration automatisch die Monitoring-Komponente über diese Änderung informieren. Dafür können bei der Nutzung von SNMP von der Monitoring-Komponente Traps auf den Switches registriert werden, bei NETCONF kann der *Notification*-Mechanismus genutzt werden. Aufgrund des Umfangs der Arbeit ist eine solche Monitoring-Komponente nicht Teil des WebCNC-Prototypen, es kann jedoch zukünftig als Erweiterung eingebracht werden.

Zur Modifikation und Abfrage der im netStateCache gespeicherten Daten implementiert ein netStateCache das INetworkStateCache-Interface. Dieses Interface ist in Abbildung 6.3 dargestellt. Die *getAllSwitches*-Methode wird genutzt, um sämtliche Switches als Liste von Objekten der Switch-Klasse abzufragen. Mit der *getSwitch*-Methode kann ein einzelner Switch auf Basis seines *switchIdentifier*-Feldes abgefragt werden. Die *getReachableSwitch*- sowie *getUnreachableSwitch*-Operationen filtern vor der Rückgabe die gespeicherten Switches nach dem Wert ihrer *reachable*-Flag. Die *getAllSwitchIdentifiers*-Operation liefert als Rückgabewert lediglich eine Liste der dem netStateCache bekannten *switchIdentifier*. Die *addSwitch*- und *removeSwitch*-Methoden dienen dazu, einzelne Switch-Objekte aus dem netStateCache zu entfernen oder dem netStateCache hinzuzufügen. Die *replaceSwitch*-Operation erlaubt es, ein bestehendes Switch-Objekt durch eine aktualisierte Version mit neueren Zustandsdaten zu ersetzen. Dafür werden vor dem Hinzufügen des neuen Objektes die bestehenden Switches auf Basis ihres *switchIdentifier* durchsucht. Hat ein altes Switch-Objekt den gleichen *switchIdentifier* wie der neue Switch, wird es entfernt. Die *replaceAllSwitches*-Methode wird genutzt, um den netStateCache zu leeren und mit komplett neuen Daten zu befüllen. Dies kommt bei einer vollständigen Aktualisierung des Netzwerkzustandes zum

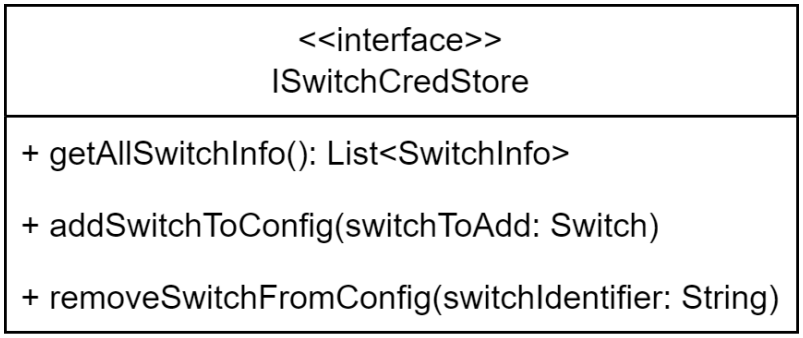

Abbildung 6.4: Interface der switchCredStore-Komponente

Einsatz. Die vom netStateCache gespeicherten Daten müssen nicht über Neustarts hinweg persistent sein, da beim Start des WebCNC der aktuelle Zustand sämtlicher registrierter Switches ohnehin neu abgefragt wird.

Da es im Kontext einer Webanwendung möglich ist, dass sich Anfragen von mehreren Nutzern zeitlich überschneiden, ist die netStateCache-Komponente außerdem verantwortlich für eine threadsichere Verwaltung der vorgehaltenen Daten. So muss insbesondere vermieden werden, dass gleichzeitige Schreibanfragen *A* und *B* nicht zu einer Situation führen, in der nach Ausführung der Schreibzugriffe Teile der Daten von Anfrage *A* und Teile von Anfrage *B* im netStateCache vorliegen. Es muss also durch die Implementierung des *INetStateCache*-Interfaces sichergestellt sein, dass solche Schreibanfragen sequentiell abgearbeitet werden. Als Implementierungdetail der Komponente können sich andere Komponenten darauf verlassen, dass diese Threadsicherheit gegeben ist.

Der WebCNC-Prototyp nutzt die NetworkStateCache-Klasse, um die netStateCache-Komponente zu implementieren. Dafür wird von der NetworkStateCache-Klasse das INetworkStateCache-Interface implementiert. Zur Speicherung der Switch-Daten wird eine einfache Liste von Switch-Objekten genutzt, auf die indirekt über die im Interface definierten Operationen zugegriffen werden kann. Damit handelt es sich um eine simple In-Memory-Implementierung. Zur Sicherstellung der Threadsicherheit werden die Methoden des Interfaces in der Implementierung mit dem *synchronized*-Schlüsselwort gekennzeichnet. Operationen, die als *synchronized* markiert sind, warten vor der Ausführung darauf, das Lock des Objekt-Monitors zu erhalten. Damit ist innerhalb eines Objektes keine gleichzeitige Ausführung mehrerer als *synchronized* gekennzeichneten Methoden möglich. Für den Einsatz in großen Netzwerken oder beim Betrieb des WebCNC-Backends auf einem ressourcenarmen System ist auch eine Auslagerung des Switch-Speichers in eine externe Datenbank denkbar.

6.4 switchCredStore-Komponente

Die switchCredStore-Komponente ist eine weitere Komponente der Datenschicht. Ihre Aufgabe ist es, die Zugangsdaten von angelegten Switches persistent zu speichern und diesen Speicher zu modifizieren und auszulesen. Diese Komponente wird benötigt, da es nicht wünschenswert ist, bei jedem Neustart des WebCNC die Zugangsdaten aller Netzerkswitches erneut konfigurieren zu müssen. Um dies zu erreichen, müssen sämtliche Informationen, die auch in den Feldern eines

SwitchInfo-Objektes vorliegen (siehe Abbildung 6.1), persistent gespeichert werden. Damit muss die switchCredStore-Komponente den Zugriff auf den persistenten Speicher verwalten. Dies kann sowohl durch eine lokale Implementierung des Speichers in der Komponente als auch durch Zugriff auf eine externe Datenbank geschehen.

Um anderen Komponenten Zugriff auf die vom switchCredStore verwalteten Daten zu gewähren, muss ein switchCredStore das ISwitchCredStore-Interface implementieren. Dieses ist in Abbildung 6.4 dargestellt. Es besteht aus lediglich drei Operationen zum Auslesen und Modifizieren der persistent gespeicherten Daten. Die *getAllSwitchInfo*-Methode liest alle Switch-Zugangsdaten aus dem Speicher aus und gibt sie als Liste von SwitchInfo-Objekten zurück. Dies wird genutzt, um beim Start oder Aktualisieren der gesamten WebCNC die anzufragenden Switches zu bestimmen. Die *addSwitchToConfig*-Methode akzeptiert ein Switch-Objekt als Parameter und speichert dieses im persistenten Speicher ab. Dies geschieht, wenn der Nutzer im Frontend einen neuen Switch anlegt, ihn an die HTTP-API des Backend sendet und diese ihn anschließend zur persistenten Speicherung dem switchCredStore übergibt. Zum Entfernen gespeicherter Switches wird die *removeSwitchFromConfig*-Operation genutzt. Analog zum Hinzufügen eines Switches ist es dem Nutzer im Frontend möglich, angelegte Switches dauerhaft aus der Switch-Konfiguration des WebCNC zu entfernen. Um den zu entfernenden Switch zu identifizieren, akzeptiert die Methode einen *switchIdentifier*-String.

Die switchCredStore-Komponente muss sicherstellen, dass Änderungen an den gespeicherten Daten threadsicher ausgeführt werden. So darf es nicht dazu kommen, dass mehrere, gleichzeitig eingehende Anfragen *A* und *B* zum Hinzufügen bzw. Entfernen von Einträgen aus dem persistenten Speicher dazu führen, dass die Änderungen durch Anfrage *A* durch das Abschließen von Anfrage *B* überschrieben werden. Da dies ein Implementierungsdetail der Komponente ist, können sich aufrufende Komponenten auf die Sicherstellung der Threadsicherheit verlassen.

Die Funktionalität der switchCredStore-Komponente wird im WebCNC-Prototyp von der SwitchCredStore-Klasse übernommen. Diese implementiert das zuvor beschriebene ISwitchCredStore-Interface. Threadsicherheit wird erreicht, indem die Operationen der switchCredStore-Klasse als *synchronized* markiert werden. Der Prototyp nutzt für die persistente Speicherung der registrierten Switches eine einfache Textdatei. In dieser werden die Zugangsdaten der Switches blockweise untereinander gespeichert. Zum Auslesen der Datei implementiert der Prototyp einen eigenen Textdatei-Parser. Die Implementierung der switchCredStore-Komponente mit einer Textdatei hat den Vorteil, dass sie auch manuelle Anpassungen ohne großen Aufwand erlaubt. So können unabhängig vom Frontend Switches direkt in die Konfigurationsdatei hinzugefügt oder aus dieser entfernt werden. Für den Einsatz in großen Netzwerken mit vielen Netzwerkswitches ist jedoch auch eine Implementierung der switchCredStore-Komponente, die auf eine externe Datenbank setzt, denkbar.

6.5 switchComms-Komponente

Die switchComms-Komponente ist die letzte Komponente der in Kapitel 5 eingeführten, adaptierten Datenschicht. Ihre Aufgabe ist es, die Kommunikation mit den Switches umzusetzen. Die switchComms-Komponente wird dieser adaptierten Datenschicht zugeordnet, da sie Operationen auf in Switches gespeicherte Daten realisiert und diese Operationen anderen Komponenten über ein Interface bereitstellt. Im Gegensatz zu den netStateCache- und switchCredStore-Komponenten, die

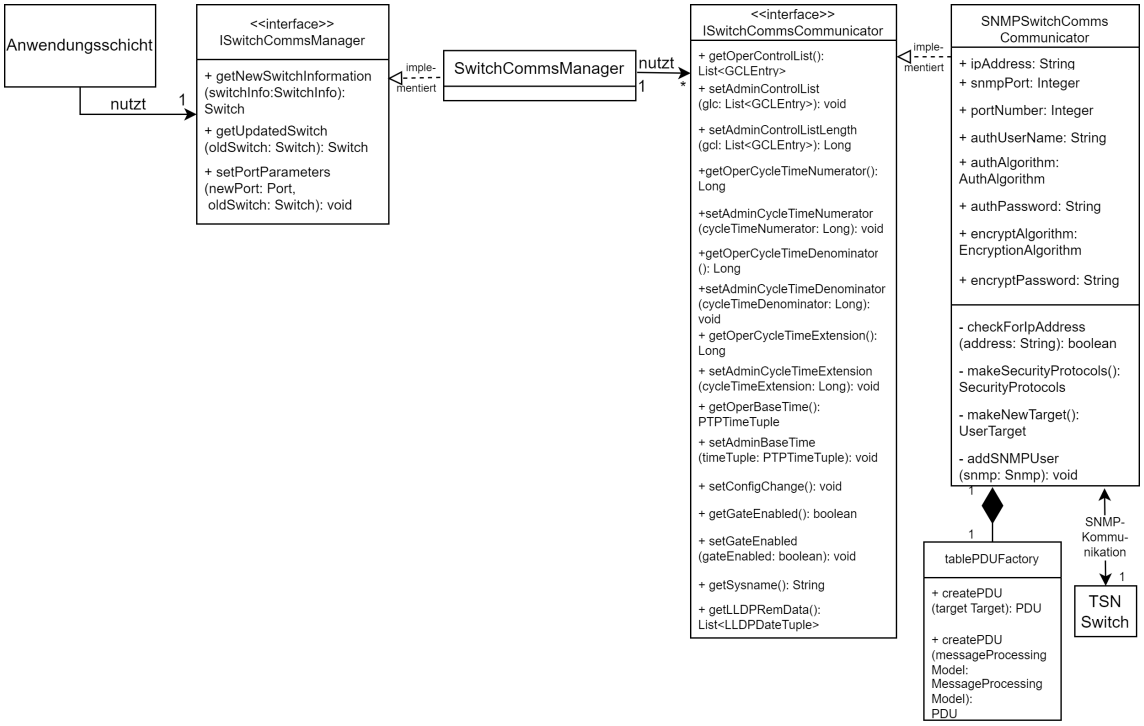

Abbildung 6.5: Aufbau der switchComms-Komponente

lediglich aus einem Interface und einer das Interface implementierenden Klasse bestehen, ist der Aufbau der switchComms-Komponente komplexer. Sie besteht intern aus zwei Subkomponenten – dem switchCommsManager und dem switchCommsCommunicator – die verschiedene Abstraktionsstufen der Switch-Kommunikation realisieren. Dargestellt ist dieser Aufbau in Abbildung 6.5.

6.5.1 switchCommsCommunicator-Subkomponente

Die innere Subkomponente ist die switchCommsCommunicator-Subkomponente. Sie stellt Operationen zum Lesen und Setzen einzelner Werte von Netzwerkswitches bereit und implementiert dazu das Southbound-Interface zu den Switches. Aufgerufen werden diese Operationen nicht von anderen Komponenten, sondern lediglich von der switchCommsManager-Subkomponente. Die bereitgestellten Operationen werden im ISwitchCommsCommunicator-Interface definiert. So gibt es Operationen zum Auslesen und Setzen der in Kapitel 2.3.3 beschriebenen TAS-Parameter, die es ermöglichen, die TSN-Konfiguration eines Ports zu identifizieren und zu modifizieren. Außerdem wird eine Operation bereitgestellt, die es ermöglicht, die mittels LLDP gesammelten Daten über die *ChassisID*, *PortID* sowie den Systemnamen der Nachbargeräte eines Netzwerkswitches abzufragen. Außerdem kann der Systemname des Switches selbst abgefragt werden. Details zu den bereitgestellten Operationen sind in Abbildung 6.5 abgebildet.

Für die Implementierung des ISwitchCommsCommunicator-Interfaces mittels SNMP nutzt der WebCNC-Prototyp die SNMPSwitchCommsCommunicator-Klasse. Zur Realisierung der SNMP-Kommunikation wird die SNMP4J-Bibliothek [Foc] verwendet. Die

SNMPSwitchCommsCommunicator-Klasse beinhaltet neben den Operationen des implementierten Interfaces eine Reihe weiterer Felder und Hilfsoperationen, die aufgrund der Implementierung mit SNMP benötigt werden. So identifizieren die *ipAddress*- und *snmpPort*-Felder zusammen mit den *authUserName*-, *authAlgorithm*-, *authPassword*-, *encryptAlgorithm*- und *encryptPassword*-Feldern die für SNMP-Kommunikation mit dem Switch notwendigen Informationen und Zugangsdaten. Das *portNumber*-Feld bestimmt, welcher Port des Switches bei TSN-bezogenen Operationen angesprochen werden soll. Da dieses Feld im Konstruktor des SNMPSwitchCommsCommunicators gesetzt wird, wird für jeden Port eines Switches ein separater switchCommsCommunicator eingesetzt. Dieses Feld wird benötigt, da die OIDs der MIB-Objekte mit einem objektspezifischen Präfix gefolgt von der Portnummer gebildet werden: <*Objektpräfix*>.<*Portnummer*> .

Ferner werden einige Hilfsklassen implementiert, die wiederkehrende Abläufe in der Verwendung der SNMP4J-Bibliothek auslagern, um Code-Duplikate zu vermeiden. Für die Abfrage der LLDP-Daten, die in MIB-Tabellen gespeichert sind, werden die von SNMP4J bereitgestellten TableUtils-Funktionen verwendet. Diese implementieren den in Kapitel 2.1.3 beschriebenen Prozess zum Durchsuchen einer MIB-Tabelle mit *getNextRequest*-Nachrichten. Damit kann in der SwitchCommsCommunicator-Klasse von diesem Prozess abstrahiert werden, indem lediglich die Tabellen-Basis-OID und die Spaltennummer an die von SNMP4J bereitgestellten Operationen übergeben wird. Für die Verwendung dieser Bibliotheksfunktionen besitzt die SwitchCommsCommunicator-Klasse eine interne tablePDUFactory-Klasse.

Da die Threadsicherheit von der übergeordneten switchCommsManager-Subkomponente übernommen wird, werden in der switchCommsCommunicator-Subkomponente keine Synchronisationsmechanismen benötigt. Im Fehlerfall werden Exceptions des *CommsException*-Typs oder seiner Subtypen geworfen, die vom Aufrufer der switchCommsCommunicator-Komponente kompensiert oder weiterpropagiert werden können. Diese Exceptions beinhalten in ihrem *message*-Attribut weitergehende textuelle Informationen über den aufgetretenen Fehler, die zur Information des Nutzers bis ins Frontend propagiert werden können.

6.5.2 switchCommsManager-Subkomponente

Von der Anwendungsschicht wird zur Kommunikation mit den Switches das ISwitchCommsManager-Interface der switchCommsManager-Subkomponenten aufgerufen. Dieses definiert Switch-Operationen auf einer hohen Abstraktionsstufe, bestimmt die dazu notwendigen Schritte auf einer niedrigeren Abstraktionsstufe und koordiniert die Umsetzung dieser mittels eines switch-CommsCommunicators.

Die *getNewSwitchInformation*-Operation des ISwitchCommsManager-Interfaces stellt die Funktionalität bereit, den aktuellen Zustand eines Netzwerkswitches mit seiner TSN-Konfiguration und den gesammelten LLDP-Daten abzufragen, in einem neuen *Switch*-Objekt abzulegen und dieses zurückzugeben. Dafür wird ein *SwitchInfo*-Objekt akzeptiert, das die nötigen Zugangsdaten für den Switch enthält. Die Operation wird genutzt, um den Zustand eines noch unbekannten Switches abzufragen und ein entsprechendes *Switch*-Objekt zu erstellen, das diesen realen Switch mit seinem aktuellen Zustand repräsentiert. Dieses muss vom Aufrufer anschließend im netStateCache abgelegt werden, damit der Switch-Zustand für Anfragen von Clients zur Verfügung steht.

Eine ähnliche Funktionalität stellt die *getUpdatedSwitch*-Operation bereit. Sie fragt ebenfalls den aktuellen Zustand eines Switches ab, akzeptiert als Parameter jedoch ein *Switch*-Objekt statt eines *SwitchInfo*-Objektes. Dieses *Switch*-Objekt enthält gemäß der Ausführungen in Kapitel 6.2.1 ebenfalls die nötigen Zugangsdaten, um den Switch zu kontaktieren, existiert jedoch nur, wenn der Switch bereits zu einem früheren Zeitpunkt mit der *getNewSwitchInformation*-Operation erstmalig kontaktiert wurde. Zurückgegeben wird ein neues *Switch*-Objekt mit dem erhaltenen Zustand des Switches. Dieses ersetzt das nun veraltete *Switch*-Objekt, das der Operation als Parameter übergeben wurde. Anschließend muss diese aktualisierte Repräsentation des realen Switches vom Aufrufer im netStateCache abgelegt werden, damit die aktuellste Version des Switch-Zustandes für Anfragen von Clients zur Verfügung steht. Damit wird diese Operation genutzt, um den zuletzt gespeicherten Zustand eines bereits früher kontaktierten Switches zu aktualisieren.

Zudem wird die *setPortParameters*-Operation bereitgestellt, deren Zweck es ist, die TAS-Konfiguration eines Ports auf neue Werte zu setzen. Dazu wird als Parameter ein *Port*-Objekt mit der neuen Konfiguration sowie das zugehörige, alte *Switch*-Objekt akzeptiert. Hier muss der Aufrufer nach erfolgreichem Abschluss der Operation ein aktualisiertes *Switch*-Objekt, das die neu gesetzte Port-Konfiguration enthält, an den netStateCache übergeben, damit die neue Konfiguration des Ports den Clients bei entsprechenden Anfragen geliefert werden kann.

Treten bei einer dieser Operationen Fehler auf, werden Exceptions des *CommsException*-Typs an den Aufrufer der switchCommsManager-Komponente propagiert, die in ihrem *message*-Feld nähere Informationen über die Fehlerursache enthalten.

Die zuvor beschriebenen, vom switchCommsManager angebotenen Switch-Operationen werden vom switchCommsManager in einzelne Schritte aufgeteilt, die jeweils einen Wert vom Switch auslesen oder auf dem Switch setzen. Um die zur Ausführung der einzelnen Schritte benötigten Anfragen an den Switch durchzuführen, wird vom switchCommsManager ein switchCommsCommunicator genutzt. Ein solcher switchCommsCommunicator setzt, wie im vorherigen Abschnitt beschrieben, einzelne Anfragen an den Switch über ein spezifisches Kommunikationsprotokoll um. Wenn mehrere Kommunikationsprotokolle unterstützt werden, ist es Aufgabe des switchCommsManager, aus den Implementierungen des ISwitchCommsCommunicator-Interfaces für verschiedene Protokolle die für den betreffenden Switch passende Implementierung zu wählen. Gibt es beispielsweise einen SNMPSwitchCommsCommunicator und einen NETCONFSwitchCommsCommunicator, wird abhängig davon, welches Protokoll der zu kontaktierende Switch unterstützt, vom switchCommsManager eine dieser Implementierungen gewählt und genutzt.

Dieser komplexe Aufbau der switchComms-Komponente wurde gewählt, um die Wiedeverwendbarkeit im Falle einer Erweiterung auf neue Kommunikationsprotokolle zu verbessern. So muss eine neuen switchCommsCommunicator-Komponente, welche das hinzuzufügende Protokoll implementiert, entwickelt werden. Außerdem muss der SwitchCommsManager angepasst werden, um aus den alten und neuen Implementierungen des switchCommsCommunicator die passende auszuwählen. Die Kommunikatoren für bestehende Protokolle können jedoch unverändert weiterverwendet werden. Wird der WebCNC-Prototyp beispielsweise um NETCONF erweitert, kann der SNMPSwitchCommsCommunicator ohne jegliche Anpassungen weiterhin für SNMP-Kommunikation genutzt werden.

Eine weitere Aufgabe der switchCommsManager-Subkomponente ist es, die Threadsicherheit der Switch-Parameter-Operationen sicherzustellen. Insbesondere können ohne Sicherstellung der Threadsicherheit Probleme entstehen, wenn parallele Anfragen *A* und *B* zur Konfigurationsänderung

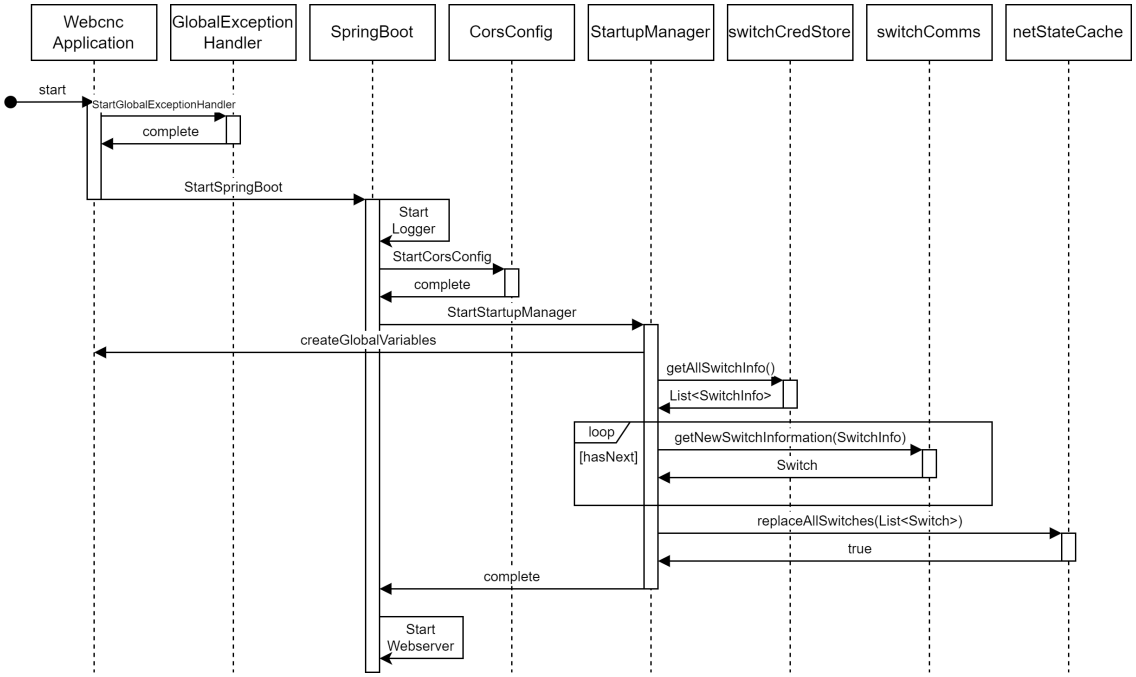

Abbildung 6.6: Sequenzdiagramm der WebCNC-Initialisierung

des gleichen Ports eingehen. Die Änderungen der TSN-Konfiguration umfasst das Setzen mehrerer TAS-Parameter, wobei der zuletzt gesetzte Wert aktiv ist. Wird im Rahmen einer nebenläufigen Ausführung ein Teil der Werte von Konfiguration *A*, dann die vollständige Konfiguration *B* und anschließend der Rest der Werte von Konfiguration *A* auf dem selben Port gesetzt, führt dies im Ergebnis zu einer Konfiguration, die aus Parameter-Werten beider Konfigurationen besteht. Dies zu vermeiden ist ein Implementierungsdetail der netStateCache-Komponente und kann von anderen Komponenten, die auf das *INetStateCache*-Interface zugreifen, als gegeben angenommen werden. Dazu setzt die SwitchCommsManager-Klasse des WebCNC-Prototyps, welche das ISwitchCommsManager-Interface implementiert, auf *synchronized*-Schlüsselwörter, die eine gleichzeitige Ausführung mehrerer Operationen eines SwitchCommsManager verhindern.

6.6 setup-Komponente

Die setup-Komponente ist Teil der Anwendungsschicht. Die in ihr befindlichen Klassen setzen Aufgaben um, die beim Start der Anwendung ausgeführt werden müssen. Die setup-Komponente ist die einzige Komponente des Backends, die keine Schnittstellen für weitere Komponenten bereitstellt. Stattdessen wird sie automatisch beim Start des WebCNC ausgeführt. Die Klassen dieser Komponente und ihre Funktionalitäten werden in den folgenden Abschnitten beschrieben. Außerdem ist der konkrete Ablauf beim Start des WebCNC in Abbildung 6.6 dargestellt. Die in der Abbildung von SpringBoot ausgehenden Aktionen werden von SpringBoot automatisch auf Basis von Annotationen der aufgerufenen Klassen ausgeführt.

6.6.1 WebcncApplication-Klasse

Die *WebcncApplication*-Klasse ist die Hauptklasse des WebCNC-Backends. Hier werden einige global für das gesamte Backend gültige Variablen gehalten. So werden hier die Instanzen der netStateCache-, switchCredStore-, und switchCommsManager-Komponenten referenziert. Außerdem wird hier das Passwort, das zur Zugriffskontrolle genutzt wird, gehalten. Ferner werden hier auch die *Retries*- und *Timeout*-Werte abgelegt, die bestimmen, wie oft die Kommunikation mit einem Switch beim Nichteintreffen einer Antwort erneut versucht werden soll, und wie lange auf eine Nachricht gewartet werden soll. In der *main*-Funktion der Klasse, die beim Start der Anwendung ausgeführt wird, wird zunächst der nachfolgend erklärte *GlobalExceptionHandler* aktiviert, und anschließend die Spring-Anwendung [WNW+] gestartet. Hierbei werden von Spring Boot die benötigten Abhängigkeiten, wie zum Beispiel der eingebettete Webserver, gestartet.

6.6.2 GlobalExceptionHandler-Klasse

Wie bereits in Kapitel 6.1 beschrieben, wird als Logging-Framework Log4j [TASFa] verwendet. Ohne weitere Konfiguration werden *Runtime Exceptions*, die das Programm zum Absturz bringen, nicht geloggt. Dies behebt der *GlobalExceptionHandler*. Diese Klasse implementiert das *java.lang.Thread.UncaughtExceptionHandler*-Interface und überschreibt die *uncaughtException*-Methode. Diese Methode wird aufgerufen, bevor das Programm nach einer nicht aufgefangenen *Runtime Exception* abstürzt. Der *GlobalExceptionHandler* ruft in dieser Methode den Logger auf und übergibt die aufgetretene Exception, um sie zu loggen. Nach Ausführung dieser Methode stürzt das Programm zwar weiterhin ab, jedoch ist die dem Absturz zugrundeliegende Exception nun in den Logs sichtbar.

6.6.3 CorsConfiguration-Klasse

Die *CorsConfiguration*-Klasse stellt eine Funktion zur Anpassung der Cross-Origin Resource Sharing (CORS)-Konfiguration der Spring-Anwendung bereit. CORS ist ein Mechanismus, der es erlaubt, den Zugriff auf die bereitgestellte HTTP-API zu beschränken. Ohne weitere Konfiguration sind in Spring Boot-Anwendungen nur Zugriffe von der gleichen Adresse und dem gleichen Port erlaubt. Da das Frontend jedoch unabhängig vom Backend von einem anderen Server unter einem anderen Port an Clients ausgeliefert werden kann, ist es notwendig, die CORS-Konfiguration anzupassen. Um größtmögliche Flexibilität zu erreichen, wird CORS so konfiguriert, dass Zugriffe von überall auf die HTTP-API möglich sind. In einem konkreten Deployment-Szenario ließe sich diese Konfiguration modifizieren, um beispielsweise nur bekannten Servern Zugriff zu gewähren.

6.6.4 StartupManager-Klasse

Die *StartupManager*-Klasse implementiert die Kernlogik des WebCNC beim Start der Anwendung. Da sie mit der *Component*-Annotation versehen ist und das *ApplicationRunner*-Interface implementiert, wird ihre *run*-Methode vom Spring-Framework automatisch beim Start der Spring-Anwendung ausgeführt.

Im ersten Schritt werden zunächst die Instanzen der Datenschicht-Komponenten erstellt und in den Variablen der *WebcncApplication*-Klasse abgespeichert. Die restlichen Felder der *WebcncApplication* werden anschließend mit Werten belegt, die dem WebCNC-Backend als Startparameter übergeben werden können. So kann das Passwort die Anzahl der Verbindungsversuche sowie das Timeout-Zeitfenster konfiguriert werden. Um das Setup des WebCNC zu erleichtern, handelt es sich hierbei um optionale Startparameter, an deren Stelle Standardwerte genutzt werden, wenn sie nicht explizit angegeben werden.

In einem zweiten Schritt wird der netStateCache auf Nutzeranfragen vorbereitet. Dazu werden zunächst die persistent gespeicherten Switch-Zugangsdaten mithilfe der switchCredStore-Komponente ausgelesen. Dann wird der Zustand der Switches mithilfe der switchComms-Komponente abgefragt und anschließend in der netStateCache-Komponente abgespeichert. Ist ein Switch nicht kontaktierbar, wird ein Dummy-Objekt der *Switch*-Klasse erstellt, dessen *reachable*-Flag den Wert *false* annimmt. Damit kann der Nutzer im Frontend darüber informiert werden, dass ein bestimmter persistent gespeicherter Switch nicht kontaktiert werden konnte.

6.7 restcontroller-Komponente

Eine weitere Komponente der Anwendungsschicht ist die restcontroller-Komponente. Ihre Aufgabe ist es, eine HTTP-API bereitzustellen, über die das Frontend des WebCNC die Funktionalitäten des Backends nutzen kann. Dafür werden Representational State Transfer (REST)-Controller definiert, die als HTTP-Endpunkte fungieren. Diese REST-Controller werden über entsprechende Klassen- und Methoden-Annotationen vom Spring-Framework automatisch als HTTP-Endpunkte erkannt und über den eingebetteten Webserver - im WebCNC-Prototyp Apache Tomcat [TASFc] - bereitgestellt. Beim Eingang einer HTTP-Anfrage für einen Endpunkt an dem Webserver werden die empfangenen Daten an den Controller übermittelt, der im Controller definierte Code ausgeführt und anschließend eine Antwort an den Aufrufer des HTTP-Endpunktes zurückgesendet. Der WebCNC-Prototyp ist dazu konfiguriert, mittels TLS verschlüsselte HTTPS-Kommunikation mit dem Frontend auf Basis von HTTP Version 2 oder niedriger zu ermöglichen. Damit entspricht der Prototyp modernen Standards an die Sicherheit von Web-Anwendungen. Die REST-Controller sind in separate Klassen ausgelagert, die im Folgenden betrachtet werden.

6.7.1 ValidatePasswordController

Der *ValidatePasswordController* stellt den einfachsten bereitgestellten HTTP-Endpunkt dar. Er empfängt eine Nachricht, die ein Passwort enthält, vergleicht dieses mit dem in der *WebcncApplication*-Klasse gespeicherten Passwort und antwortet entweder mit einer Bestätigung oder einer Fehlermeldung. Dies wird genutzt, um das im Frontend eingegebene Passwort zu validieren.

6.7.2 GetSwitchesController

Der *GetSwitchesController* ermöglicht es, den im netStateCache gespeicherten Netzwerkzustand abzufragen. Zur Zugriffskontrolle wird als HTTP-Header das Passwort mitgeliefert. Ist das Passwort als korrekt validiert worden, werden die im netStateCache gespeicherten *Switch*-Objekte abgefragt

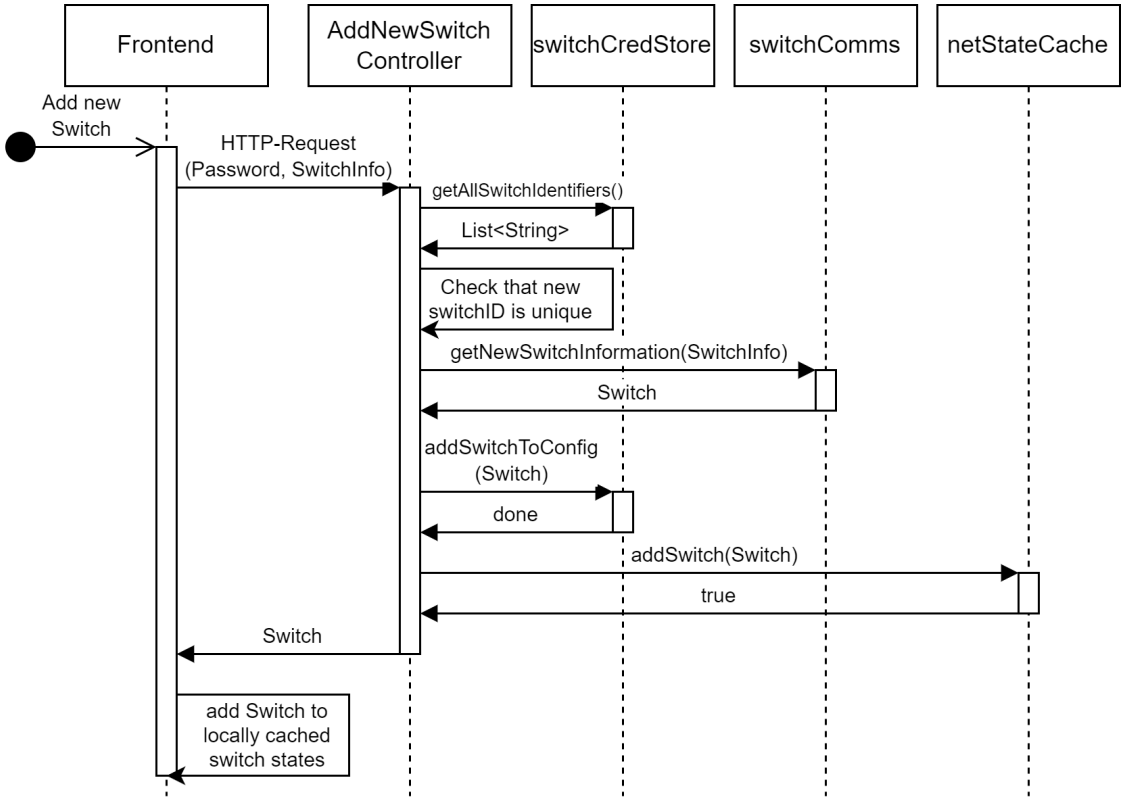

Abbildung 6.7: Sequenzdiagramm einer AddSwitchController-Anfrage

und im JSON-Format zurückgesendet. Die Transformation von der internen Repräsentation der Switch-Liste in ein JSON-Dokument wird von Spring mithilfe der Jackson-Bibliothek [SHK+24] übernommen. Dabei werden die in Abbildung 6.1 mit der *JsonIgnore*-Annotation versehenen Felder ignoriert, um nicht benötigte, sensible Zugangsdaten dem Frontend-Client nicht zur Verfügung zu stellen.

6.7.3 AddSwitchController

Der *AddSwitchController* erlaubt es, neue Switches aus dem Frontend über die HTTP-API anzulegen. Der Ablauf der Controllerausführung – ausgenommen verschiedener Fehlerfälle – ist in Abbildung 6.7 dargestellt. Dazu akzeptiert der Controller neben dem Zugangspasswort auch ein *SwitchInfo*-Objekt im JSON-Format. Dieses wird mittels Jackson in ein internes *SwitchInfo*-Objekt überführt. Anschließend wird der Switch vom Controller mithilfe der switchComms-Komponente kontaktiert. Ist dies erfolgreich, werden seine Zugangsdaten persistent mithilfe der switchCredStoret-Komponente abgespeichert und sein aktueller Zustand im netStateCache abgelegt. Anschließend wird das neue *Switch*-Objekt an das Frontend zurückgesendet. Konnte der Switch nicht kontaktiert werden, wird eine Antwort mit einer Fehlermeldung zurückgesendet.

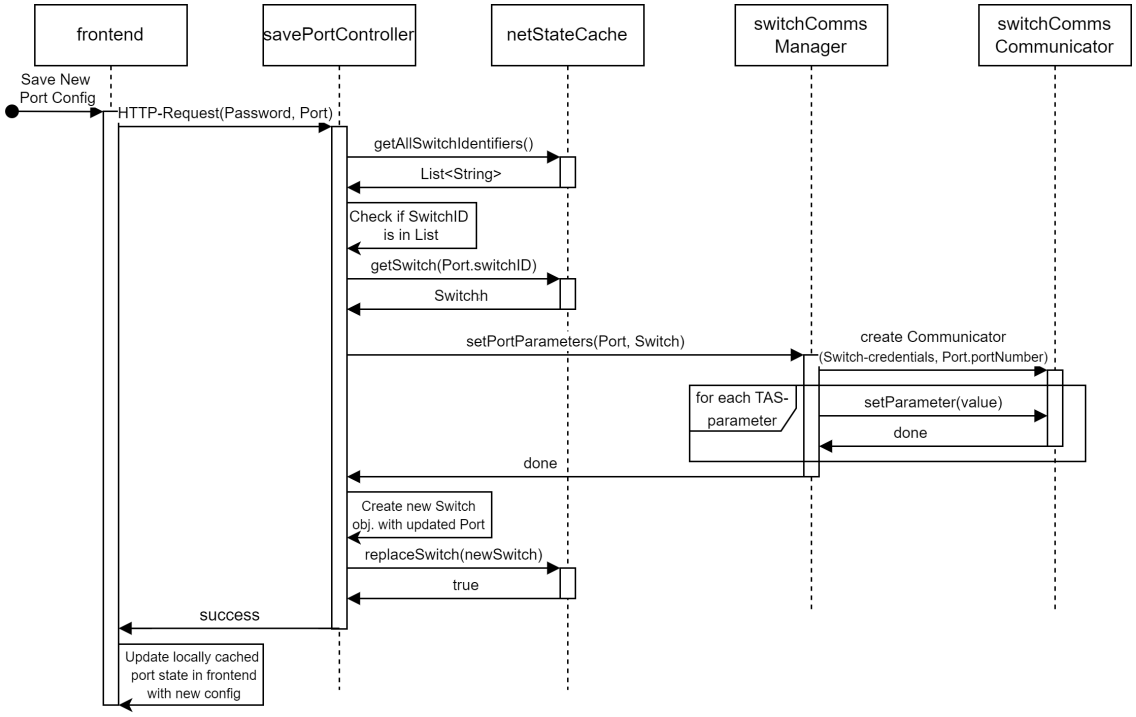

Abbildung 6.8: Sequenzdiagramm einer SavePortController-Anfrage

6.7.4 RemoveSwitchController

Der *RemoveSwitchController* arbeitet als Gegenstück zum vorher beschriebenen *AddSwitchController*. Er erlaubt es, die Daten eines Switches dauerhaft aus dem netStateCache und dem persistenten Speicher der switchCredStore-Komponente zu entfernen. Dafür erwartet er als Parameter neben dem Passwort auch den Identifier des Switches. Konnte der zu entfernende Switch nicht im Backend gefunden werden, wird eine entsprechende Fehlermeldung zurückgesendet.

6.7.5 SavePortController

Um die TSN-Konfiguration eines Ports zu modifizieren, wird der *SavePortController* aufgerufen. Der generelle Ablauf einer Anfrage – ohne möglicherweise auftretende Fehler – an diesen Controller ist in Abbildung 6.8 dargestellt.

Der Controller akzeptiert neben dem Passwort auch das mit der neuen Konfiguration versehene *Port*-Objekt im JSON-Format. Diese neue Konfiguration wird mithilfe der switchComms-Komponente auf dem Switch gesetzt. Ist dies erfolgreich, wird die neue Switch-Konfiguration im netStateCache gespeichert und dem Aufrufer des HTTP-Endpunktes wird eine Bestätigung zurückgesendet. Tritt ein Fehler auf, wird stattdessen eine Fehlermeldung zurückgesendet. Im Falle einer *CommsException* werden dabei detaillierte Informationen über die Art des Fehlers mitgeteilt.

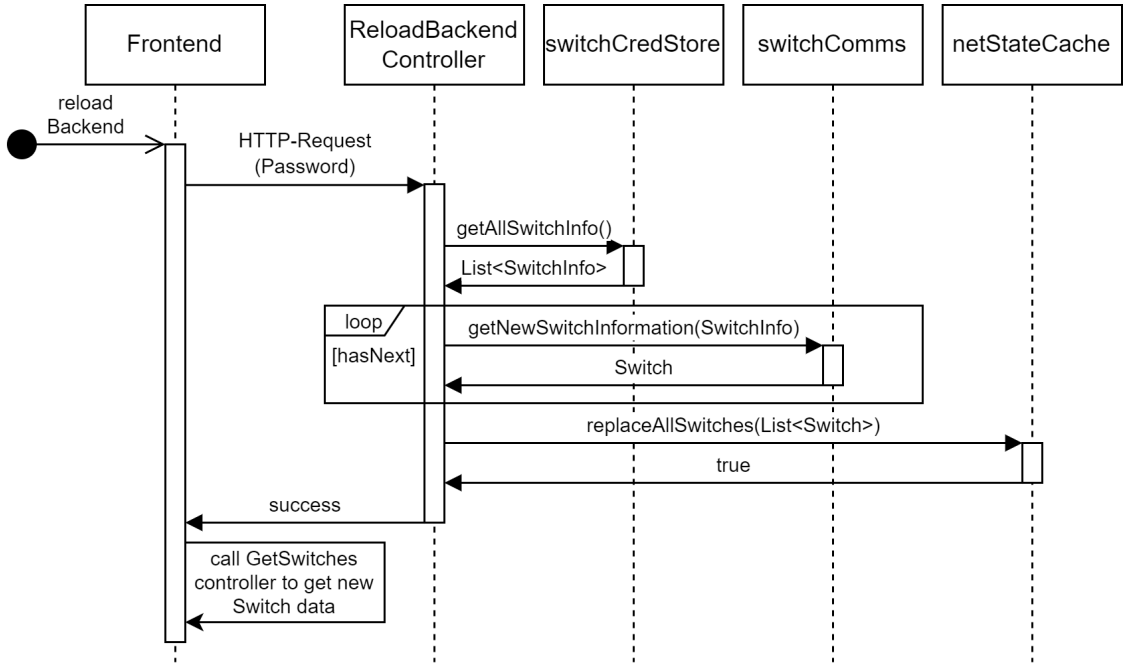

Abbildung 6.9: Sequenzdiagramm einer ReloadBackendController-Anfrage

6.7.6 ReloadBackendController

Als letzter REST-Controller wird der *ReloadBackendController* bereitgestellt. Er ermöglicht es, über die HTTP-API eine vollständige Aktualisierung des netStateCache zu bewirken. Der Ablauf einer Anfrage an diesen Controller ist in Abbildung 6.9 dargestellt.

Es werden, wie beim Start der Anwendung, die persistent gespeicherten Switch-Zugangsdaten mithilfe der switchCredStore-Komponente neu ausgelesen. Dann werden die Switches mithilfe der switchComms-Komponente neu abgefragt. Anschließend wird der Inhalt des netStateCache vollständig geleert und mit den neuen Switch-Daten befüllt. Ist dieser Prozess beendet, wird dem Aufrufer eine Bestätigung gesendet. Der neue Inhalt des netStateCache wird nicht direkt an den Aufrufer zurückgesendet. Ist der Aufrufer daran interessiert, die neuen Daten zu erhalten, ist im Anschluss der *GetSwitchesController* aufzurufen. Diese Entscheidung wurde getroffen, da es sich beim Aufrufer des *ReloadBackendController* nicht unbedingt um ein Frontend, das an den neuen Daten interessiert ist, handeln muss. Denkbar ist auch, einen Service einzusetzen, der periodisch diesen Controller aufruft, um die Daten des netStateCache regelmäßig zu aktualisieren. Ein solcher Service ist im WebCNC-Prototypen jedoch nicht enthalten.

7 Entwurf und Implementierung des Frontends

Dieses Kapitel widmet sich der Präsentationsschicht des WebCNC. Zunächst wird die grundlegende Funktionsweise eines Web-Frontends erläutert. Anschließend werden die zur Realisierung des Prototypen verwendeten Technologien und Bibliotheken präsentiert. Außerdem werden die Komponenten des Frontends, die ansichtsbasierte Rendering-Logik sowie der Navigationsfluss durch die Anwendung betrachtet.

7.1 Funktionsweise einer Web-Anwendung

Eine Web-Anwendung ist eine Anwendung, deren Nutzerinterface im Browser des Nutzers ausgeführt wird. Die Basisfunktionalität eines Webbrowsers ist es, HTML- und CSS-Dokumente graphisch darzustellen, um dem Nutzer den Inhalt einer Webseite zu präsentieren. Außerdem ist ein Webbrowser in der Lage, HTTP-Anfragen an Server, die eine entsprechende API anbieten, zu stellen, und Antworten von diesen zu empfangen. Auf diese Weise werden die Ressourcen, die für die Darstellung einer Webseite benötigt werden, von Servern heruntergeladen.

Moderne Webbrowser unterstützen neben der Präsentation von HTML- und CSS-Dokumenten und der Durchführung von HTTP-Anfragen eine weitere, grundlegende Technologie. So beinhalten Webbrowser eine Laufzeitumgebung für JavaScript-Code, die es Webseiten ermöglicht, im Browser des Nutzers Code auszuführen. Dieser wird in der Regel vom Rest des Systems isoliert und kann nur auf den Kontext der Webseite, von welcher der JavaScript-Code stammt, zugreifen. Mit der Browser-Unterstützung für JavaScript ist es möglich, Webseiten zu entwickeln, die clientseitig Berechnungen ausführen und auf deren Basis dynamischen HTML-Code darstellen.

Damit wird auch die Umsetzung einer Webanwendung als Single Page Application (SPA) ermöglicht. SPAs sind Web-Anwendungen, die beim Abruf der Anwendung von einem Webserver die gesamte Anwendung herunterladen und während der Nutzung der Anwendung keine neue Webseite laden. Verschiedene Inhalte werden stattdessen durch bedingtes Rendering auf Basis des mitgelieferten JavaScript-Codes angezeigt. Dies steht im Gegensatz zu klassischen Web-Anwendungen, die nach jeder ausgeführten Aktion eine neue Webseite laden. SPAs bieten im Vergleich zu klassischen Webanwendungen eine angenehmere Nutzererfahrung, da eine flüssigere Benutzung der Web-Anwendung, die einem lokal installierten Programms ähnelt, erreicht wird. Insbesondere bei der Navigation durch die Anwendung muss bei einer SPA nicht bei jedem Ansichtswechsel darauf gewartet werden, dass eine neue Seite von einem Webserver geladen wird. Das Frontend des WebCNC-Prototypen wird aufgrund dieses Vorteils in der Nutzererfahrung als SPA entwickelt.

7.2 Verwendete Technologien, Bibliotheken und Werkzeuge

Zur Implementierung des Frontend-Prototypen wurde eine Reihe an Bibliotheken und Werkzeugen verwendet, welche die Entwicklung unterstützen. Diese werden im folgenden Abschnitt kurz dargestellt.

Als Programmiersprache wurde für das Frontend TypeScript [Mic] gewählt. TypeScript ist eine von Microsoft entwickelte Programmiersprache, die JavaScript um statische Typisierung ergänzt. Bei statisch typisierten Programmiersprachen werden Variablentypen explizit im Code angegeben oder vom Compiler inferiert, um Typfehler zur Laufzeit zu vermeiden. Damit soll die Codequalität im Vergleich zum dynamisch typisierten JavaScript verbessert werden. Die Arbeit "To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub" [BM22] von Justus Bogner und Manuel Merkel deutet darauf hin, dass die Codequalität von realen TypeScript-Projekten tatsächlich höher ist als die von JavaScript-Projekten. Wie bereits dargestellt sind moderne Browser in der Lage, JavaScript-Code auszuführen - eine Laufzeitumgebung, die TypeScript-Code ausführen kann, ist in gängigen Browsern jedoch nicht vorhanden. Um dennoch eine in Web-Browsern lauffähig Webanwendung zu erstellen, wird TypeScript-Code beim Bauen des Projektes in gleichwertigen JavaScript-Code transpiliert. Da die explizite statische Typisierung in TypeScript optional ist, ist jeder JavaScript-Code auch gültiger TypeScript-Code. Damit ist es einem TypeScript-Projekt möglich, Bibliotheken zu verwenden, die für JavaScript entwickelt wurden.

Als Basis-Bibliothek für das Frontend wurde React [Met] gewählt. React ist eine User Interface (UI)-Bibliothek für Web-Anwendungen, die eine komponentenbasierte Architektur unterstützt und ein reichhaltiges Ökosystem aus ergänzenden Bibliotheken besitzt. React ist außerdem gut für die Umsetzung von SPAs geeignet. Da React im Gegensatz zu Konkurrenten wie Angular [Gooa] oder VueJS [Evab] kein vollwertiges Framework ist, werden für viele gängige Funktionalitäten weitere Bibliotheken benötigt. So wird als HTTP-Client zum Stellen von Anfragen an das Backend Axios [ZUJ+] verwendet. Zur Darstellung der Netzwerktopologie als Graph wird die react-force-graph-Bibliothek verwendet [SAO+24], die Graphen auf Basis einer Kräftesimulation an den Kanten erzeugt. Zum Styling der Webseite wird Bootstrap [OTX+] genutzt. Bootstrap ist eine weit verbreitete Styling-Bibliothek, die es erleichtert, Styling-Konstruktionen wie ein Rastersystem für React-Komponenten umzusetzen. Als Build Tool für das Projekt wird Vite [Evaa] eingesetzt. Vite liefert auch einen eingebauten Development Server mit, der es ermöglicht, die in Entwicklung befindliche Webseite lokal zu hosten, ohne das komplette Projekt bauen und über einen separaten Webserver hosten zu müssen. Dies vereinfacht den Prozess, Änderungen an der Webseite vorzunehmen und diese direkt zu testen, ohne einen aufwändigen Setup-Prozess durchlaufen zu müssen.

7.3 Datenstrukturen

Als statisch typisierte Sprache unterstützt TypeScript explizite Typdefinitionen. Die im Frontend genutzten Typen sind in Abbildung 7.1 dargestellt. Sie sind stark an die Datenstrukturen des Backends, die in Abbildung 6.1 dargestellt sind, angelehnt. Dies ist insbesondere bei Bezeichnern notwendig, um die Konvertierung der Typen vom JSON-Format in die interne Repräsentation und umgekehrt

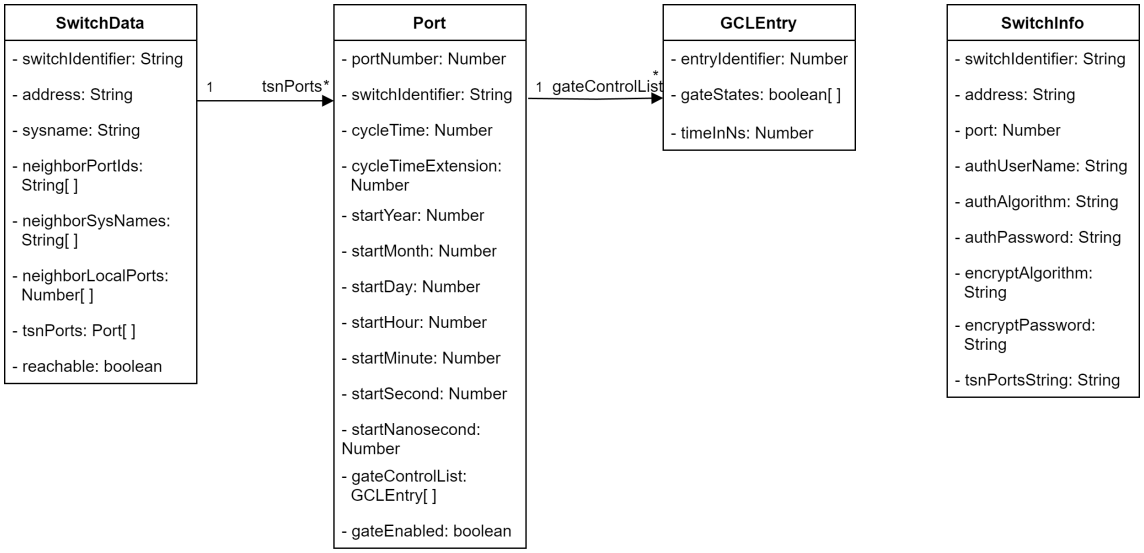

Abbildung 7.1: Typdefinitionen im Frontend

zu erleichtern. Dies geschieht, wenn *Port*-, *Switch*- oder *SwitchInfo*-Objekte durch Aufrufen der in Kapitel 6.7 vorgestellten REST-Controller zwischen Frontend und Backend ausgetauscht werden. Im Frontend übernimmt die Axios-Bibliothek, die den HTTP-Client bereitstellt, diese Aufgabe.

Neben der Anpassung der Java-Integer zu TypeScript-Numbers und der Verwendung von Arrays statt Listen sind die größten Unterschiede im *SwitchData*-Typ zu finden. Da dem Frontend die Konfigurationsprotokoll-Zugangsdaten eines Switches nicht vom Backend mitgeteilt werden, sind die entsprechenden Felder aus der *Switch*-Klasse des Backends im Frontend nicht vorhanden. Um diesen Unterschied deutlich zu kennzeichnen, wird der der *Switch*-Klasse entsprechende Typ im Frontend *SwitchData* genannt. Der *SwitchInfo*-Typ hingegen ist bis auf programmiersprachenbedingte Unterschiede gleich der entsprechenden Klasse im Backend und enthält die vollen SNMP-Zugangsdaten. Dies ist unproblematisch, da das Backend seine *SwitchInfo*-Objekte nie dem Frontend zur Verfügung stellt. So werden *SwitchInfo*-Objekte nur beim Anlegen eines neuen Switches vom Frontend ans Backend gesendet.

7.4 Ansichten und Navigationsfluss

Die Präsentation der Komponenten auf der graphischen Oberfläche ist in fünf verschiedene Ansichten unterteilt. Diese Ansichten können aus einer oder mehreren Hauptkomponenten sowie einer Reihe weiterer Navigationskomponenten bestehen. Das Zustandsdiagramm in Abbildung 7.2 stellt die verfügbaren Ansichten, die in den Ansichten gerenderten Komponenten sowie die Navigationsmöglichkeiten zwischen den Ansichten dar. Durch eine Reihe von Variablen wird gesteuert, welche der Ansichten aktuell gerendert wird. Diese Ansichten werden im folgenden Abschnitt diskutiert.

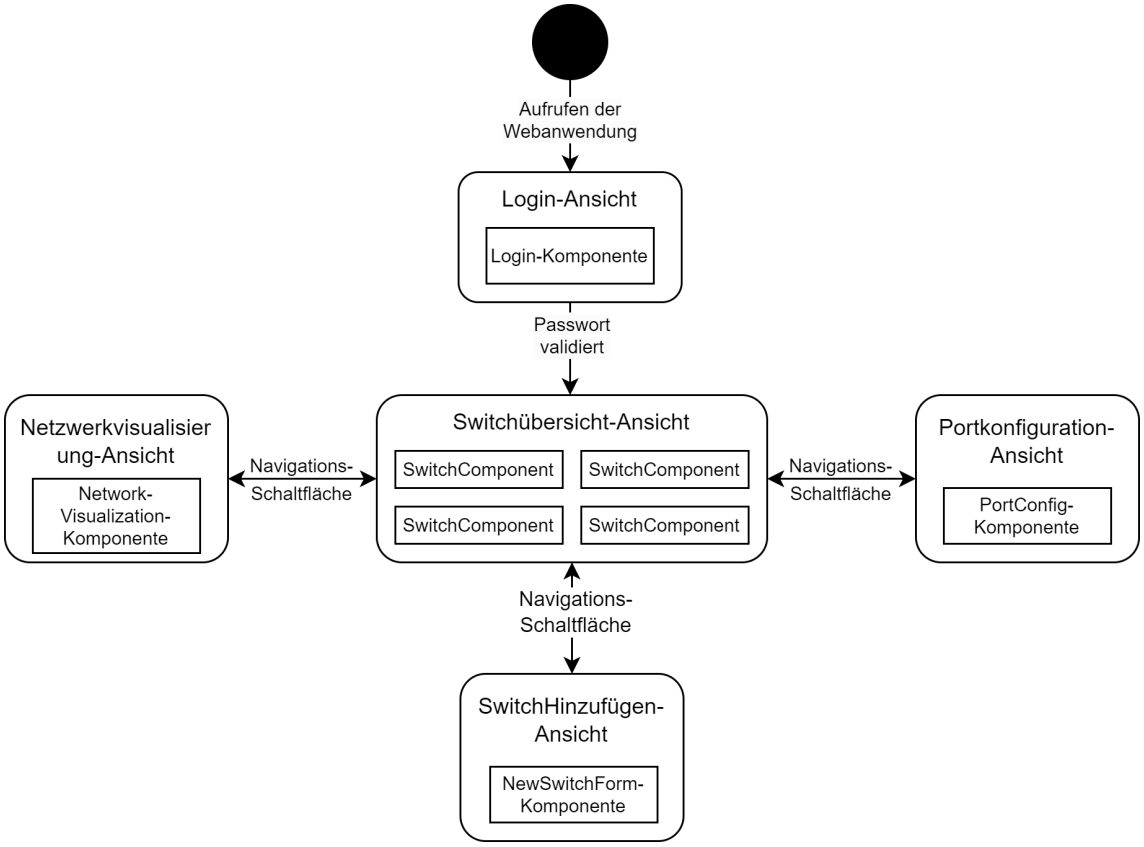

Abbildung 7.2: Zustandsdiagramm der Frontend-Ansichten

7.4.1 Login-Ansicht

Beim Aufrufen der Anwendung ist zunächst die Login-Ansicht aktiv, die es mit dem Rendering einer *Login*-Komponente ermöglicht, das Passwort einzugeben. Dies wird erreicht, indem die *isAuthenticated*-Flag, die das Rendering der Login-Ansicht steuert, zum Start der Anwendung den Wert *false* annimmt.

Die *Login*-Komponente präsentiert dem Nutzer ein Eingabefeld für ein Passwort und eine Schaltfläche zum Absenden des Passwortes. Diese Komponente wird zur Zugriffskontrolle eingesetzt, damit nur dazu berechtigte Nutzer den WebCNC nutzen können.

Wird vom Nutzer ein Passwort eingegeben und abgesendet, wird der in Kapitel 6.7 beschriebene *ValidatePasswordController* des Backends angesprochen, um das eingegebene Passwort zu validieren. Liefert das Backend eine Bestätigung zurück, wird der *GetSwitchesController* kontaktiert, um die vom Backend vorgehaltenen Daten über den Netzwerkzustand abzurufen. Diese werden lokal in der Webanwendung zwischengespeichert. Ist dies erfolgt, wird die *isAuthenticated*-Flag auf *true* gesetzt, um zur *Switchübersicht*-Ansicht zu wechseln. Außerdem wird das eingegebene Passwort vorgehalten, um es zur Authentifikation beim Ansprechen weiterer RESTController verwenden zu können.

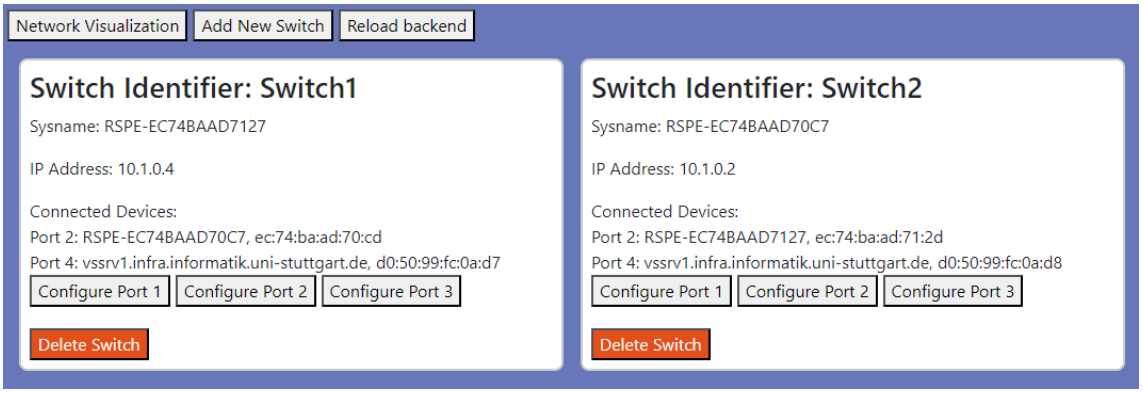

Abbildung 7.3: Switchübersicht-Ansicht

7.4.2 Switchübersicht-Ansicht

Die *Switchübersicht*-Ansicht wird gerendert, wenn *isAuthenticated* den Wert *true* annimmt und keine weitere Ansichtensteuerungsvariable gesetzt ist. Die Ansicht ist in Abbildung 7.3 dargestellt. Ihr Zweck ist es, dem Nutzer einen Überblick über die bekannten Switches zu verschaffen und alle relevanten, für den Switch global gültigen Informationen darzustellen. Dafür wird für jeden bekannten Switch eine *SwitchComponent* gerendert.

Die *SwitchComponent*-Komponente wird dazu eingesetzt, generelle Informationen über einen Switch darzustellen. So werden der vergebene SwitchIdentifier, der Systemname und die IP-Adresse des Switches präsentiert. Außerdem werden die LLDP-Informationen über benachbarte Geräte tabellarisch dargestellt. Jeder Eintrag enthält die Portnummer des Switches, an dem das Gerät angeschlossen ist, den Systemnamen des Nachbargerätes sowie die PortID des Nachbargerätes. Die mittels LLDP erfasste PortID ist üblicherweise die MAC-Adresse des Netzwerkinterfaces des Nachbargeräts. Es kann sich jedoch auch um eine andere, eindeutige Identifikation des Netzwerkadapters handeln. Neben der Darstellung dieser Informationen sind auch Schaltflächen vorhanden, welche die TSN-fähigen Ports des Switches repräsentieren. Mit ihnen kann zu den TAS-Konfigurationen der Ports gewechselt werden. Zuletzt gibt es noch eine Schaltfläche zum permanenten Entfernen des Switches aus dem WebCNC. Wird diese betätigt und das Entfernen vom Nutzer in einem Kontextfenster bestätigt, wird der *RemoveSwitchController* des Backends mit dem *switchIdentifier* des betroffenen Switches kontaktiert, um den Switch permanent zu entfernen.

Ist die *reachable*-Flag eines Switches auf false gesetzt, wird die zugehörige *SwitchComponent* farblich hervorgehoben, um den Nutzer auf ein mögliches Problem aufmerksam zu machen. Außerdem sind in diesem Fall die Informationen über Nachbargeräte sowie die Schaltflächen für TSN-Ports nicht verfügbar.

Außerdem steht in der *Switchübersicht* eine Schaltfläche zum vollständigen Aktualisieren des Backends bereit. Wird diese Schaltfläche betätigt, wird der *ReloadBackendController* des Backends angesprochen, um eine vollständige Aktualisierung des im Backend gecacheten Netzwerkzustands zu bewirken. Da das Frontend im Anschluss an diesen Prozess auch seine lokal zwischengespeicherten Switch-Daten mit den neuen Daten des Backends aktualisieren soll, wird erneut der *GetSwitchesController* aufgerufen.

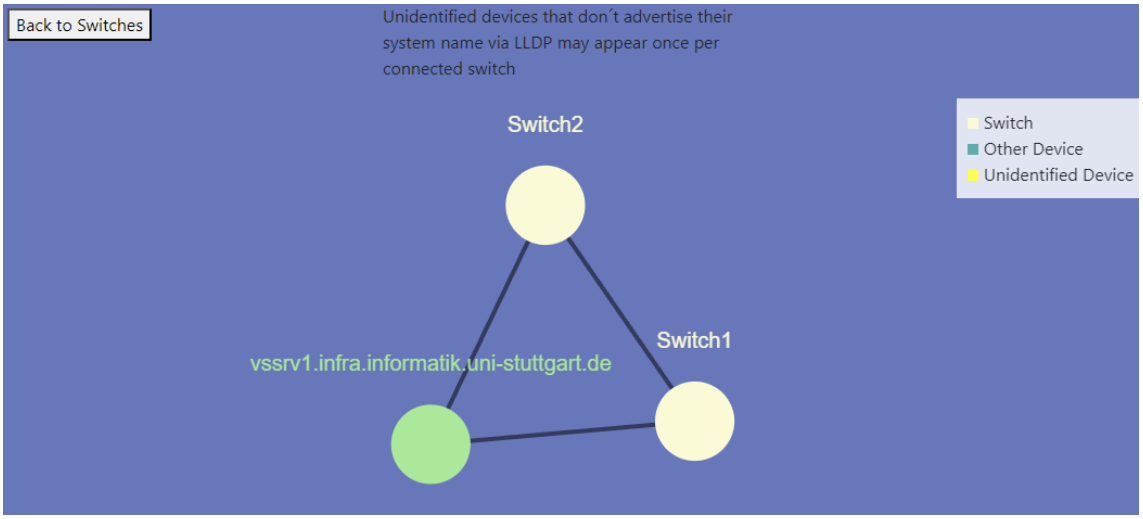

Abbildung 7.4: Netzwerkvisualisierung-Ansicht

Die *Switchübersicht*-Ansicht dient außerdem als zentraler Punkt für die Navigation zu anderen Ansichten. So stehen Schaltflächen bereit, um zu den *Netzwerkvisualisierung*- oder *SwitchHinzufügen*-Ansichten zu wechseln. Die in den gerenderten *SwitchComponents* vorhandenen Schaltflächen, welche die TSN-fähigen Ports der Switches repräsentieren, erlauben außerdem eine Navigation zur *Portkonfiguration*-Ansicht des jeweiligen TSN-Ports.

7.4.3 Netzwerkvisualisierung-Ansicht

Die Netzwerkvisualisierung-Ansicht beinhaltet eine *NetworkVisualization*-Komponente zur graphischen Darstellung der Netzwerktopologie. Die Ansicht ist in Abbildung 7.4 dargestellt.

Die *NetworkVisualization*-Komponente wird genutzt, um dem Nutzer eine graphische Repräsentation der Netzwerktopologie zu präsentieren. Die angedachten Hauptnutzungszwecke dieser Komponente sind es, sich einen ersten Überblick über die Netzwerkstruktur zu verschaffen und ein bekanntes Netzwerk auf Vollständigkeit zu prüfen. Daher wird der Netzwerkgraph nicht auf Basis von Netzwerkinterfaces, sondern auf Basis von Geräten generiert. Maßgeblich für die Identifikation eines Gerätes ist der mittels LLDP erfasste Systemname. Damit werden zum Zweck der vereinfachten Übersicht im Graph keine genauen Informationen über die Verbindungen zwischen einzelnen Netzwerkswitch-Ports und den Netzwerkinterfaces der Nachbargeräte sichtbar. Hat sich ein Nutzer mithilfe des Graphen einen ersten Überblick über die Netzwerkstruktur verschafft, können für nähere Informationen zu einzelnen Verbindungen – wie man sie beispielsweise zur Konfiguration des TAS benötigt – die tabellarischen Informationen der *SwitchComponent*-Komponenten zu Rate gezogen werden.

Diese Ansicht ist aktiv, wenn die *ShowVis*-Flag den Wert *true* annimmt. Um zur *Switchübersicht* zurückzukehren, ist außerdem eine Navigationsschaltfläche vorhanden, welche die *ShowVis*-Flag auf *false* setzt.

Abbildung 7.5: Portkonfiguration-Ansicht

7.4.4 Portkonfiguration-Ansicht

Zur Einsicht und Modifikation der TSN-Konfiguration eines Ports wird die *Portkonfiguration*-Ansicht genutzt. Die Ansicht ist in Abbildung 7.5 dargestellt. Dafür wird in dieser Ansicht eine *PortConfig*-Komponente gerendert.

Die *PortConfig*-Komponente stellt die TAS-Konfiguration eines bestimmten Ports an einem Netzwerkswitch dar. Hier ist es möglich, die *CycleTime*, *CycleTimeExtension* und *gateEnabled*-Parameter einzusehen und zu modifizieren. Außerdem lässt sich der Aktivierungszeitpunkt der Konfiguration sowie die Gate Control List (GCL) anpassen. Die GCL kann eintragsweise manipuliert werden. So können GCL-Einträge entfernt und gelöscht werden. Außerdem kann die Zeitdauer des Eintrags sowie die Zustände der Gates, während der Eintrag aktiv ist, konfiguriert werden.

Der Aktivitätszustand dieser Ansicht kann nicht durch eine einfache, boolesche Flag gesteuert werden, da die Daten, auf denen die *PortConfig*-Komponente arbeitet, von dem ausgewähltem Port abhängen. So ist die Ansicht aktiv, wenn der Wert der *currentPort*-Variable auf ein bestimmtes Port-Objekt gesetzt wird. Inaktiv ist die Ansicht, wenn diese Variable den *null*-Wert annimmt.

Beim Abspeichern einer neuen Portkonfiguration wird dem *SavePortController* des Backends die neue Konfiguration in Form eines *Port*-Objektes übermittelt. Wird eine Bestätigungsantwort empfangen, werden auch der im Frontend lokal zwischengespeicherte Switch mit dem neuen Port

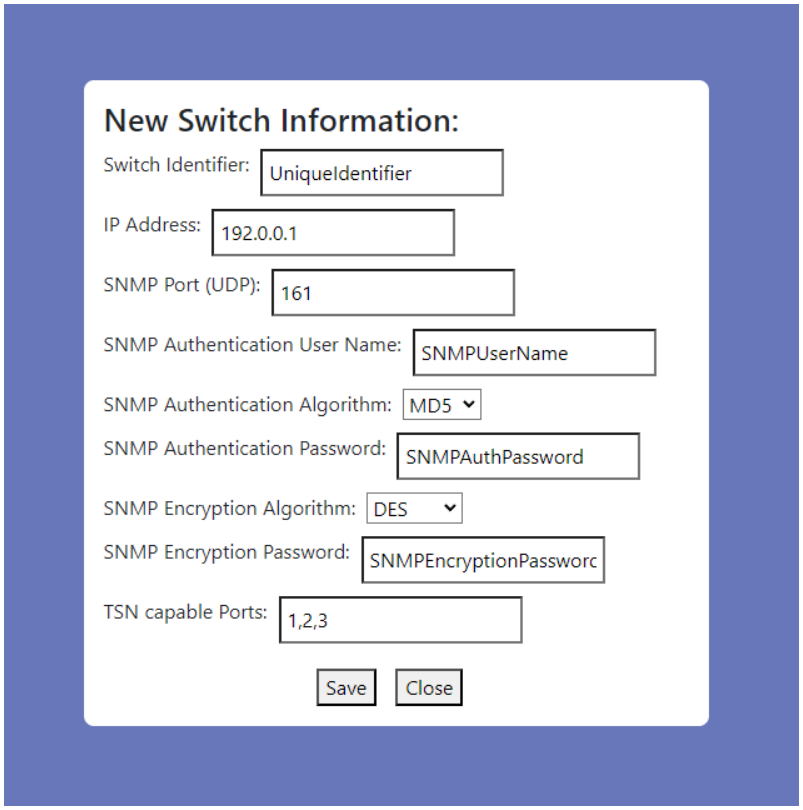

Abbildung 7.6: SwitchHinzufügen-Ansicht

upgedated. Anschließend wird *currentPort* auf null gesetzt, um zur *Switchübersicht* zurückzukehren. Alternativ gibt es auch eine Schaltfläche, die alle getroffenen Änderungen an der TSN-Konfiguration des Ports verwirft und ebenfalls zur *Switchübersicht* zurückkehrt.

7.4.5 SwitchHinzufügen-Ansicht

Als letzte verfügbare Ansicht bietet die *SwitchHinzufügen*-Ansicht die Möglichkeit, einen neuen Switch zum WebCNC hinzuzufügen. Die Ansicht ist in Abbildung 7.6 dargestellt. Dafür wird eine *NewSwitchForm*-Komponente verwendet.

Die *NewSwitchForm*-Komponente stellt eine formularartige Komponente dar, die genutzt wird, um die Zugangsdaten eines neu anzulegenden Switches zu erfassen. So wird für jedes Attribut des *SwitchInfo*-Typs ein Eingabefeld gerendert. Dies ermöglicht es Nutzern, über die graphische Oberfläche des Frontends einen neuen Switch zu registrieren.

Um dem Nutzer die Eingabe gültiger Werte im richtigen Format zu erleichtern, sind die Eingabefelder mit Demo-Werten vorausgefüllt, welche die erwartete Formatierung der Eingaben demonstrieren. Aktiv ist diese Ansicht, wenn die *showNewSwitchForm*-Flag auf *true* gesetzt ist.

Hat der Nutzer die Daten eines neuen Switches eingegeben, werden diese im Frontend zunächst auf Korrektheit überprüft. So muss eine IP-Adresse beispielsweise dem Format einer gültigen Adresse entsprechen. Ist diese Überprüfung erfolgreich, wird der *AddSwitchController* des Backends mit den eingegebenen Daten im Form eines *SwitchInfo*-Objektes angesprochen, um den Switch hinzuzufügen. Hat das Backend den aktuellen Zustand des Switches erfolgreich einholen können, erhält das Frontend das neue *Switch*-Objekt als Antwort zurück und aktualisiert den lokal im Browser zwischengespeicherten Netzwerkzustand. Anschließend wird die *showNewSwitchForm*-Flag auf *false* gesetzt, um zur *Switchübersicht* zurückzukehren. Alternativ gibt es eine Schaltfläche, die alle eingegebenen Daten verwirft und zur *Switchübersicht* zurückkehrt.

7.4.6 Spinner-Overlay

Die *Spinner*-Komponente ist eine Hilfskomponente, die genutzt wird, um andauernde Kommunikationsprozesse mit dem Backend für den Nutzer graphisch darzustellen. Es handelt sich um einen sich drehenden Lade-Halbkreis, der auf einem semitransparenten Hintergrund platziert ist. Die *Spinner*-Komponente kann während eines Kommunikationsprozesses mit dem Backend über den zuvor präsentierten Ansichten platziert werden, um Interaktionen des Nutzers mit der Anwendung während des Kommunikationsprozesses zu verhindern. Außerdem werden Nutzer so über einen laufenden Kommunikationsvorgang in Kenntnis gesetzt. Dass die eigentliche Anwendungsoberfläche mit der zuvor aktiven Ansicht unter dem semitransparenten Hintergrund des Spinners weiterhin zu erkennen ist, soll dem Nutzer das Gefühl einer flüssigeren Benutzeroberfläche vermitteln.

8 Validierung

In diesem Kapitel wird untersucht, ob die in Kapitel 4.2 definierten Ziele erreicht wurden. Dafür wird zunächst die Funktionalität des WebCNC in Bezug auf die Darstellung der Netzwerktopologie sowie die TSN-Konfiguration der Netzwerkswitches validiert. Anschließend wird diskutiert, ob die Architekturziele der Benutzerfreundlichkeit, Portabilität und Erweiterbarkeit erreicht wurden.

8.1 Funktionstests

Im diesem Abschnitt wird untersucht, ob die Darstellung der Netzwerktopologie sowie die Anzeige und Änderung der TSN-Konfiguration in einem Testaufbau korrekt funktioniert. Eine korrekte Funktion des WebCNC wird als gegeben definiert, wenn die angezeigten und konfigurierten Werte mit den tatsächlich auf den Switches vorhandenen Werten übereinstimmen. Bei den verwendeten Switches handelt es sich um RSP-Switches von Hirschmann, die pro Switch eine eigene Weboberfläche zur Konfiguration bereitstellen. Da dieses Webinterface unter anderem Informationen über die gesammelten LLDP-Daten sowie die TAS-Konfiguration der Ports beinhaltet, werden die Daten der Webinterfaces als Autorität für tatsächlich auf den Switches vorhandene Werte genutzt. Um die Lesbarkeit mancher der verwendeten Screenshots zu verbessern, wurden mittels Bildbearbeitung für den jeweiligen Test irrelevante Bildteile ausgeschnitten und entfernt.

8.1.1 Versuchsaufbau

Der Versuchsaufbau zur Validierung der Funktionalität ist in Abbildung 8.1 dargestellt. Es kommen zwei Switches sowie ein Endgerät, auf dem der WebCNC-Prototyp läuft, zum Einsatz. Die Netzwerkinterfaces des Endgeräts sind jeweils mit Port vier der beiden Netzwerkswitches verbunden. Außerdem sind die Switches untereinander an Port zwei der jeweiligen Switches verbunden. TSN wird von den Switches an den Ports eins bis drei unterstützt.

8.1.2 Funktionstest 1: Darstellung der Netzwerktopologie

Dieser Funktionstest wird genutzt, um die Darstellung der Netzwerktopologie im WebCNC zu überprüfen.

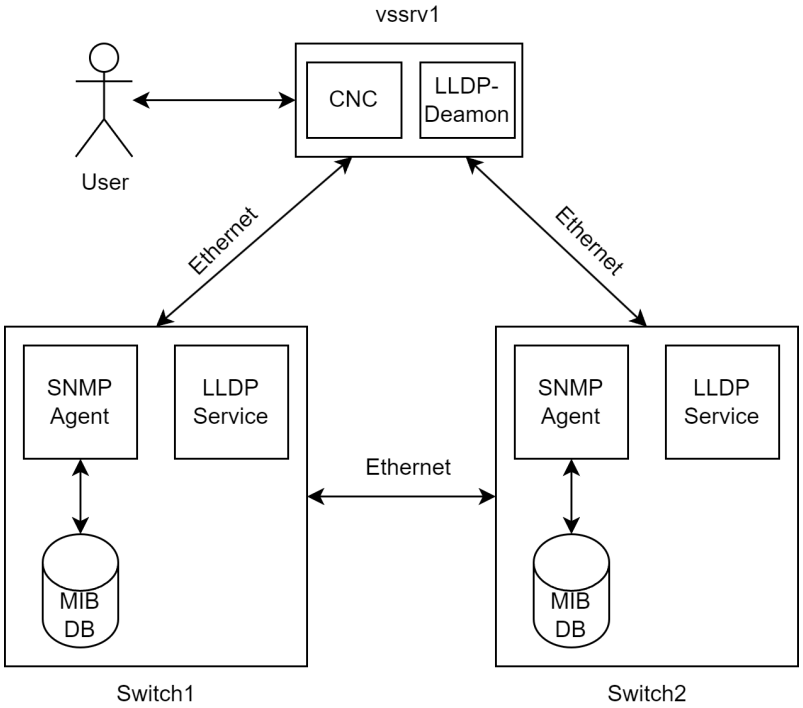

Abbildung 8.1: Versuchsaufbau

Szenario

In diesem Szenario wird überprüft, ob die Netzwerktopologie korrekt in der GUI des WebCNC angezeigt wird. Hier müssen die Netzwerkvisualisierung-Ansicht mit ihren eingeschränkten, graphisch dargestellten Informationen sowie die in der Switchübersicht tabellarisch präsentierten, umfangreicheren Informationen überprüft werden.

Die erwarteten Ergebnisse ergeben sich direkt aus dem Versuchsaufbau. Da die Netzwerkvisualisierung-Ansicht, wie in Kapitel 7.4 beschrieben, Geräte und deren Verbindungen untereinander ohne detaillierte Informationen darstellt, ist zu erwarten, dass zwei Switches und ein weiteres Gerät in der Netzwerkvisualisierung angezeigt werden, die alle miteinander verbunden sind. In den weitergehenden Informationen über Nachbargeräte, die in den Switch-Komponenten der Switchübersicht angezeigt werden, sind außerdem Informationen über die Systemnamen, Port IDs – hier gleich der MAC-Adresse des Netzwerkinterfaces – sowie der lokalen Portnummern der Switches, an denen das Nachbargerät angeschlossen ist, erwartet. Diese sollen mit den in den Weboberflächen der Switches angezeigten Informationen übereinstimmen.

Testergebnis

Im ersten Schritt wird die Validierung der Netzwerktopologie-Informationen durchgeführt. Abbildung 8.2 zeigt die Networkvisualization-Ansicht des WebCNC. Wie erwartet sind zwei Switches und ein weiteres Gerät vorhanden, die alle paarweise miteinander verbunden sind. Abbildung 8.3 zeigt die

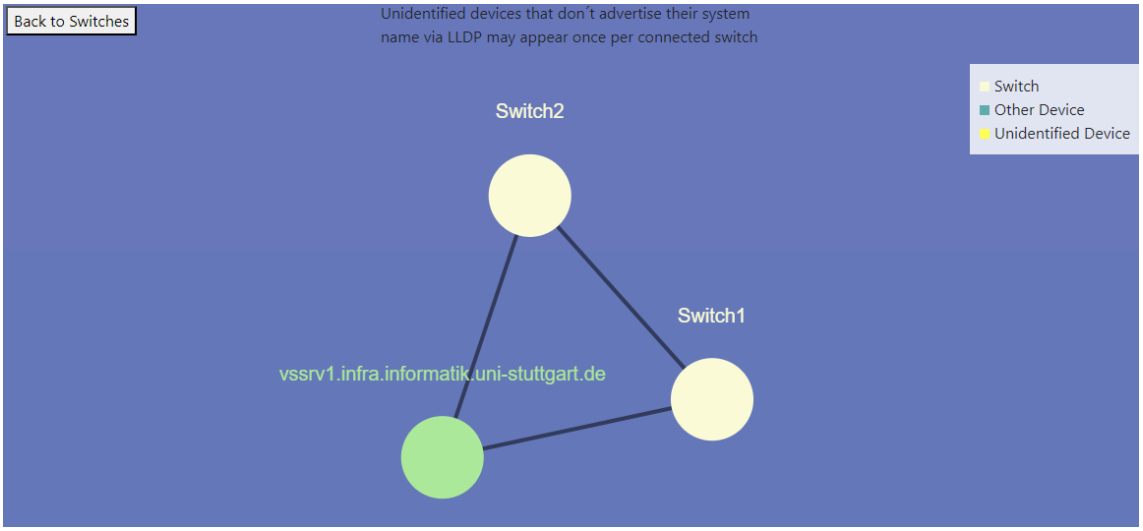

Abbildung 8.2: Netzwerkvisualisierung des WebCNC

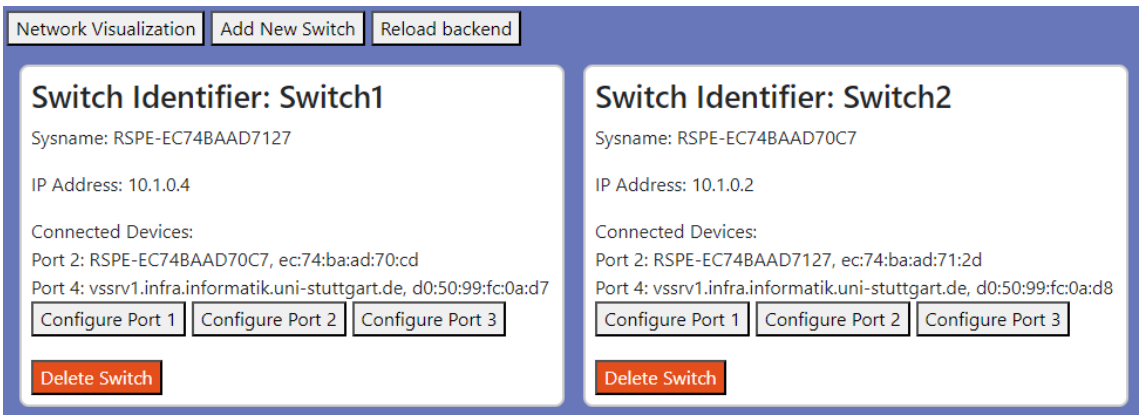

Abbildung 8.3: Switchübersicht des WebCNC

Switchübersicht-Ansicht mit den erweiterten Informationen über Nachbargeräte der Switches. Diese Informationen decken sich mit den in Abbildung 8.4 und Abbildung 8.5 dargestellten Informationen aus den Webinterfaces der Switches. Damit ist die Validierung der Netzwerktopologie-Darstellung erfolgreich abgeschlossen.

8.1.3 Funktionstest 2: Auslesen und Darstellung der IEEE 802.1Qbv-Schedules

In diesem Funktionstest wird überprüft, ob der WebCNC die aktuellen Zustände der Netzwerkswitches beim WebCNC-Start oder Ausführen eines Reloads korrekt ausliest und dem Nutzer präsentiert.

Port	Nachbar-Systemname	Port-ID
1/4	vssrv1.infra.informatik.uni-s	D0:50:99:FC:0A:D7
1/2	RSPE-EC74BAAD70C7	EC:74:BA:AD:70:CD

Abbildung 8.4: LLDP-Informationen von Switch1

Port	Nachbar-Systemname	Port ID
1/2	RSPE-EC74BAAD7127	EC:74:BA:AD:71:2D
1/4	vssrv1.infra.informatik.uni-stuttgart.de	D0:50:99:FC:0A:D8

Abbildung 8.5: LLDP-Informationen von Switch2

Szenario

In diesem Szenario wird zunächst über die Webinterfaces der Switches eine TSN-Konfiguration vorgenommen, die beim anschließenden Start des WebCNC korrekt angezeigt werden soll. Konkret wird Port zwei von Switch1 auf eine Zykluszeit von 100 000 ns mit Startzeitpunkt 1. Januar 1970, 00:00:00 Uhr und 0 Nanosekunden konfiguriert. Die GCL wird mit zwei Einträgen konfiguriert, wobei der erste 80 000 ns aktiv ist und alle Gates öffnet. Der zweite Einträg ist für 20 000 ns aktiv und öffnet ausschließlich Gate 7. Dies soll entsprechend in dem WebCNC angezeigt werden.

Testergebnis

Port zwei von Switch1 wird zunächst wie beschrieben über das Switch-Webinterface mit den im Szenario festgelegten Werten belegt. Dies ist in Abbildung 8.6 sowie Abbildung 8.7 zu sehen. Nach einem anschließenden Start oder Reload des WebCNC werden diese Werte wie erwartet in der Portkonfigurations-Ansicht von Port zwei, Switch1 angezeigt. Dies ist in Abbildung 8.8 dargestellt. Damit ist dieser Test erfolgreich abgeschlossen.

	Zyklus-Zeit [ns]		Basiszeit			
Port	Konfiguriert	Aktuell	Datum	Zeit	[ns]	Aktuell
1/2	100.000	100.000	Donnerstag, 1. Januar 1970	12:00 AM	0	01.01.1970, 00:00:00 0 ns

Abbildung 8.6: Port-Konfiguration des Switch-Webinterfaces

1/1 1/2 1/3

Index	Gate-Zustände	Intervall [ns]
1	7, 6, 5, 4, 3, 2, 1, 0	80.000
2	7	20.000

Abbildung 8.7: GCL-Konfiguration des Switch-Webinterfaces

Configuring Port 2 of Switch1:

TSN enabled on port: ☑

Cycle Time: 100000

Cycle Time Extension: 0

Start Year:	Start Month:	Start Day:	Start Hour:	Start Minute:	Start Second:	Start Nanosecond:
1970	1	1	0	0	0	0

Gate Control List:

GCL Entry 1

Gate 7: ☑ Gate 6: ☑ Gate 5: ☑ Gate 4: ☑ Gate 3: ☑ Gate 2: ☑ Gate 1: ☑ Gate 0: ☑

Time in ns: 80000

Remove Entry

GCL Entry 2

Gate 7: ☑ Gate 6: ☐ Gate 5: ☐ Gate 4: ☐ Gate 3: ☐ Gate 2: ☐ Gate 1: ☐ Gate 0: ☐

Time in ns: 20000

Remove Entry

Add GCL Entry Save Close

Abbildung 8.8: TSN-Konfiguration des Ports im WebCNC

8.1.4 Funktionstest 3: Konfiguration der IEEE 802.1Qbv-Schedules

In diesem Funktionstest wird geprüft, dass über den WebCNC-GUI vorgenommene Änderungen der TSN-Konfiguration korrekt auf den Switches eingestellt werden.

Szenario

In diesem Szenario wird die aus Funktionstest 2 vorhandene Konfiguration über die Weboberfläche des WebCNC verändert. Anschließend wird mithilfe des Switch-Webinterfaces abgeglichen, ob die vorgenommene Konfiguration korrekt übernommen wurde. Konkret wird Port zwei von Switch eins mit einer Zykluszeit von 150 000 ns und einem Startzeitpunkt vom 20. September 2005 um 17:34:18 Uhr und 30 000 Nanosekunden konfiguriert. Der erste GCL-Eintrag aus Funktionstest 2 wird unverändert übernommen. Der zweite Eintrag wird so verändert, dass nun alle Gates geschlossen sind. Es wird ein dritter Eintrag hinzugefügt, der 50 000 ns aktiv ist und die Gates 4 und 5 öffnet. Wird diese Änderung gemäß des Webinterface des Switches korrekt auf dem Switch übernommen, gilt der Test als bestanden

Testergebnis

Der Test beginnt mit einer Anpassung der Port-Konfiguration durch die Web-Oberfläche des WebCNC entsprechend dem definierten Szenario. Dies ist in Abbildung 8.9 dargestellt. Anschließend wird in den Web-Interfaces des Netzwerkswitches geprüft, ob diese neue Konfiguration erfolgreich und korrekt übernommen wurde. Dies ist in Abbildung 8.10 und Abbildung 8.11 zu sehen. Da die Werte wie erwartet übereinstimmen, wurde dieser Test ebenfalls erfolgreich abgeschlossen. Damit ist die grundlegende Funktionalität des WebCNC validiert.

8.2 Architekturziele

Nachdem die Funktionalität des WebCNC-Prototypen validiert wurde, wird im Folgenden diskutiert, ob und wie die in Kapitel 4.2 aufgestellten, nicht-funktionalen Anforderungen erfüllt sind. So wird die Portabilität des Gesamtpakets, bestehend aus Frontend und Backend, betrachtet. Außerdem wird der modulare, komponentenbasierte Aufbau und die damit einhergehende Erweiterbarkeit analysiert. Auch die Anforderung der Benutzerfreundlichkeit wird kurz betrachtet.

8.2.1 Portabilität

Eine der Anforderungen an den WebCNC ist die Portabilität. So soll es möglich sein, den WebCNC mit möglichst geringen Infrastrukturanforderungen und Einschränkungen auf verschiedensten Hostsystemen zu betreiben.

Das in Kapitel 6 vorgestellte Backend kann auf jedem System, für das ein Java Runtime Environment (JRE) existiert, ausgeführt werden. Da der WebCNC-Prototyp mit der netStateCache-Komponente den Netzwerkzustand-Cache im Hauptspeicher vorhält und die persistenten Switch-Zugangsdaten-Konfigurationen mit der switchCredStore-Komponente in einer einfachen Textdatei abgespeichert

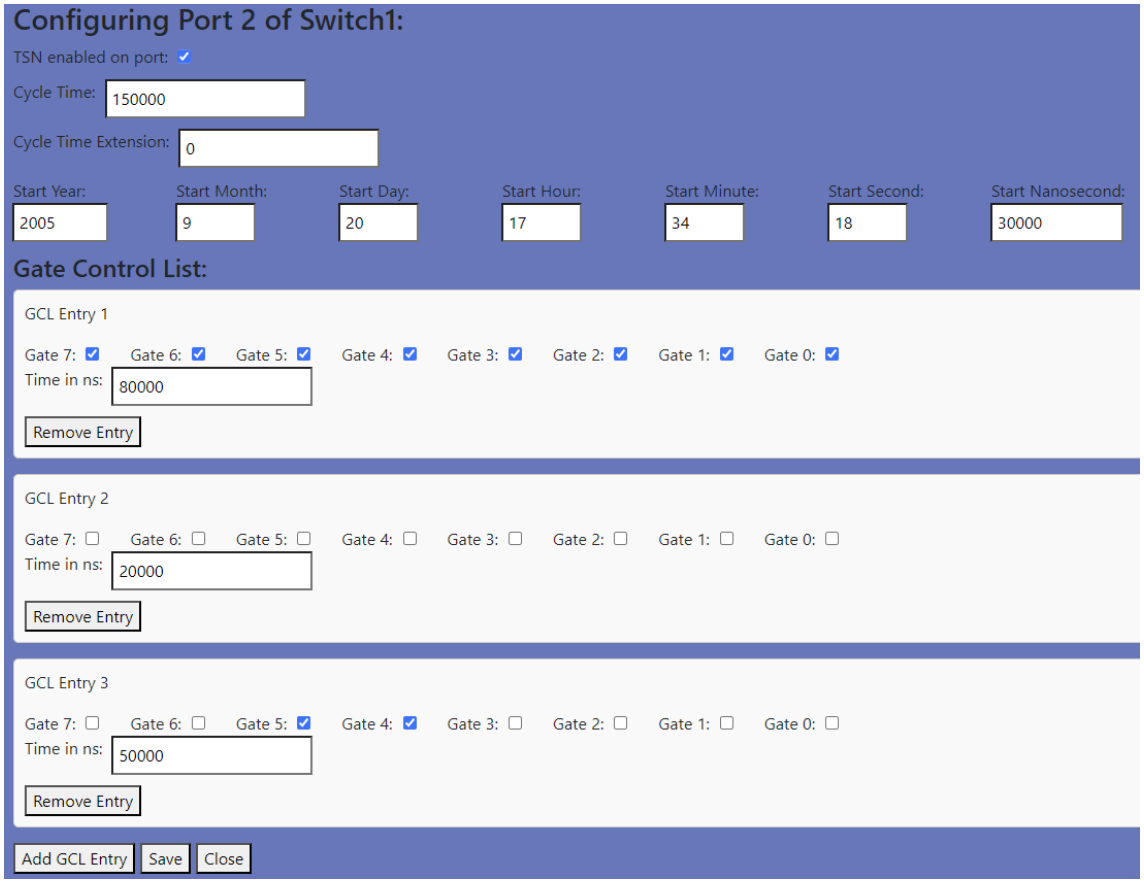

Abbildung 8.9: TSN-Konfiguration des Ports im WebCNC nach Konfigurationsänderung

		Zyklus-Zeit [ns]				Basiszeit		
	Port	Konfiguriert	Aktuell	Datum	Zeit	[ns]	Aktuell
	1/2	150.000	150.000	Dienstag, 20. September 2005	05:34 PM	30000	20.09.2005, 17:34:18 30000 ns

Abbildung 8.10: Port-Konfiguration des Switch-Webinterfaces nach Konfigurationsänderung

werden, liegen auch hier keine weiteren Anforderungen an externe Infrastruktur vor. Damit kann das Backend auf allen gängigen Host-Systemen mit verschiedensten Betriebssystemen und CPU-Befehlssätzen ausgeführt werden.

Das Frontend ist ähnlich portabel wie das Backend. Das Frontend besteht lediglich aus einem Ordner mit statischen Dateien, die über einen Webserver auf Anfrage an Clients ausgeliefert werden müssen, um lokal im Browser der Clients ausgeführt zu werden. Hier gibt es keine speziellen Vorgaben, welche Webserver verwendet werden müssen. So kann jeder Webserver, der HTTPS-Kommunikation unterstützt, genutzt werden. Für die Validierung des Prototyps wurde nginx [Sys] verwendet, dies ist jedoch in keiner Weise durch das Frontend vorgegeben. Da es geeignete Webserver für jedes gängige System gibt, ist auch das Frontend als sehr portabel einzustufen. Damit ist die Portabilität des WebCNC insgesamt auf einem hohen Niveau gegeben.

1/1 1/2 1/3

Index	Gate-Zustände	Intervall [ns]
1	7, 6, 5, 4, 3, 2, 1, 0	80.000
2	-	20.000
3	5, 4	50.000

Abbildung 8.11: GCL-Konfiguration des Switch-Webinterfaces nach Konfigurationsänderung

8.2.2 Modularität

Ein Kernaspekt der WebCNC-Architektur ist die Modularität des WebCNC. Diese soll die Wartbarkeit und insbesondere die Erweiterbarkeit des WebCNC gewährleisten. Dies betrifft sowohl das Frontend als auch das Backend. Im Folgenden werden zwei verschiedene Arten der Erweiterbarkeit unterschieden. Zum einen soll Erweiterbarkeit im Sinne des Hinzufügens neuer Komponenten mit möglichst geringen Änderungen an dem bestehenden System gewährleistet werden. Außerdem wird Erweiterbarkeit im Sinne der Erweiterung des WebCNC auf weitere Southbound-Protokolle betrachtet.

Erweiterung um neue Komponenten

Das Hinzufügen neuer Komponenten ist ein Kernaspekt der Erweiterbarkeit des WebCNC. Dies lässt sich in drei Subaspekte weiter aufteilen. So unterscheidet sich die Erweiterung um Backend-Komponenten mit Visualisierung im Frontend von Backend-Komponenten ohne eine Visualisierung. Außerdem ist das Einführen einer neuen Visualisierung, die nur auf den bereits bestehenden Daten und Funktionen operiert, und keine Änderung am Backend erfordert, möglich.

Komponenten ohne Visualisierung

Das Hinzufügen neuer Backend-Komponenten ohne Visualisierung im Frontend ist die einfachste Art der Erweiterung. Grundsätzlich sind neue Komponenten unabhängig von allen anderen Komponenten. Insbesondere sind Komponenten selbst dafür verantwortlich, ihre komponenteninternen Daten zu halten. Ist der Zugriff auf andere Komponenten der in Abschnitt 5 eingeführten, adaptierten Datenschicht notwendig, in denen komponentenübergreifende Daten verwaltet werden, geschieht dies durch Aufruf der definierten Komponenten-Interfaces ohne Änderungen an den Komponenten.

Wird eine Komponente der Anwendungsschicht hinzugefügt, muss diese über entsprechende Annotationen beim SpringBoot-Framework registriert werden. Abhängig von diesen Annotationen wird die Komponente unter bestimmten Bedingungen — wie beispielsweise beim Start der Anwendung – ausgeführt.

Komponenten der Datenschicht werden grundsätzlich nur von anderen Komponenten aufgerufen. Bis auf die Erstellung einer Objekt-Referenz auf die Implementierung der neuen Komponente in der Setup-Komponente sind keine weiteren Schritte notwendig.

Ein Beispiel für eine solche Komponente ist eine Monitoring-Komponente, die regelmäßig den Zustand aller bekannten Switches abfragt, und nur mittels einer Konfigurationsdatei konfiguriert wird. Dies ist eine sinnvolle, mögliche Erweiterung des WebCNC, um die in Abschnitt 6.3 diskutierten Inkonsistenzen zwischen netStateCache und tatsächlichem Netzwerkzustand abzumildern.

Komponenten mit Visualisierung

Das Hinzufügen von Backend-Komponenten mit einer Visualisierung im Frontend erfordert neben dem im vorherigen Paragraph dargelegten Ablauf einige weitere Schritte.

So muss die neue Komponente im Backend mit der Visualisierung im Frontend kommunizieren können. Dies geschieht durch das Hinzufügen eines oder mehrerer REST-Controller in der restcontroller-Komponente. Da die einzelnen Controller in separate Klassen ausgelagert sind, ist keine Anpassung der existierenden Controller notwendig.

Im Frontend muss für die Visualisierung der neuen Komponente eine neue Ansicht, analog zu den in Abschnitt 7.4 vorgestellten Ansichten, erstellt werden. Diese Ansicht muss, abhängig von ihrer konkreten Funktionalität, die REST-Controller des Backends aufrufen. Außerdem wird eine Navigationsmöglichkeit zu der neuen Ansicht benötigt. Dies soll gemäß des in Abschnitt 7.4 präsentierten Navigationsflusses im Regelfall durch das Hinzufügen einer Navigationsschaltfläche geschehen.

Als Beispiel lässt sich hier ebenfalls eine Monitoring-Komponente betrachten. So ist es denkbar, eine Monitoring-Komponente zu entwickeln, die nicht nur über eine Konfigurationsdatei konfiguriert wird, sondern im Frontend Möglichkeiten zur Konfiguration durch den Nutzer anbietet.

Visualisierung ohne Backend-Komponente

Die Architektur des WebCNC erlaubt es auch, neue Ansichten im Frontend hinzuzufügen, die nur auf bestehenden Daten arbeiten. Dazu muss lediglich eine neue Ansicht sowie eine Schaltfläche zur Navigation zu der neuen Ansicht hinzugefügt werden.

Ein denkbares Beispiel wäre hier das Hinzufügen einer weiteren Netzwerkvisualisierung, die im Vergleich zur existierenden, rudimentären Netzwerkvisualisierung mehr Details zu den Verbindungen zwischen den Geräten darstellt. Der Zweck einer solchen, neuen Netzwerkvisualisierung ist, dass der Nutzer die TSN-Konfiguration des Netzwerks vornehmen kann, ohne für Detailinformationen auf die tabellarischen Informationen der Switchübersicht-Ansicht zurückgreifen zu müssen.

Erweiterung der Southbound-Protokolle

Die zweite Art der Erweiterbarkeit ist die Erweiterung des WebCNC auf weitere Southbound-Protokolle wie beispielsweise NETCONF. Dies ist komplizierter als die bisher vorgestellen Komponentenerweiterungen und erfordert Änderungen an einigen existierenden Komponenten.

Zunächst müssen die Datenstrukturen des Backends angepasst werden. Wie in Abschnitt 6.2.1 dargestellt, müssen im Backend die *Switch*- und *SwitchInfo*-Klassen des Backends um weitere Felder, welche die Switch-Zugangsdaten des neuen Protokolls repräsentieren, ergänzt werden. Eine denkbare Lösung für den Umgang mit Switches, die nicht alle Southbound-Protokolle unterstützen, ist es, die Felder des nicht unterstützten Protokolls auf null-Werte zu setzen.

Anschließend müssen zwei Komponenten des Backends überarbeitet werden. Die *switchCredStore*-Komponente, deren Aufgabe es ist, die Zugangsdaten von Switches persistent zu speichern, muss durch eine neue Implementierung des *ISwitchCredStore*-Interfaces ersetzt werden. Diese Implementierung muss in der Lage sein, neben den bestehenden Zugangsdaten auch die Zugangsdaten des neuen Protokolls abzuspeichern und auszulesen.

An der *switchComms*-Komponente sind ebenfalls Änderungen notwendig. Die existierende *SNMPSwitchCommsCommunicator*-Subkomponente muss nicht geändert werden. Sie kann für die Implementierung der SNMP-Kommunikation unverändert weiter genutzt werden. Für die Implementierung des neuen Southbound-Protokolls ist es jedoch notwendig, einen weiteren *Communicator* für das neue Protokoll zu implementieren.

Um auszuwählen, welcher der Kommunikatoren für eine Switch-Operation genutzt werden soll, und diesen zu koordinieren, muss außerdem der *SwitchCommsManager* durch eine neue Implementierung des *ISwitchCommsManager*-Interfaces ersetzt werden. Die Entscheidung, welcher Kommunikator verwendet wird, muss auf Basis der Felder des erhaltenen *Switch*- oder *SwitchInfo*-Objektes getroffen werden.

Neben Anpassungen des Backends sind auch einige Modifikationen im Frontend notwendig. Hier muss, wie in Abschnitt 7.3 dargestellt, zunächst die *SwitchInfo*-Datenstruktur um die neuen Felder ergänzt werden. Die *SwitchData*-Datenstruktur ist im Frontend nicht betroffen, da sie keine protokollspezifischen Daten enthält.

Außerdem ist eine Anpassung der *NewSwitchForm*-Komponente in der *SwitchHinzufügen*-Ansicht notwendig. Diese Komponente wird genutzt, um die Zugangsdaten eines Switches zu erfassen, der zur Verwaltung durch den WebCNC registriert werden soll. Hier müssen Eingabefelder für die neuen Zugangsdaten hinzugefügt werden. Außerdem sollte die in der Komponente verortete Überprüfung der eingegebenen Werte auf ungültige Werte für die neuen Felder ergänzt werden, um dem Nutzer ein konstantes Fehler-Feedback zu geben.

Insgesamt ist die Erweiterung um neue Southbound-Protokolle also aufwändiger als die Einführung einer neuen Komponente. Dennoch ist es möglich, einen Großteil der bestehenden Komponenten unverändert weiterzuverwenden. Damit ist auch diese Form der Erweiterbarkeit gewährleistet.

8.2.3 Benutzerfreundlichkeit

Als letzte nicht-funktionale Anforderung wurde die Benutzerfreundlichkeit definiert. So soll der Einsatz des WebCNC mit möglichst geringem Konfigurationsaufwand für Administratoren möglich sein.

Für den Betrieb des Backends muss lediglich ein JRE auf dem Hostsystem installiert werden und ein TLS-Zertifikat zur Verfügung stehen. Dies sind Standardtätigkeiten eines Systemadministrators, die keine langwierige Einarbeitung erfordern. Die Konfiguration des WebCNC mit neuen Switches kann gänzlich durch das Frontend vorgenommen werden. Einige wenige Parameter - das Frontend-Zugangspasswort sowie die Timeout-Zeit und die Anzahl der Verbindungsversuche der Switch-Kommunikation - können über Launchparameter beim Start des Backends konfiguriert werden. Auch dies ist zur weiteren Vereinfachung optional, da bei Nichtangabe der Parameter konservative Standard-Werte eingesetzt werden. Damit ist der Betrieb des Backends unkompliziert und ohne spezielle Einarbeitung möglich.

Das Frontend hängt, wie bereits im Abschnitt zur Portabilität geklärt, lediglich von einem frei wählbaren Webserver ab. Die Konfiguration eines Webservers kann ebenfalls als Standardaufgabe eines Systemadministrators betrachtet werden. Die Flexibilität, beliebige Webserver zum Ausliefern der gebauten Dateien einzusetzen, ermöglicht es, individuell auf bereits bekannte Lösungen zu setzen, die keine neue Einarbeitung erfordern. Somit ist auch das Frontend einfach und unkompliziert einsetzbar.

9 Fazit und Ausblick

In diesem letzten Kapitel wird ein Rückblick auf die wichtigsten Eigenschaften und Entwurfsentscheidungen des WebCNC vorgenommen. Außerdem werden Möglichkeiten für weitere Arbeiten, die sich aus den Ergebnissen dieser Arbeit ableiten lassen, betrachtet.

9.1 Fazit

Diese Arbeit hat das Ziel, eine Architektur für einen TSN-CNC zu entwickeln, mit dem die manuelle Konfiguration der TSN-Switches eines Netzwerkes zentral und unkompliziert über einen normalen Webbrowser möglich ist. Der in Kapitel 5 präsentierte Architekturentwurf für den WebCNC ist insbesondere auf die Sicherstellung der Erweiterbarkeit des Systems fokussiert. So ist der komponentenbasierte Aufbau der Architektur geeignet, zusätzliche Komponenten zur Erweiterung der angebotenen Funktionalitäten einzuführen. Dadurch, dass bestehende Komponenten vordefinierte Interfaces implementieren und über diese mit anderen Komponenten kommunizieren, sind hierzu keine oder kaum Änderungen an anderen Komponenten nötig. Dies erlaubt es auch, bestehende Komponenten durch andere Implementierungen, welche die gleiche Schnittstelle bereitstellen, zu ersetzen, ohne Änderungen an anderen Komponenten vornehmen zu müssen. Erweiterungen des WebCNC um Unterstützung für weitere Southbound-Protokolle zur Kommunikation mit Netzwerkswitches sind architekturbedingt ebenfalls möglich. Neben der Erweiterbarkeit sind auch die Portabilität und Benutzerfreundlichkeit des WebCNC wichtige Faktoren für den Entwurf.

Der Entwurf als Web-Anwendung ermöglicht es Nutzern des WebCNC, die angebotenen Funktionalitäten zu nutzen, ohne lokal zusätzliche Software installieren zu müssen. Ein Web-Browser, der das Frontend von einem Webserver bezieht und mit dem Backend kommuniziert, ist alles, das für die Nutzung des WebCNC nötig ist. Dies wird insbesondere durch die Client-Server-Architektur möglich, bei der das Frontend HTTP-Anfragen an das Backend stellen kann, um auf Funktionen des CNC zuzugreifen. Das Frontend setzt also die Interaktion mit dem Nutzer um, während das Backend die Brücke zwischen dem Frontend und den Netzwerkswitches darstellt. Damit benötigt ein Frontend-Client nur das Backend als Kommunikationspartner und muss nie direkt mit den Switches kommunizieren. Solange der Frontend-Client das Backend erreichen kann, ist also eine Nutzung des WebCNC möglich - unabhängig von der direkten Erreichbarkeit der Switches durch den Nutzer.

In der Konzeptionierung des Frontends spielt außerdem die Nutzererfahrung in der Interaktion mit dem WebCNC eine entscheidende Rolle. So wurde in Abschnitt 7.4 ein ansichtsbasiertes Konzept zur Darstellung der verschiedenen Informationen erarbeitet. Außerdem wurde eine Möglichkeit zur Realisierung eines unkomplizierten Navigationsflusses durch diese Ansichten präsentiert.

Validiert wurde diese Architektur anhand eines Software-Prototypen, welcher die Architektur in einer praktisch nutzbaren Software umsetzt. Dieser Prototyp wird nicht nur zur Validierung der Funktionalität genutzt, sondern stellt auch einen Ausgangspunkt für zukünftige Erweiterungen des WebCNC dar.

9.2 Ausblick

Wie bereits in den vorherigen Kapiteln diskutiert wurde, ist die Erweiterbarkeit ein zentraler Aspekt der Architektur des WebCNC. Der Software-Prototyp erfüllt die essenziellen Aufgaben eines TSN-CNC. Aufgrund des begrenzten Umfangs der Arbeit steht jedoch eine Reihe an sinnvollen Erweiterungen aus, die den CNC um nützliche Funktionalitäten erweitern können.

So ist es denkbar, eine Monitoring-Komponente einzubeziehen, die periodisch den netState-Cache mit neuen Switchinformationen updated, um die in Abschnitt 6.3 diskutierten, potentiellen Inkonsistenzen zwischen netStateCache und realem Netzwerkzustand abzumildern. Dies kann entweder als eigenständige Komponente der Anwendungsschicht oder als externer Dienst, der den ReloadBackendController aufruft, geschehen.

Eine andere Erweiterung, die insbesondere für die Nutzung des WebCNC in großen Netzwerken mit vielen Nutzern sinnvoll sein kann, ist eine Überarbeitung der Zugriffskontrolle. So überprüfen die REST-Controller aktuell nur, ob ein globales Passwort korrekt gesetzt ist, um den Zugriff auf die Funktionen des Backends zu steuern. Dies ließe sich um ein nutzer- und gruppenbasiertes Rechtesystem erweitern, um den Zugriff einzelner Nutzer auf bestimmte Bereiche des Netzwerks einzuschränken.

Der WebCNC basiert auf den in Kapitel 2 beschriebenen Standards, um eine große Anzahl verschiedener Geräte zu unterstützen. Jedoch gibt es Netzwerkswitches, die andere Technologien einsetzen, gerade im Hinblick auf das Netzwerkkonfigurationsprotokoll. So gibt es eine steigende Anzahl an Netzwerkswitches, die statt SNMP auf NETCONF setzen. Eine Erweiterung des WebCNC auf die Unterstützung weiterer Konfigurationsprotokolle ist daher für Netzwerke sinnvoll, deren Switches über den ausschließlich auf SNMP setzenden Prototypen nicht angesprochen werden können.

Schließlich wäre auch eine Erweiterung des CNC um einen CUC, der automatisch von Geräten im Netzwerk Kommunikationsanforderungen erhält, ein entsprechendes Scheduling für die TSN-Konfiguration der Netzwerkswitches berechnet und diese Konfiguration mithilfe des WebCNC auf den Switches setzt, denkbar. Ein solcher CUC könnte parallel zum Frontend, das der manuellen Konfiguration dient, arbeiten, und ebenfalls die HTTP-API des Backends nutzen, um die Änderungen an der Switch-Konfiguration vorzunehmen. Da beide Komponenten mit dem selben Backend auch auf den Daten des selben netStateCache arbeiten, wäre automatisch ein konsistenter Zustand zwischen den von Frontend und CUC genutzten Daten gegeben. Diese Erweiterungsmöglichkeit ist ein ursprünglich nicht geplanter, aber sehr willkommener Nebeneffekt der modularen Architektur, der es ermöglicht, ohne eine Neuentwicklung des Backends Feature-Parität mit anderen TSN-CNCs zu erreichen oder diese sogar zu übertreffen.

Literaturverzeichnis

[ÁSP+22] I. Álvarez, A. Servera, J. Proenza, M. Ashjaei, S. Mubeen. „Implementing a First CNC for Scheduling and Configuring TSN Networks“. In: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2022, S. 1–4. DOI: 10.1109/ETFA52439.2022.9921518 (zitiert auf S. 29, 30).

[BAP+22] D. Bujosa, M. Ashjaei, A. V. Papadopoulos, T. Nolte, J. Proenza. „HERMES: Heuristic Multi-queue Scheduler for TSN Time-Triggered Traffic with Zero Reception Jitter Capabilities“. In: *Proceedings of the 30th International Conference on Real-Time Networks and Systems*. RTNS ’22. Paris, France: Association for Computing Machinery, 2022, S. 70–80. ISBN: 9781450396509. DOI: 10.1145/3534879.3534906. URL: https://doi.org/10.1145/3534879.3534906 (zitiert auf S. 29).

[Bis22] M. Bishop. *HTTP/3*. RFC 9114. Juni 2022. DOI: 10.17487/RFC9114. URL: https://www.rfc-editor.org/info/rfc9114 (zitiert auf S. 26).

[BM22] J. Bogner, M. Merkel. „To type or not to type? a systematic comparison of the software quality of JavaScript and typescript applications on GitHub“. In: MSR ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, S. 658–669. ISBN: 9781450393034. DOI: 10.1145/3524842.3528454. URL: https://doi.org/10.1145/3524842.3528454 (zitiert auf S. 60).

[BP23] D. Belson, L. Pardue. *Examining HTTP/3 usage one year on*. 2023. URL: https://blog.cloudflare.com/http3-usage-one-year-on (zitiert auf S. 26).

[BPT15] M. Belshe, R. Peon, M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. Mai 2015. DOI: 10.17487/RFC7540. URL: https://www.rfc-editor.org/info/rfc7540 (zitiert auf S. 26, 27).

[BW02] U. Blumenthal, B. Wijnen. *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*. RFC 3414. Dez. 2002. DOI: 10.17487/RFC3414. URL: https://www.rfc-editor.org/info/rfc3414 (zitiert auf S. 17, 21).

[CJ] H. Chahed, A. Johansson. *OpenCNC*. URL: https://git.cs.kau.se/hamzchah/opencnc_demo (zitiert auf S. 30, 31).

[CMPS02] D. J. D. Case, R. Mundy, D. Partain, B. Stewart. *Introduction and Applicability Statements for Internet-Standard Management Framework*. RFC 3410. Dez. 2002. DOI: 10.17487/RFC3410. URL: https://www.rfc-editor.org/info/rfc3410 (zitiert auf S. 17).

[CMRW96] D. J. D. Case, K. McCloghrie, D. M. T. Rose, S. Waldbusser. *Introduction to Community-based SNMPv2*. RFC 1901. Jan. 1996. DOI: 10.17487/RFC1901. URL: https://www.rfc-editor.org/info/rfc1901 (zitiert auf S. 20).

[DBN+01] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, J. Dray. *Advanced Encryption Standard (AES)*. 2001-11-26 2001. DOI: 10.6028/NIST.FIPS.197. URL: https://doi.org/10.6028/NIST.FIPS.197 (zitiert auf S. 21).

[DJ01] Donald E. Eastlake 3rd, P. Jones. *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174. Sep. 2001. DOI: 10.17487/RFC3174. URL: https://www.rfc-editor.org/info/rfc3174 (zitiert auf S. 21).

[DN16] F. Dürr, N. G. Nayak. „No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)“. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 2016, S. 203–212. ISBN: 9781450347877. DOI: 10.1145/2997465.2997494. URL: https://doi.org/10.1145/2997465.2997494 (zitiert auf S. 23).

[EBBS11] R. Enns, M. Björklund, A. Bierman, J. Schönwälder. *Network Configuration Protocol (NETCONF)*. RFC 6241. Juni 2011. DOI: 10.17487/RFC6241. URL: https://www.rfc-editor.org/info/rfc6241 (zitiert auf S. 21).

[Evaa] Evan You et al. *Vite*. URL: https://vitejs.dev/ (zitiert auf S. 60).

[Evab] Evan You et al. *VueJS*. URL: https://vuejs.org/ (zitiert auf S. 60).

[FLWR03] R. Frye, D. B. Levi, B. Wijnen, S. A. Routhier. *Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework*. RFC 3584. Aug. 2003. DOI: 10.17487/RFC3584. URL: https://www.rfc-editor.org/info/rfc3584 (zitiert auf S. 17).

[Foc] F. Fock. *SNMP4J*. URL: https://www.snmp4j.org/ (zitiert auf S. 42, 50).

[FR14] R. T. Fielding, J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. Juni 2014. DOI: 10.17487/RFC7231. URL: https://www.rfc-editor.org/info/rfc7231 (zitiert auf S. 27).

[GJS+21] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Buckley, D. Smith, G. Bierman. *The Java Language Specification*. 2021. URL: https://docs.oracle.com/javase/specs/jls/se17/jls17.pdf (zitiert auf S. 41).

[Gooa] Google. *Angular*. URL: https://angular.io/ (zitiert auf S. 60).

[Goob] Google. *gRPC*. URL: https://grpc.io/ (zitiert auf S. 30).

[HCWP02] D. Harrington, D. J. D. Case, B. Wijnen, R. Presuhn. *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)*. RFC 3412. Dez. 2002. DOI: 10.17487/RFC3412. URL: https://www.rfc-editor.org/info/rfc3412 (zitiert auf S. 17).

[HWP02] D. Harrington, B. Wijnen, R. Presuhn. *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*. RFC 3411. Dez. 2002. DOI: 10.17487/RFC3411. URL: https://www.rfc-editor.org/info/rfc3411 (zitiert auf S. 17).

[IEEE10] „IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams“. In: *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)* (2010), S. 1–72. DOI: 10.1109/IEEESTD.2010.8684664 (zitiert auf S. 25).

[IEEE16a] „IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery“. In: *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)* (2016), S. 1–146. DOI: 10.1109/IEEESTD.2016.7433915 (zitiert auf S. 25).

[IEEE16b] „IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic“. In: *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)* (2016), S. 1–57. DOI: 10.1109/IEEESTD.2016.8613095 (zitiert auf S. 14, 17, 21–23, 29, 35, 42).

[IEEE20] „IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems“. In: *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)* (2020), S. 1–499. DOI: 10.1109/IEEESTD.2020.9120376 (zitiert auf S. 24).

[IEEE22a] „IEEE Standard for Ethernet“. In: *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)* (2022), S. 1–7025. DOI: 10.1109/IEEESTD.2022.9844436 (zitiert auf S. 13, 21, 25).

[IEEE22b] „IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks“. In: *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)* (2022), S. 1–1993. DOI: 10.1109/IEEESTD.2022.10004498 (zitiert auf S. 13, 21).

[IEEE24] Institute of Electrical and Electronics Engineers. *Time-Sensitive Networking (TSN) Task Group*. 2024. URL: https://1.ieee802.org/tsn/ (zitiert auf S. 13, 21).

[IETF24] *Internet Engineering Task Force*. 2024. URL: https://www.ietf.org/ (zitiert auf S. 17).

[LF] LF Networking. *OpenDaylight*. URL: https://www.opendaylight.org/ (zitiert auf S. 33).

[LLM+20] A. Larrañaga, M. C. Lucas-Estañ, I. Martinez, I. Val, J. Gozalvez. „Analysis of 5G-TSN Integration to Support Industry 4.0“. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Bd. 1. 2020, S. 1111–1114. DOI: 10.1109/ETFA46521.2020.9212141 (zitiert auf S. 13).

[LS19] L. Lo Bello, W. Steiner. „A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems“. In: *Proceedings of the IEEE* 107.6 (2019), S. 1094–1120. DOI: 10.1109/JPROC.2019.2905334 (zitiert auf S. 13).

[LYB+21] T. Lindholm, F. Yellin, G. Bracha, A. Buckley, D. Smith. *The Java Virtual Machine Specification*. 2021. URL: https://docs.oracle.com/javase/specs/jvms/se17/jvms17.pdf (zitiert auf S. 41).

[MBM04] F. Maino, U. Blumenthal, K. McCloghrie. *The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model*. RFC 3826. Juni 2004. DOI: 10.17487/RFC3826. URL: https://www.rfc-editor.org/info/rfc3826 (zitiert auf S. 17, 21).

[Met] Meta Platforms Inc. *React*. URL: https://react.dev/ (zitiert auf S. 60).

[Mic] Microsoft. *TypeScript*. URL: https://www.typescriptlang.org/ (zitiert auf S. 60).

[MLS02] P. Meyer, D. B. Levi, B. Stewart. *Simple Network Management Protocol (SNMP) Applications*. RFC 3413. Dez. 2002. DOI: 10.17487/RFC3413. URL: https://www.rfc-editor.org/info/rfc3413 (zitiert auf S. 17).

[MQJ+] G. Maier, Q, K. Jiang, S. Zhou, K.-C. Wang, R. Izard, S. Das, M. Yang, A. Wundsam, vishnu-emmadi, D. Coutinho, G. Barrineau, J. Parraga, W. Felter, sanjivininaikar, A. Tahir, H. Chiu, S. Banerjee, sandeep hebbani, R. Lane, A. Ferguson, J. Chappell, M. Mehta, C. Banse, paaguti-work, V. Yazıcı, S. Shimizu, aweally, Alex, sandipshah. *Floodlight OpenFlow Controller (OSS)*. URL: https://github.com/floodlight/floodlight (zitiert auf S. 33).

[MS05] D. R. Mauro, K. J. Schmidt. *Essential SNMP, Second Edition*. O'Reilly Media, Inc., 2005. ISBN: 0596008406 (zitiert auf S. 19).

[MSPM99a] K. McCloghrie, J. Schönwälder, D. T. Perkins, K. McCloghrie. *Conformance Statements for SMIv2*. RFC 2580. Apr. 1999. DOI: 10.17487/RFC2580. URL: https://www.rfc-editor.org/info/rfc2580 (zitiert auf S. 17, 18).

[MSPM99b] K. McCloghrie, J. Schönwälder, D. T. Perkins, K. McCloghrie. *Structure of Management Information Version 2 (SMIv2)*. RFC 2578. Apr. 1999. DOI: 10.17487/RFC2578. URL: https://www.rfc-editor.org/info/rfc2578 (zitiert auf S. 17–19).

[MSPM99c] K. McCloghrie, J. Schönwälder, D. T. Perkins, K. McCloghrie. *Textual Conventions for SMIv2*. RFC 2579. Apr. 1999. DOI: 10.17487/RFC2579. URL: https://www.rfc-editor.org/info/rfc2579 (zitiert auf S. 17, 18).

[MSz+] Murphy, C. Scott, zarifis, J. Tsao, A. Panda, D. Guedes, P. Peresini, S. Whitlock, A. Bezroutchko, nbjsw, S. St. John, F. Németh, S. Russell, Y. Takashi, A. Elhassany, J. Czentye, J. Sommers, J. T. Araújo, K. Slavov, A. Al-Shabibi. *POX network software platform*. URL: https://github.com/noxrepo/pox (zitiert auf S. 33).

[MWP02] K. McCloghrie, B. Wijnen, R. Presuhn. *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*. RFC 3415. Dez. 2002. DOI: 10.17487/RFC3415. URL: https://www.rfc-editor.org/info/rfc3415 (zitiert auf S. 17, 21).

[Nat88] National Institute of Standards and Technology. *DATA ENCRYPTION STANDARD (DES)*. 1988-1-22 1988. DOI: 10.6028/NIST.FIPS.46-2. URL: https://doi.org/10.6028/NIST.FIPS.46-2 (zitiert auf S. 21).

[Ope15] Open Networking Foundation. *OpenFlow Switch Specification*. 2015. URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (zitiert auf S. 33).

[Ora] Oracle. *javadoc - The Java API Documentation Generator*. URL: https://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html (zitiert auf S. 42).

[OTX+] M. Otto, J. Thornton, XhmikosR, G. Sotiropoulos, P. Lauke, J. Déramond, G. Poupard, R. Sharma, apladev, M. Cuppens, J. Servoire, G. Mazovetskiy. *Bootstrap*. URL: https://getbootstrap.com/ (zitiert auf S. 60).

[Pre02a] R. Presuhn. *Transport Mappings for the Simple Network Management Protocol (SNMP)*. RFC 3417. Dez. 2002. DOI: 10.17487/RFC3417. URL: https://www.rfc-editor.org/info/rfc3417 (zitiert auf S. 17).

[Pre02b] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. RFC 3416. Dez. 2002. DOI: 10.17487/RFC3416. URL: https://www.rfc-editor.org/info/rfc3416 (zitiert auf S. 17, 20).

[RD08] E. Rescorla, T. Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/info/rfc5246 (zitiert auf S. 27).

[Riv92] R. L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321. Apr. 1992. DOI: 10.17487/RFC1321. URL: https://www.rfc-editor.org/info/rfc1321 (zitiert auf S. 21).

[RKS+20] B. Rother, M. Kasparick, E. Schweißguth, F. Golatowski, D. Timmermann. „Automatic Configuration of a TSN Network for SDC-based Medical Device Networks“. In: *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. 2020, S. 1–8. DOI: 10.1109/WFCS47810.2020.9114471 (zitiert auf S. 13).

[RM90] D. M. T. Rose, K. McCloghrie. *Structure and identification of management information for TCP/IP-based internets*. RFC 1155. Mai 1990. DOI: 10.17487/RFC1155. URL: https://www.rfc-editor.org/info/rfc1155 (zitiert auf S. 17, 18).

[RM91] D. M. T. Rose, K. McCloghrie. *Concise MIB definitions*. RFC 1212. März 1991. DOI: 10.17487/RFC1212. URL: https://www.rfc-editor.org/info/rfc1212 (zitiert auf S. 17, 18).

[Ros91] D. M. T. Rose. *Convention for defining traps for use with the SNMP*. RFC 1215. März 1991. DOI: 10.17487/RFC1215. URL: https://www.rfc-editor.org/info/rfc1215 (zitiert auf S. 17, 18).

[Ryu] Ryu SDN Framework Community. *Ryu SDN Framework*. URL: https://ryu-sdn.org/ (zitiert auf S. 33).

[SAO+24] M. Stubbs, V. Asturiano, C. J. Otvik, F. Dreier, muralimulagalapati, N. Povo. *react-force-graph*. 10. Feb. 2024. URL: https://github.com/vasturiano/react-force-graph (zitiert auf S. 60).

[SHK+24] T. Saloranta, R. Heaton, J. H. Kim, P. Fanning, P. Brown, D. Pilato, D. Stephens, P. Lundberg, R. Papp, T. Saloranta, R. zu Dohna, R. Sennov, R. Vieira, P. Remen, N. Y. Man, O. Kopp, N. Resare, M. Frost, L. Marek, A. Gómez-Goiri, S. Aggarwal, Simulant87, S. Slavić, T. H. Hüpner, T. Aizenberg, T. Howard, W. Tanaka, based2, litetex, A. Yu. *Jackson*. 20. Feb. 2024. URL: https://github.com/FasterXML/jackson (zitiert auf S. 41, 56).

[SSBM22] R. Shakir, A. Shaikh, P. Borman, C. M. Marcus Hines Carl Lebsack. *gRPC Network Management Interface*. 2022. URL: https://datatracker.ietf.org/meeting/98/materials/slides-98-rtgwg-gnmi-intro-draft-openconfig-rtgwg-gnmi-spec-00 (zitiert auf S. 30).

[Sys] I. V. Sysoev. *nginx*. URL: https://nginx.org/ (zitiert auf S. 75).

[SZ18] S. Samii, H. Zinner. „Level 5 by Layer 2: Time-Sensitive Networking for Autonomous Vehicles“. In: *IEEE Communications Standards Magazine* 2.2 (2018), S. 62–68. DOI: 10.1109/MCOMSTD.2018.1700079 (zitiert auf S. 13).

[TASFa] The Apache Software Foundation. *Apache Log4j*. URL: https://logging.apache.org/log4j/2.x/ (zitiert auf S. 41, 54).

[TASFb] The Apache Software Foundation. *Apache Maven*. URL: https://maven.apache.org/ (zitiert auf S. 42).

[TASFc] The Apache Software Foundation. *Apache Tomcat*. URL: https://tomcat.apache.org/ (zitiert auf S. 41, 55).

[TD21] F. von Tüllenburg, J. L. Du. „The Effect of Time-Sensitive Networking Onto Performance and Robustness of Power Grid Protection“. In: *2021 11th International Conference on Advanced Computer Information Technologies (ACIT)*. 2021, S. 576–581. DOI: 10.1109/ACIT52158.2021.9548620 (zitiert auf S. 13).

[TTT] TTTech Industrial Automation AG. *SlateXNS*. URL: https://www.tttech-industrial.com/products/slate/slate-xns (zitiert auf S. 32).

[WNW+] A. Wilkinson, S. Nicoll, P. Webb, M. Bhave, M. Halbritter, S. Frederick, B. Clozel, J. Lim, C. Dreis, D. Syer, S. Builds, E. M. Gonzales, V. Pavic, C. Dupuis, G. Russell, D. Nosan, S. Deleuze, I. Suhorukov, Y. Zhou, C. Bono, J. Ivanov, A. Yudovin, A. Bilan, M. Simons, K. Shimizu, polarbear567, G. L. Turnquist, O. Drotbohm, terminux, H. YunKun. *Spring Boot*. URL: https://github.com/spring-projects/spring-boot (zitiert auf S. 41, 54).

[ZUJ+] M. Zabriskie, N. Uraltsev, Jay, D. Mozgovoy, E. Morehouse, R. Norte, J. Beckwith, M. Laine, X. Zhong, R. Gibson, R. Haszing, Y. Flores, B. Carp, D. Lopretto, I. Gazit, J. Tangelder, F. Martins, M. Di Prisco, G. Cheng, H. B. Manilla, staticinstance, A. Fiorette, 38elements, C. Timmermans, M. Bargiel, Kosta, Nero, N. Bottomley. *Axios*. URL: https://github.com/axios/axios (zitiert auf S. 60).

Alle URLs wurden zuletzt am 20. 04. 2024 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift