HAW HAMBURG

BACHELORTHESIS
Mohammad Fazel Soltani

Ein Transaktionsmodell zur Netzwerkkonfiguration mit Zeitanforderungen am Beispiel von Fahrzeugen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Mohammad Fazel Soltani

Ein Transaktionsmodell zur Netzwerkkonfiguration mit Zeitanforderungen am Beispiel von Fahrzeugen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Martin Becke

Eingereicht am: 14. Juli 2021

Mohammad Fazel Soltani

Thema der Arbeit

Ein Transaktionsmodell zur Netzwerkkonfiguration mit Zeitanforderungen am Beispiel von Fahrzeugen

Stichworte

Transaktionsmodell, Netzwerkrekonfiguration, SDN, TSN, NETCONF, Echtzeit-Ethernet

Kurzzusammenfassung

Zeitkritische Netzwerke müssen bei veränderten Anforderungen rekonfiguriert werden. Rekonfigurationen in zeitkritischen Netzwerken müssen unter der Einhaltung von Deadlines zeitkritischer Prozesse erfolgen. Weiter müssen Rekonfigurationen zur Laufzeit zeitsynchron und transaktional auf betroffenen Netzwerkgeräten erfolgen. Zeitkritische Netzwerke wie ethernetbasierte Fahrzeugboardnetzwerke werden durch gängige Standards wie TSN ermöglicht, in denen Datenpakete unter zeitlichen Fristen übertragen werden müssen. Durch SDN wird die Logik der Netzwerkgeräte in den SDN-Controller verlagert, um diese Netzwerke programmierbar zu machen, wodurch Erweiterungen wie Transaktionalität implementiert werden können. Diese Arbeit evaluiert ein Transaktionsmodell, welches Rekonfigurationen transaktional und zeitsynchron in einem programmierbaren und zeitkritischen Netzwerk ohne Paketverlust und Verzögerungen durchführt. Im Rahmen einer Fallstudie konnte gezeigt werden, dass das Transaktionsmodell weder Verzögerungen noch Paketverluste an bestehendem sowie hinzugefügtem Datenverkehr verursacht.

Mohammad Fazel Soltani

Title of Thesis

A transaction model for network configuration with time requirements using the example of vehicles

Keywords

transaction model, Network reconfiguration, SDN, TSN, NETCONF, Real-Time Ethernet

Abstract

Time-critical networks must be reconfigured when requirements change. Reconfigurations in time-critical networks must be performed in compliance with the deadlines of time-critical processes. Furthermore, reconfigurations must be performed at runtime in a time-synchronous and transactional manner on affected network devices. Time-critical networks such as Ethernet-based vehicle board networks are enabled by common standards such as TSN, in which data packets must be transmitted under time-critical deadlines. SDN moves the logic of network devices into the SDN controller to make these networks programmable, allowing extensions such as transactionality to be implemented. This thesis evaluates a transactional model that performs reconfigurations transactionally and time-synchronously in a programmable and time-critical network without packet loss and delays. In a case study, it was shown that the transactional model does not cause delays or packet loss on existing and added traffic.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Bereits vor 30 Jahren setzte man sich mit der Thematik der Rekonfiguration von Netzwerken auseinander. Netzwerke müssen bei veränderten Anforderungen rekonfiguriert werden und in einem konsistenten Zustand sein. Steigende Latenzen und Paketverlust sind häufige Probleme, die Inkonsistenz in Netzwerken aufweisen. Diese Probleme, die während und nach einer Netzwerkrekonfigurationen passieren könnten, können auch in zeitkritischen Netzwerken auftreten, wie beispielsweise in ethernetbasierten Fahrzeugboardnetzwerken. Der Ausfall eines zu sendenden Signals in solch einem zeitkritischen Netzwerk könnte fatale Folgen haben. Ein inkonsistentes ethernetbasiertes Fahrzeugboardnetzwerk kann entweder dazu führen, dass Signale an falsche Steuergeräte oder sogar gar nicht vermittelt werden. Die Rekonfiguration von ethernetbasierten Fahrzeugboardnetzwerke muss dafür sorgen, dass diese atomar durchgeführt und den zeitlichen Anforderungen gerecht werden. Mit Hilfe von Software-defined Networking, bei der die Kontrollebene des Netzwerks von den Switches auf einen zentralen Host separiert wird und somit zentralisierte Entscheidungen über den Netzwerkfluss treffen kann, können dynamische Rekonfigurationen von Netzwerkflüssen durch Hosts ermöglicht werden [11]. Das Konzept der Transaktionen, welches seinen Ursprung aus dem Themengebiet der Datenbanken hat, kann als eine mögliche Art der Netzwerkrekonfiguration verwendet werden. Hierdurch können mehrere Änderungen als ein gemeinsamer Arbeitsschritt beziehungsweise als eine Transaktion zusammengefasst und atomar an das zeitkritische Netzwerk eingeführt werden. Dies allein reicht jedoch nicht aus, um die benannten Probleme zu vermeiden beziehungsweise gar erst in Kraft treten zu lassen. Es muss neben der transaktionalen Netzwerkrekonfiguration auch der Aspekt der Zeitsynchronität in Betracht gezogen werden. Durch die Zeitsynchronität wird garantiert, dass transaktionale Netzwerkrekonfigurationen auch gleichzeitig auf allen beteiligten Switches durchgeführt werden. Damit die Garantie erfüllt wird, muss ein fest definierter Zeitpunkt errechnet werden, in dem alle Switches eines Netzwerkes gleichzeitig die Rekonfiguration ausführen.

Das Ziel dieser Arbeit besteht darin, ein zeitsynchrones Transaktionsmodell in einem programmierbaren und zeitkritischen Netzwerk zu etablieren und dessen Auswirkungen auf dieses Netzwerk zu evaluieren. Durch zeitsynchrone Transaktionen sollen in einem programmierbaren und zeitkritischen Netzwerk Änderungen an der Netzwerkkonfiguration von Echtzeit-Switches durchgeführt werden. Außerdem sollen zeitsynchrone Transaktionen den Zeitplan (Schedule) des zeitkritischen Netzwerkes berücksichtigen, keinen Paketverlust bei bestehendem und hinzukommenden Datenverkehr verursachen sowie priorisierte Datenpakete nicht verzögern. In dieser Arbeit wird ein Konzept für ein zeitsynchrones Transaktionsmodell entwickelt und im diskreten Ereignissimulator OMNeT++ implementiert und evaluiert. Es wird im Rahmen einer Fallstudie die Auswirkungen von verschiedenen Arten der Netzwerkrekonfiguration, zu dem das zeitsynchrone Transaktionsmodell dazugehört, gemessen und beobachtet. Diese Arbeit ist in der Communication-over-Realtime-Ethernet Research Group (CoRE RG) der HAW Hamburg entstanden. Die CoRE RG befasst sich mit Echtzeit-Ethernet-Kommunikation in Fahrzeugen.

1.2 Aufbau der Arbeit

Die Arbeit ist in sieben Kapitel gegliedert. In Kapitel 2 wird das Basiswissen dieser Arbeit vermittelt. Dabei werden auf Themen eingegangen wie Transaktionen, Echtzeiterweiterung von Ethernet, Software-defined Networking, Time-Sensitive Networking, sowie die Protokolle OpenFlow und NETCONF. In Kapitel 3 wird auf vergangene Arbeiten eingegangen, die sich mit Netzwerkrekonfigurationen von programmierbaren oder zeitkritischen Netzwerken befasst haben. Jeder dieser vergangenen Arbeiten wird von dieser Arbeit abgegrenzt. In Kapitel 4 werden die Anforderungen des zeitsynchronen Transaktionsmodells gestellt. Das Konzept des zeitsynchronen Transaktionsmodells wird in Kapitel 5 behandelt. Hier werden alle relevanten Schritte, die bei der Konzeption des zeitsynchronen Transaktionsmodell gegangen wurden, erklärt. Alle Details zur Implementierung und zur Qualitätssicherung werden in Kapitel 6 beschrieben. In Kapitel 7 werden die Anforderungen dieser Arbeit evaluiert. Es werden die Messungen beschrieben und die Ergebnisse präsentiert. Zum Schluss folgt in Kapitel 8 das Fazit des zeitsynchronen Transaktionsmodells. Außerdem wird auf Anlaufstellen eingegangen, die für zukünftige Arbeiten von Relevanz sein könnten.

2 Grundlagen

In diesem Kapitel wird das Grundwissen der Arbeit vermittelt. Zu Beginn wird auf Transaktionen im Kontext von Datenbanken und verteilten Systemen eingegangen. Darauf folgen Technologien wie Time-Sensitive Networking und Software-defined Networking. Außerdem wird auf NETCONF und OpenFlow eingegangen. Zum Schluss folgt eine Erläuterung der in dieser Arbeit verwendeten Simulationsumgebung.

2.1 Transaktionen

Die folgende Erklärung zu Transaktionen stützt sich auf die Lehrbücher [28, 18, 26, 25, 27]. Eine Transaktion wird als eine endliche Folge von Aktionen bezeichnet, die in ihrer Gesamtheit als ein ununterbrochener Arbeitsschritt angesehen wird. Das Konzept der Transaktion hat ihren Ursprung im Bereich der Datenbanken. Das Ziel war, ein Konzept zu entwickeln, damit die Konsistenz von Servern, die zur Datenhaltung dienen, durch den Zugriff von mehreren Clients gewährleistet wird. Im Laufe der Zeit wurde das Konzept der Transaktion auch an die Belange von verteilten Systemen und ihrer verteilten Umgebung angepasst ([27] Kapitel 4.1).

2.1.1 Eigenschaften von Transaktionen

Mandl gibt in [18] eine Einführung in das Themengebiet der Transaktion, die hier auszugsweise dargestellt wird. Es wird zwischen lokalen und globalen Transaktionen unterschieden. Eine lokale Transaktion führt Änderungen an den Ressourcen auf einem Rechnersystem aus. Dabei muss keine Abstimmung zu weiteren Rechnersystemen gemacht werden. Eine globale Transaktion ist hingegen eine Rechner-übergreifende Transaktion, bei der eine Abstimmung von den beteiligten Rechnersystemen erforderlich ist, um Änderungen an die Ressourcen der beteiligten Rechnersysteme zu vollziehen. Eine Transaktion führt Operationen aus, um Änderungen an einem System durchzuführen. Änderungen,

die während der Transaktion vorgenommen wurden, werden auf einem persistenten Speicher gesichert und damit dauerhaft gemacht. Diesen Vorgang nennt man Commit. Der Vorgang, der Änderungen während der Transaktion vornimmt und wieder rückgängig macht, heißt Rollback. Eine Transaktion kann die folgenden Eigenschaften haben:

- **Atomizität**
  Eine Transaktion muss die Eigenschaft bewahren, dass sie entweder ganz oder gar nicht ausgeführt wird. Sie muss atomar sein.

- **Konsistenz**
  Ein System wird durch eine Transaktion von einem konsistenten Zustand in einen neuen konsistenten Zustand überführt. Hierbei muss diese Eigenschaft nach Fehlern während einer Transaktion beispielsweise bei einem Systemfehler, bei Stromausfall oder bei Absturz des Netzwerks oder des Programms einzuhalten sein.

- **Isolation**
  Nebenläufige Transaktionen dürfen sich nicht beim Zugriff von gemeinsamen Ressourcen behindern. Diese Transaktionen müssen sich wie serielle Ausführungen verhalten.

- **Dauerhaftigkeit**
  Jegliche Änderungen nach einer erfolgreichen Transaktion bleiben in einem System persistent und können in einem nicht-flüchtigen Speicher gesichert werden.

Eine Transaktion, die die benannten vier Eigenschaften erfüllt, wird als ACID-Transaktion bezeichnet.

2.1.2 Mechanismen in Transaktionen

Zur Einhaltung der Eigenschaften wurden im Laufe der Zeit Mechanismen für Transaktionen entwickelt. Für diese Arbeit sind die Mechanismen **Nebenläufigkeitskontrolle** und **Koordination** von Relevanz und werden in den nächsten Abschnitten genauer erläutert.

Nebenläufigkeitskontrolle

Die Nebenläufigkeitskontrolle ist ein Mechanismus zur Synchronisation von Transaktionen. Sie ist dafür zuständig, dass der konkurrierende Zugriff auf einem System ermöglicht wird, ohne in Inkonsistenz zu verfallen. Es ist möglich, dass mehrere Transaktionen abwechselnd auf das System zugreifen, bis sie mit einem Commit oder Rollback beendet werden. Dabei ist es auch möglich, dass angefragte Daten aus dem System für mehrere Transaktionen benötigt werden. Falls in diesem Fall keine geeigneten Maßnahmen getroffen werden, kann es zu Anomalien führen. Diese Anomalien können zum Beispiel lost-update, das Phantomproblem, dirty-read oder unrepeatable-read sein. Für weitere Details zu Anomalien wird auf ([18] Kapitel 4.2.4), ([28] Kapitel 14.1), [26] und [25] verwiesen.

Damit es nicht zu solchen Anomalien kommt, werden Sperrprotokolle für die Nebenläufigkeitskontrolle verwendet. Das **Zwei-Phasen-Sperrprotokoll** ist ein Protokoll, mit dem eine Transaktion den exklusiven Zugriff auf die Ressourcen eines Systems erwirbt, um ohne den konkurrierenden Zugriff weiterer Transaktionen seine Aktionen durchzuführen. Der Ablauf des Zwei-Phasen-Sperrprotokolls ist in zwei Phasen gegliedert, welche in Abbildung 2.1 dargestellt sind. Zu Beginn einer Transaktion gibt es eine Zunahmephase. In

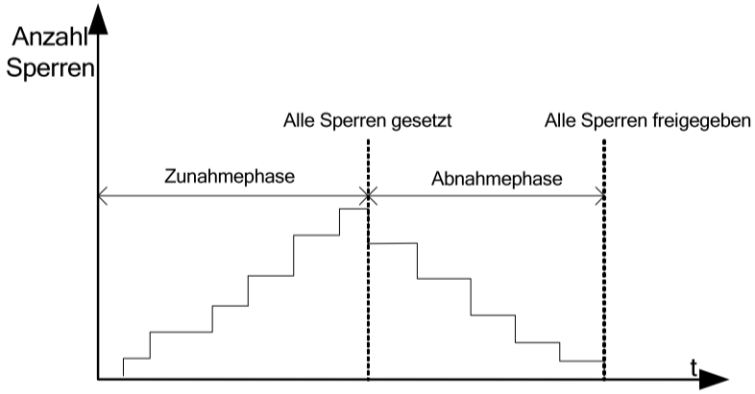

Abbildung 2.1: Das Zwei-Phasen-Sperren (Quelle: [18] Abbildung 4-7)

dieser Phase fordert eine Transaktion jegliche Sperren für sich an. Nachdem die Transaktion vollendet ist, werden die Sperren nacheinander freigegeben. Diese Phase lautet Abnahmephase. Mit der Freigabe einer Ressource hat nun eine weitere Transaktion die Möglichkeit, auf diese freigegebene Ressource zuzugreifen. Dadurch ist das Risiko eines Deadlocks immer noch vorhanden. Im Kontext einer Transaktion wird ein Deadlock als

eine Situation angesehen, in der sich Transaktionen in einem Wartezustand befinden, wobei jede dieser Transaktionen darauf wartet, dass einer der Transaktionen eine gesperrte Ressource wieder freigibt.

Das **strikte Zwei-Phasen-Sperrprotokoll** hat den Vorteil, dass alle Sperren nach der letzten Aktion einer Transaktion freigegeben werden. Dadurch wird das Risiko eines Deadlocks vermindert. Jedoch steigt die Dauer der zu entsperrenden Ressourcen, da alle Sperren einmalig freigegeben werden. Zur Verdeutlichung des strikten Zwei-Phase-Sperrens dient Abbildung 2.2.

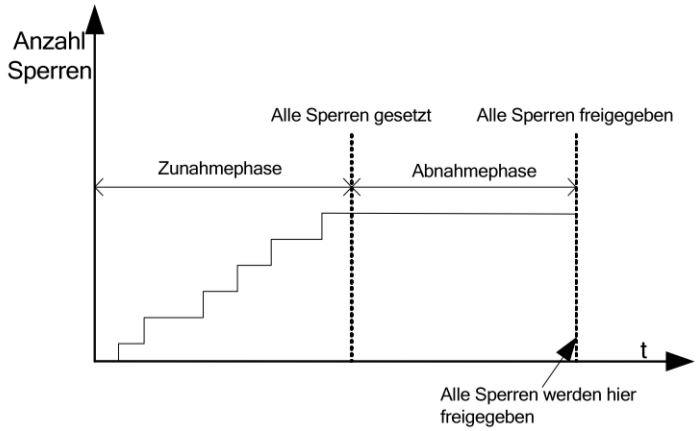

Abbildung 2.2: Das strikte Zwei-Phasen-Sperren (Quelle: [18] Abbildung 4-8)

Für das zeitsynchrone Transaktionsmodell dieser Arbeit wird die Idee des **strikten Zwei-Phasen-Sperrprotokolls** verwendet. Hieraus wird die Sperrphase in Unterabschnitt 5.2.1 und die Entsperrphase in Unterabschnitt 5.2.4 konzipiert und im Konzept auf Kapitel 5 detailliert beschrieben.

Koordination

Die Koordination von Transaktionen ist ein Mechanismus, der dafür gedacht ist, dass mehrere Instanzen die erfolgreich durchgeführten Änderungen gemeinsam bestätigen. Sie wird in verteilten Transaktionen angewendet. Zur Realisierung der Koordination von Transaktionen werden Koordinationsprotokolle angewendet. Eines davon ist das **Two-Phase-Commit-Protokoll** und in Abbildung 2.3 abgebildet. Beim Two-Phase-Commit-Protokoll wird ein Koordinator ausgesucht, der für das Einleiten und Kontrollieren der

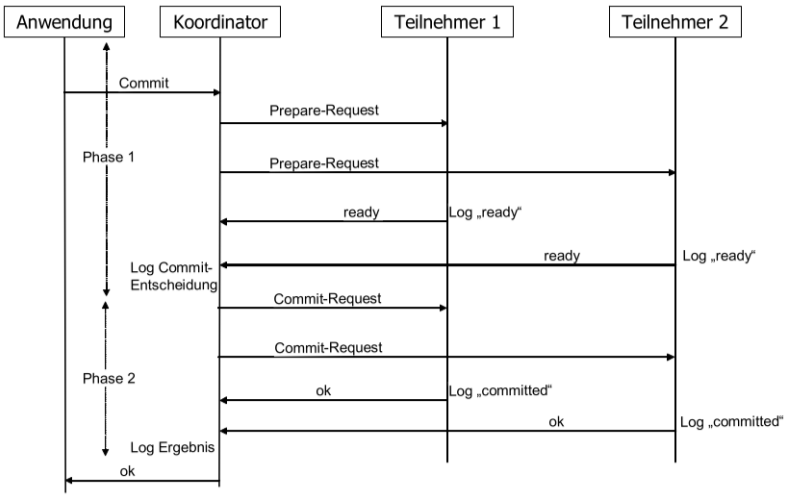

Abbildung 2.3: Das Zwei-Phasen-Commit-Protokoll im erfolgreichen Fall (Quelle: [18] Abbildung 4-12)

Transaktion zuständig ist. Zunächst wird ein Koordinator gewählt, der die erfolgreich durchgeführten Änderungen einer Transaktion an alle beteiligten Teilnehmer der Transaktion mitteilen soll. Danach folgen zwei Phasen des Two-Phase-Commit-Protokolls. In der Wahlphase fragt der Koordinator alle Teilnehmer der Transaktion, ob diese einen Commit durchführen können. Die Teilnehmer teilen dem Koordinator ihre Entscheidung mit. In der Entscheidungsphase entscheidet der Koordinator je nach erhaltener Antwort aller Teilnehmer der Transaktion, ob diese den Commit durchführen können oder nicht. Nur wenn alle Teilnehmer der Transaktion ein Ja zurückmelden, wird der Commit durchgeführt. Falls ein Teilnehmer ein Nein oder eine nicht definierte Antwort zurückmeldet, werden alle durchgeführten Änderungen zurückgesetzt und die Transaktion beendet. In beiden Fällen bekommen alle Teilnehmer eine Benachrichtigung und führen diese aus.

Mit dem Two-Phase-Commit-Protokoll ist es zwar möglich, den Konsens zwischen mehreren Teilnehmern einer Transaktion zu finden, jedoch verbirgt sich ein Problem darin. Im Falle von Nachrichtenverlust seitens der Kommunikation erhalten Teilnehmer oder Koordinator nicht sofort eine Nachricht und müssen somit warten. Aus diesem Grund sind Timer zur Begrenzung der Wartezeit erforderlich [18].

Für das zeitsynchrone Transaktionsmodell dieser Arbeit wird die Idee des **Two-Phase-Commit-Protokolls** verwendet. Sie wird als Bestandteil der Bestätigungsphase in Unterabschnitt 5.2.3 konzipiert und im Konzept auf Kapitel 5 detailliert beschrieben.

2.1.3 Transaktionsmodelle

Transaktionen können durch verschiedene Transaktionsmodelle realisiert werden. Die häufig verwendeten Transaktionsmodelle in Datenbanken und verteilten System lauten flaches, verkettetes und geschachteltes Transaktionsmodell.

Flaches Transaktionsmodell In diesem Transaktionsmodell wird der gesamte Ablauf über einen Teilnehmer einer Transaktion koordiniert. Dieser Teilnehmer teilt sequenziell jeden weiteren beteiligten Teilnehmer der Transaktion mit, welche Änderungen durchzuführen sind. Das flache Transaktionsmodell erfüllt alle vier Eigenschaften einer Transaktion ([33] Kapitel 3.1). Eines der Vorteile vom flachen Transaktionsmodell ist das schnelle Abarbeiten der Transaktion. Dadurch ist die Ausführung von kurzer Dauer.

Verkettetes Transaktionsmodell Die verkettete Transaktion erlaubt die sequenzielle Ausführung mehrerer flacher Transaktionen, wobei die Informationen einer Transaktion beim Übergang weitergeleitet wird. Jede Transaktion in der Verkettung erfüllt die vier Eigenschaften einer Transaktion und terminiert beim Übergang zur nächsten Transaktion. Die Verkettung hat den Vorteil, dass das Bestätigen der Transaktion und der Beginn der nächsten Transaktion als eine Operation angesehen wird und dadurch ein Leistungsgewinn erzielt werden kann ([18] Kapitel 4.2.2). In Datenbanksystemen werden zu den flachen Transaktionen Sicherungspunkte realisiert. Mit Sicherungspunkten wird innerhalb einer Transaktion sichergestellt, dass auf sie wieder aufgesetzt werden kann, ohne die gesamte Transaktion zurückzusetzen. Bei einem Abbruch der Transaktion verfallen jedoch die Sicherungspunkte und werden zurückgesetzt.

Geschachteltes Transaktionsmodell Geschachtelte Transaktionen bestehen aus Subtransaktionen, die wiederum auch geschachtelt sind. Dadurch entsteht ein Transaktionsbaum, in dessen Blätter die Änderungen vorhanden sind und wie in einer flachen Transaktion bearbeitet werden. Abbildung 2.4 visualisiert dies. Die Subtransaktionen im Transaktionsbaum können nebenläufig ausgeführt werden. Jede Subtransaktion wird atomar für sich abgewickelt, jedoch gilt die Persistenz für die gesamte Transaktion. Geschachtelte Transaktionen finden ihre Anwendung in verteilten Systemen wieder. Dies liegt zum einen darin, da sie zu einer Erhöhung der Parallelität umfangreicher Transaktionen führen und zum anderen robust gegen Abbrüche von Subtransaktionen sind ([18] Kapitel 4.2.2).

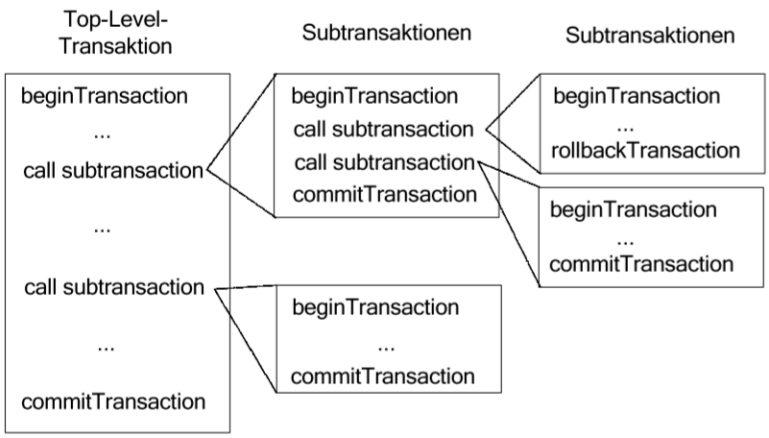

Abbildung 2.4: Der Transaktionsbaum (Quelle: [18] Abbildung 4-2)

Für das zeitsynchrone Transaktionsmodell dieser Arbeit wird die Idee des **geschachtelten Transaktionsmodells** verwendet. Sie wird als wesentlicher Bestandteil der Änderungsphase in Unterabschnitt 5.2.2 konzipiert und im Konzept auf Kapitel 5 detailliert beschrieben.

2.2 Time-Sensitive Networking

Time-Sensitive Networking (TSN) ist eine Technologie, welche Ethernet um die Einhaltung von Zeitgarantien erweitert. TSN besteht aus einer Menge von Standards, die von der TSN-Task-Group [32] standardisiert werden. Das Ziel von TSN ist es, ein verlässliches Zeitverhalten in einem zeitkritischen Netzwerk zu erreichen. Der Standard IEEE 802.1Q-2018 [13] enthält bereits viele der einzelnen Substandards von TSN. In den folgenden Abschnitten werden auf die Echtzeiterweiterung von Ethernet, den Standard IEEE 802.1Q-2018 [13] und den darin enthaltenen Substandard IEEE 802.1Qbv-2015 [12] eingegangen. Zuvor wird der Begriff Latenz definiert.

Latenz Die Latenz beschreibt die Dauer einer Nachricht von einem Sender zu einem Empfänger (Ende-zu-Ende). In der Automobilbranche wird die Ende-zu-Ende Latenz als asymmetrische und unidirektionale Verbindungen angesehen. Im Gegensatz dazu gibt es aber auch die bidirektionale Roundtrip-Latenz in Client-Server Anwendungen, die nicht näher betrachtet wird [30]. Für diese Arbeit wird der Begriff Latenz als Synonym für die Ende-zu-Ende Latenz verwendet.

Echtzeiterweiterung von Ethernet

Die folgende Erklärung zu Echtzeiterweiterung von Ethernet baut auf die Doktorarbeit von Steinbach [30] auf. Ethernet ist von der Institute of Electrical and Electronics Engineers (IEEE) in IEEE 802.3 [15] standardisiert. In konventionellen Netzwerken arbeitet der Datenverkehr in Ethernet mit der Nutzung von Transportprotokollen nach dem Best Effort (BE)-Prinzip. Außerdem erfüllt Ethernet keine Mechanismen wie erneutes Senden oder Übertragungsgarantien mit Bestätigungen. Ethernet-Datenpakete können weder während der Übertragung unterbrochen noch blockiert werden. Dennoch hat Ethernet eine universelle und flexible Eigenschaft. Ethernet setzt nicht den Fokus auf die Darstellung der Daten, sondern ausschließlich auf die Übertragung der Daten. Um Ethernet echtzeitfähig zu machen, muss das Aufstauen von Nachrichten verhindert werden. Dazu werden in diesem Kapitel die Kernmerkmale der relevanten Echtzeiterweiterung von Ethernet für diese Arbeit beschrieben. Für detailliertere Beschreibungen wird auf die Standards verwiesen.

IEEE 802.1Q-2018

Für den weiteren Verlauf der Arbeit wird IEEE 802.1Q-2018 als 802.1Q bezeichnet. Die folgende Erklärung zu 802.1Q baut auf die Doktorarbeit von Steinbach [30] auf. 802.1Q wurde von der IEEE in [13] standardisiert. Der Standard 802.1Q nutzt Ethernet [15], um die Echtzeitfähigkeit darin zu etablieren. Es wird in 802.1Q der Ethernet-Frame um vier Byte (802.1Q-Header) vergrößert, um Ethernet-Datenpakete zu priorisieren. In Abbildung 2.5 ist die Erweiterung des Ethernet-Frames mit dem 802.1-Header dargestellt. Der in Abbildung 2.6 dargestellte 802.1Q-Header ist in zwei Byte Tag Protocol Identifier

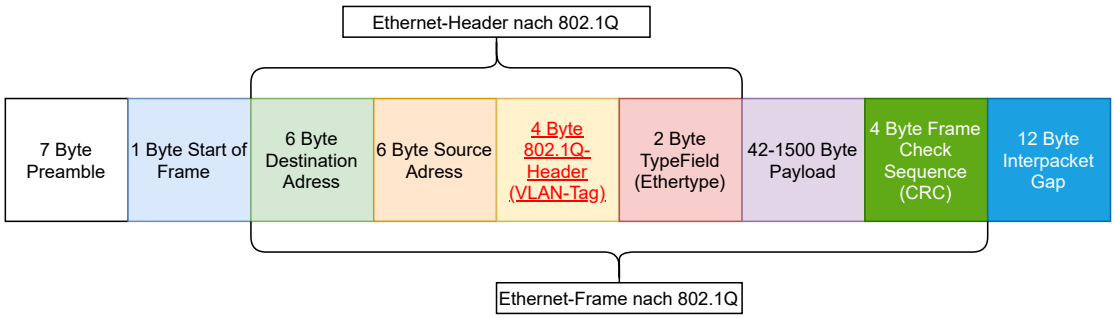

Abbildung 2.5: Der Aufbau eines Ethernet-Frames mit der Erweiterung 802.1Q-Header (Quelle: [30] Abbildung 2.6)

(TPI) und zwei Byte Tag Control Information (TCI) unterteilt. Die TCI ist wiederrum in

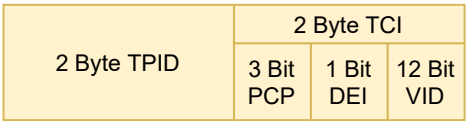

Abbildung 2.6: Der Inhalt vom 802.1Q-Header (Quelle: [30] Abbildung 2.6)

drei Bit Priority Code Point (PCP), ein Bit Drop Eligible Indicator (DEI) und zwölf Bit VLAN Identifier (VID) unterteilt. Das PCP-Feld erlaubt die Zuweisung von Ethernet-Datenpaketen zu einer Prioritätsklasse von 0 bis 7. Das DEI-Feld kann gesetzt werden, um Ethernet-Datenpakete im Fall eines Staues zu verwerfen. Mit dem VID-Feld wird das Erstellen von Virtual Local Area Networks (VLANs) ermöglicht. Dadurch kann einem Ethernet-Frame 4094 mögliche virtuelle Netzwerke zugewiesen werden [30]. Durch diese Erweiterung können Netzwerkkommunikation mit Echtzeitanforderungen umgesetzt werden. Netzwerkgeräte wie beispielsweise Switches sollten somit einen gewissen Grad an Dienstgüte zusichern. Die Ausprägung dieser Dienstgüte kann je nach Implementierung eines Switches unterschiedlich sein und unter anderem gewichtet, bandbreitenbasiert oder strikt prioritätsbasiert arbeiten. Um Prioritäten aus 802.1Q umsetzen zu können, wird ein Planer beziehungsweise Scheduler benötigt, der anhand der Prioritätsklasse und eines der beschriebenen Implementierungen Ethernet-Datenpakete versenden kann. Ethernet-Datenpakete mit derselben Priorität werden gleichbehandelt. Diese Erweiterung hat den Nachteil, dass Ethernet-Frames mit niedrigeren Prioritäten verzögert oder gar vollkommen verdrängt werden. Es werden nur noch Ethernet-Frames mit höheren Prioritäten bevorzugt. In dieser Arbeit werden Datenpakete von Hosts mit der Erweiterung des Ethernet-Frames verwendet und übertragen. Die in dieser Arbeit verwendeten Switches arbeiten strikt prioritätsbasiert.

IEEE 802.1Qbv-2015

Für den weiteren Verlauf der Arbeit wird IEEE 802.1Qbv-2015 als 802.1Qbv bezeichnet. Die folgende Erklärung zu 802.1Qbv baut auf die Doktorarbeit von Steinbach [30] auf. Der Standard 802.1Qbv ist eine Erweiterung und ein Bestandteil von 802.1Q. 802.1Qbv ist in [12] standardisiert. Mit 802.1Qbv wird eine Ausgangskontrolle realisiert. Die Ausgangskontrolle soll das Problem von Ethernet-Frames mit niedrigen Prioritäten lösen, die durch höher priorisierten Ethernet-Frames verdrängt werden können. Sie sieht vor, dass acht Warteschlangen synchronisiert und zeitgesteuert aktiviert oder deaktiviert werden.

Die acht Warteschlangen resultieren aus den acht Prioritäten des 802.1Q-Header aus Unterunterabschnitt 2.2. 802.1Qbv setzt eine zeitgesteuerte Kommunikation auf Basis von Nachrichtenklassen durch. Die Nachrichtenklassen können entweder aus zeitgesteuerte, aus bandbreitenbasierte oder aus BE-Nachrichtenklassen resultieren. Mit diesem Schedule ist es erlaubt, Zeitschlitze zu definieren, in denen nur bestimmte Nachrichtenklassen eine Sendeerlaubnis haben. Für die Ethernet-Frame-Auswahl, die in Abbildung 2.7 visualisiert ist, gibt es sieben Warteschlangen mit je einer Priorität von null bis sieben, von denen jeder den Transmission Selection Algorithm, den Transmission Gate und alle gemeinsam auf die Transmission Selection münden. In jeder Warteschlange kommen Ethernet-Frames

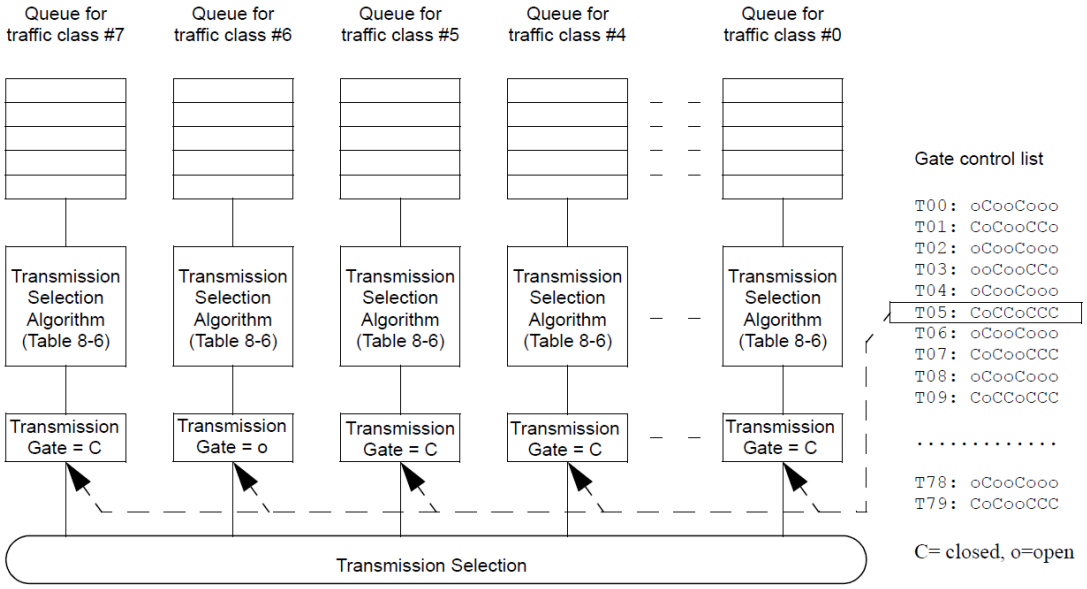

Abbildung 2.7: Die Ausgangskontrolle nach 802.1Qbv (Quelle: [13] Abbildung 8.14)

mit ihrer Priorität an. Ein Shaper pro Warteschlange wendet einen Übertragungsauswahlalgorithmus (Transmission Selection Algorithm) an. Nach 802.1Qbv [12] stehen vier Algorithmen zur Verfügung:

- Übertragungsauswahlalgorithmus mit Credit-based Shaper (CBS)
- Enhanced-Transmission-Selection Algorithmus (ETS-Algorithmus)
- Übertragungsauswahlalgorithmus mit strikter Priorität
- selbstentwickelter Übertragungsauswahlalgorithmus

Der Übertragungsauswahlalgorithmus mit strikter Priorität wird von allen Switches als Standardalgorithmus zur Auswahl von Frames für die Übertragung unterstützt. Der Übertragungsauswahlalgorithmus nach CBS und der ETS-Algorithmus können zusätzlich zum Übertragungsauswahlalgorithmus mit strikter Priorität unterstützt werden.

Nachdem ein priorisiertes Ethernet-Frame aus einer Warteschlange von einem Übertragungsauswahlalgorithmus ausgewählt wurde, landet dieses an das Transmission Gate. Der Zustand des Transmission-Gates bestimmt, ob in der Warteschlange bestehende priorisierte Ethernet-Frames für die Übertragung ausgewählt werden können oder nicht. Für eine bestimmte Warteschlange kann sich das Transmission Gate in einem von zwei Zuständen befinden:

- **OPEN**
  Die in der Warteschlange befindlichen priorisierten Ethernet-Frames werden für die Übertragung ausgewählt, entsprechend der Definition des mit der Warteschlange verbundenen Übertragungsauswahlalgorithmus'.
- **CLOSE**
  Priorisierte Ethernet-Frames, die in der Warteschlange sind, werden nicht für die Übertragung ausgewählt.

Dadurch kann eine Zeitsteuerung des Datenverkehrs erzielt werden. Der Zustand der Transmission Gates wird über die GCL verwaltet. Jede Gate-Operation ändert den Transmission Gate-Zustand für das jeweilige Gate, das jeder der Warteschlangen des Ports zugeordnet ist und ermöglicht die Planung zugehöriger Steuerungsoperationen. Wenn das Transmission Gate auf OPEN ist, wandert das Ethernet-Frame weiter zur Transmission Selection. In der Transmission Selection befindet sich ein Time-Aware-Shaper (TAS). Falls mehrere Transmission Gates auf OPEN sind, entscheidet der TAS welche Ethernet-Frames weitergeleitet und über den Port eines Switches ausgesendet wird. Wenn das Transmission Gate auf CLOSED ist und sich die Warteschlange auffüllt, werden beim Erreichen des maximalen Füllstandes Ethernet-Frames verworfen. In der GCL ist eine Liste von Einträgen vorhanden, die den Netzwerk-Zeitplan beinhalten. Ein Eintrag fasst die Zustände aller Transmission Gates und die Dauer dieses Zustandes zusammen. So lautet der Zustand CLOSED für alle Transmission Gates o o o o o o o o und der Zustand OPEN für alle Transmission Gates C C C C C C C C. Da 802.1Q acht Prioritätsklassen von null bis sieben gewährleistet, sind pro Port die Transmission Gates null bis sieben vorhanden. Mit diesem Mechanismus werden Zeitgarantien für Latenz im Mikrosekunden-Bereich gewährleistet, solange der Zeitschlitz getroffen wird.

Damit nicht-unterbrechbare und niedriger priorisierte Ethernet-Frames nicht von höher priorisierte Ethernet-Frames verzögert werden, wird das Guard Band (GB) oder Schutzband definiert. Dies wird in Abbildung 2.8 dargestellt. Der Schutzband ist in zeitgesteu-

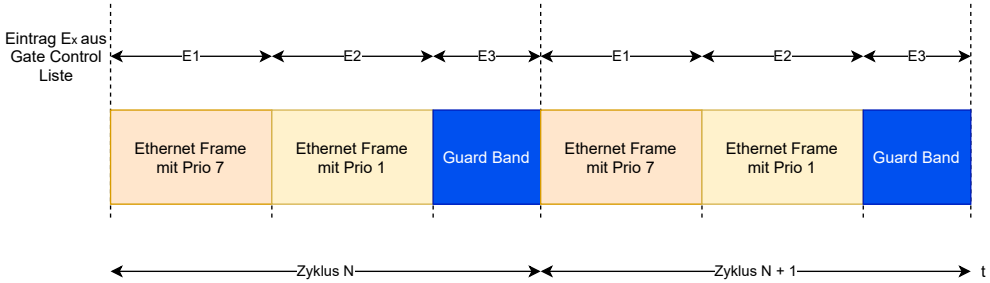

Abbildung 2.8: Der Schedule mit Schutzband (Vgl. Quelle: [2])

erten Echtzeit-Ethernet-Varianten ein Mechanismus, um zu verhindern, dass priorisierte Nachrichten durch eine asynchrone Nachricht verzögert wird. Innerhalb des Schutzbandes ist die Übertragung von asynchronen Nachrichten nicht erlaubt, da dieser in das zeitgesteuerte Sendefenster hineinreichen könnte [30]. In dieser Arbeit implementieren die Switches den 802.1Qbv-Standard. Für den weiteren Verlauf dieser Arbeit werden Switches, die den 802.1Qbv-Standard implementieren, als Synonym gegenüber Switches verwendet.

2.3 Network Configuration Protocol

Das Network Configuration Protocol (NETCONF) ist ein Protokoll für die Verwaltung und Konfiguration von Netzwerkgeräten. Es wurde von der Internet Engineering Task Force (IETF) in RFC 6241 spezifiziert [8]. Mittels Remote Procedure Call (RPC) werden Nachrichten über ein sicheres und verbindungsorientiertes Transportprotokoll wie das Transmission Control Protocol (TCP) oder Secure Shell (SSH) versendet, um Datenmodelle von Netzwerkgeräten abzurufen. Während der Übertragung sind alle Daten in Extensible Markup Language (XML) codiert. Dabei unterscheidet NETCONF zwischen Konfigurationsdaten und Zustandsdaten, die auf einem NETCONF-Server existieren. Ein NETCONF-Server kann ein Switch sein und führt Protokolloperationen aus, die von einem NETCONF-Client aufgerufen werden. Die Yet Another Next Generation (YANG)-Datenmodellierungssprache, die von der IETF in RFC 6020 spezifiziert [3] wurde, wird genutzt, um Zustandsdaten und Konfigurationsdaten für das NETCONF-Protokoll zu modellieren. Bei Zustandsdaten handelt es sich um Informationen, die von

einem NETCONF-Client abgerufen, jedoch nicht von diesem manipuliert werden können. Im Gegensatz dazu können Konfigurationsdaten von einem NETCONF-Server manipuliert werden. Die Manipulation von Konfigurationsdaten werden auf Konfigurationsdatenspeicher beziehungsweise configuration data store vollzogen. Konfigurationsdatenspeicher sind auf NETCONF-Server gehaftet und können in drei Arten unterteilt werden. In Abbildung 2.9 wird eine Übersicht von NETCONF-Server, NETCONF-Client, Konfigurationsdaten, die als YANG-Datenmodelle dargestellt sind und wie die Konfigurationsdaten auf den Konfigurationsdatenspeicher von NETCONF-Servern vorhanden sind. Die YANG-Datenmodelle sind auf einer Art von Konfigurationsdatenspeicher eingestellt.

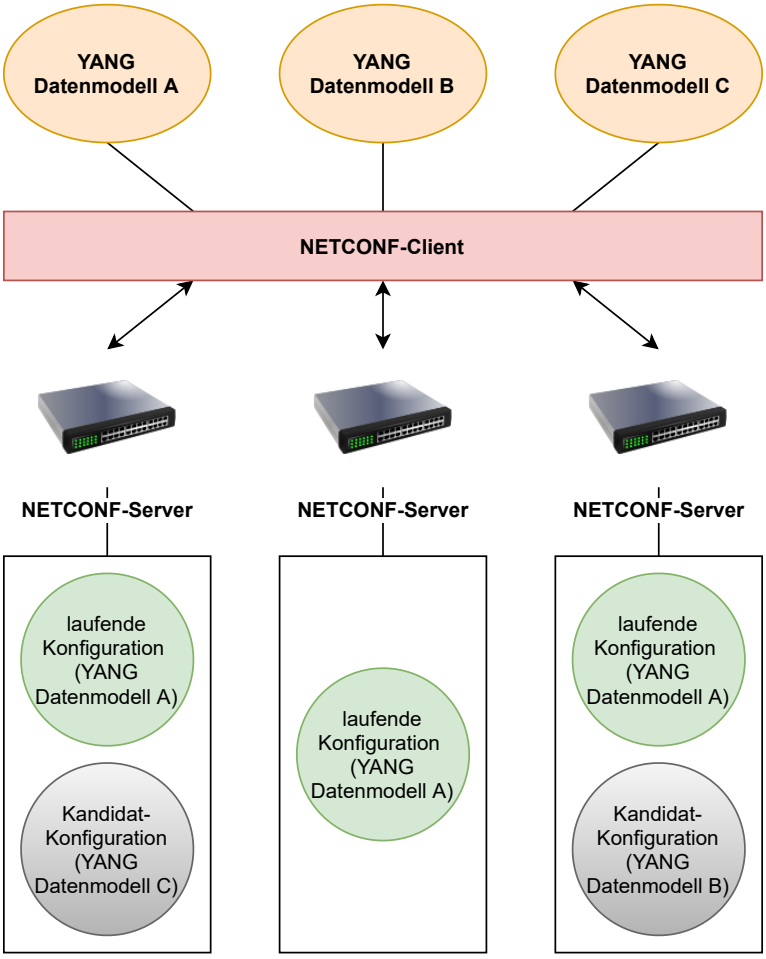

Abbildung 2.9: Der NETCONF-Server und der NETCONF-Client

Ein Konfigurationsdatenspeicher enthält den vollständigen Satz an Konfigurationsdaten, der erforderlich ist, um ein NETCONF-Server aus seinem anfänglichen Standardzustand

in einen gewünschten Betriebszustand zu bringen. Das in Abbildung 2.9 dargestellte YANG-Datenmodell A visualisiert das **running configuration datastore**. Das running configuration datastore ist ein Konfigurationsdatenspeicher, der die laufende Konfiguration auf dem NETCONF-Server enthält. Dieser ist immer auf einem NETCONF-Server vorhanden. Außerdem können zwei weitere Arten von Konfigurationsdatenspeicher auf einem NETCONF-Server vorhanden sein. Das in Abbildung 2.9 dargestellte YANG-Datenmodell B visualisiert das **candidate configuration datastore**. Das candidate configuration datastore ist ein Konfigurationsdatenspeicher, der manipuliert werden kann, ohne die laufende Konfiguration des NETCONF-Servers zu beeinflussen und der in das running configuration datastore übertragen werden kann. Im candidate configuration datastore befindet sich die Kandidat-Konfiguration. Die Kandidat-Konfiguration ist erst aktiv, wenn sie zur neuen laufenden Konfiguration eingestellt wird. Dieser Ansatz hat die Vorteile, dass Konfigurationssätze an einem NETCONF-Server übergeben und bei der Manipulation der running configuration datastore Inkonsistenzen vermieden werden. Ein weiterer Ansatz, um Inkonsistenzen zu vermeiden, ist das Sperren eines Datenspeichers und das Sichern des Datenspeichers mit der laufenden Konfiguration in einer Checkpoint-Datei. Dadurch wird der Zugriff von anderen Clients vermieden und kann zur Wiederherstellung einer Konfiguration bei Fehlersituationen, die aus NETCONF-Operationen resultieren können, verwendet werden. Des Weiteren können NETCONF-Server einen Konfigurationsdatenspeicher besitzen, der das running configuration datastore nach einem Neustart ersetzt und in Betrieb nimmt. Dies nennt man **startup configuration datastore**. Im startup configuration datastore befindet sich die Start-up-Konfiguration. Zur Realisierung dieser Datenspeicher müssen diese als Antwort nach einem Verbindungsaufbau zu einem NETCONF-Server zurückgegeben werden.

Eine NETCONF-Sitzung beginnt mit einem Handshake, indem der NETCONF-Server und der NETCONF-Client die von ihnen unterstützten NETCONF-Fähigkeiten (capabilities) angeben. Der NETCONF-Client sendet das hello-Tag-Element mit seinen Fähigkeiten und der NETCONF-Server antwortet mit seinem hello-Tag-Element, seiner Session-ID und seinen Fähigkeiten. Das NETCONF-Protokoll definiert eine Menge von Basisoperationen, die jedes NETCONF-fähige Netzwerkgerät unterstützen muss.

- **get**
  Diese Operation ruft den gegenwärtig laufenden Konfigurationsdatenspeicher und den Status eines Netzwerkgeräts ab.

- **get-config**
  Diese Operation ruft alle Informationen eines definierten Konfigurationsdatenspeichers ab.
- **edit-config**
  Diese Operation lädt einen Datensatz auf einen definierten Konfigurationsdatenspeicher.
- **copy-config**
  Diese Operation ersetzt den Inhalt eines definierten Konfigurationsdatenspeichers mit dem eines anderen.
- **delete-config**
  Diese Operation löscht einen definierten Konfigurationsdatenspeicher. Der gegenwärtig laufende Konfigurationsdatenspeicher ist hiervon ausgeschlossen.
- **lock**
  Diese Operation sperrt einen definierten Konfigurationsdatenspeicher, um den Zugriff anderer Clients zur Rekonfigurierung zu hindern.
- **unlock**
  Diese Operation entsperrt einen definierten Konfigurationsdatenspeicher.
- **close-session**
  Diese Operation schließt eine NETCONF-Sitzung und gibt alle mit der Sitzung verbundenen Sperren und Ressourcen frei und schließt alle zugehörigen Verbindungen. Eine NETCONF-Sitzung schließt nur dann, wenn alle Operationen ganz durchgeführt wurden.
- **kill-session**
  Diese Operation schließt eine NETCONF-Sitzung. Es bricht alle laufenden Vorgänge ab und gibt alle mit der Sitzung verbundenen Sperren und Ressourcen frei. Letztlich schließt es alle zugehörigen Verbindungen und Sitzungen.

NETCONF unterstützt mehrere verschiedene Transaktionsmodelle für Konfigurationsänderungen:

- **Direct model**
- **Candidate model**

- **Distinct startup model**

In Abbildung 2.10 ist zu erkennen, dass beim **Direct model** die Änderung am Datenspeicher der laufenden Konfiguration getätigt wird. NETCONF ermöglicht es, einen

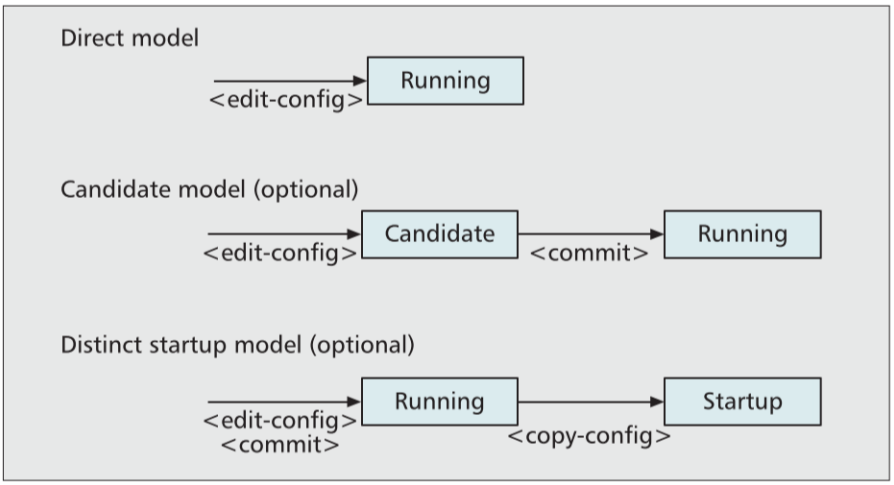

Abbildung 2.10: Die Transaktionsmodelle in NETCONF (Quelle: [29] Abbildung 4)

Satz von Konfigurationsänderungen oder eine vollständige neue Konfiguration in einer **edit-config**-Operation zu senden und einzuführen. Dadurch werden alle Konfigurationsänderungen direkt im Datenspeicher der laufenden Konfiguration übernommen. Dieses Transaktionmodell zeichnet sich als die einfachste Variante aus. Jede Operation, die ein NETCONF-Client an einen NETCONF-Server mittels der **edit-config**-Operation tätigt, wird sofort übernommen.

Mit Hilfe des **Candidate model** ist ein NETCONF-Client in der Lage, eine netzwerkweite Transaktion auf mehreren Netzwerkgeräten zu vollziehen. Wie bereits beschrieben wird hierfür ein weiterer Datenspeicher als Kandidat benötigt. Die Änderungen werden an den Kandidaten jedes teilnehmenden Netzwerkgerätes gesendet. Danach erfolgt eine Validierung des Kandidaten. Wenn alle Kandidaten alle Änderungen erfolgreich eingestellt haben, werden die teilnehmenden Netzwerkgeräte angewiesen, die Änderungen zu bestätigen. Falls jedoch ein Kandidat fehlschlägt, da Inkonsistenzen während der Konfiguration entstanden sind, werden allen teilnehmenden Netzwerkgeräte Bescheid gegeben, dass der Datenspeicher des Kandidaten nicht aktiviert werden soll. Das hat den Vorteil, dass ein Zurücksetzen (Rollback) gar nicht iniziiert wird, sondern dass der Kandidat eines jeden Netzwerkgerätes verworfen wird.

Mit dem **Distinct startup model** erlaubt NETCONF separate Datenspeicher, in denen jeweils die laufende und die Start-up-Konfiguration vorhanden ist. Die Start-up-Konfiguration wird von dem Netzwerkgerät geladen, wenn dieser neugestartet wird. Jegliche Vorgänge, die den Datenspeicher der laufenden Konfiguration betreffen, werden nicht in den Datenspeicher der Start-up-Konfiguration kopiert. Es wird eine explizite **copy-config**-Operation von der laufenden Konfiguration zur Start-up-Konfiguration durchgeführt, um die Start-up-Konfiguration auf den aktuellen Inhalt der laufenden Konfiguration zu aktualisieren.

Das zeitsynchrone Transaktionsmodell dieser Arbeit wird nach der Idee des **Candidate model** entwickelt. Mit dem Candidate model können Transaktionen realisiert werden, die mehrere Änderungen als ein zusammengefasster Arbeitsschritt an den Netzwerkgeräten eingeführt wird. Die Durchführung der Änderungen an den Konfigurationsdatenspeichern wird mit NETCONF getätigt.

2.4 Software-defined Networking

Die folgende Erklärung zu Software-defined Networking (SDN) basiert auf die Arbeit von Kreutz u.a. [17]. In konventionellen Netzwerken, wie in Abbildung 2.11 (oben) dargestellt, sind Netzwerkgeräte wie Switches, Routern oder Firewalls neben dem Weiterleiten von Datenpaketen auch für das Verarbeiten der Datenpaketen nach definierten Richtlinien zuständig. Die Netzwerkgeräte bestehen aus der zusammengesetzten Logik von Kontroll-, Daten- und Managementschicht. Die Kontrollschicht beziehungsweise Control Plane steuert das Weiterleitungsverhalten von Netzwerkgeräten und den Netzwerkverkehr in der Datenschicht. Die Datenschicht beziehungsweise Data Plane bildet das Netzwerkverkehr und ist für das Weiterleiten von Datenpaketen zuständig. In der Managementschicht beziehungsweise Management Plane wird das Netzwerk verwaltet und überwacht. Da sich die Aufgabenbereiche von Management- und Kontrollschicht überschneiden, wird die Managementschicht als Teil der Kontrollschicht angesehen. Die zusammengesetzte Logik dieser Netzwerkgeräte erschwert das Anpassen definierter Richtlinien oder das Rekonfigurieren nach Ausfällen und Änderungen in größeren Netzwerken [17]. In SDN wird, wie in Abbildung 2.11 (unten) dargestellt, die Kontrollschicht aller Netzwerkgeräte extrahiert und in den SDN-Controller zusammengefasst. Netzwerkgeräte wie Switches oder Routern werden zu Weiterleitungsgeräte. Die Weiterleitungsgeräte leiten Datenpakete in einer Richtung, die von der Kontrollschicht angegeben wurde. Der SDN-Controller

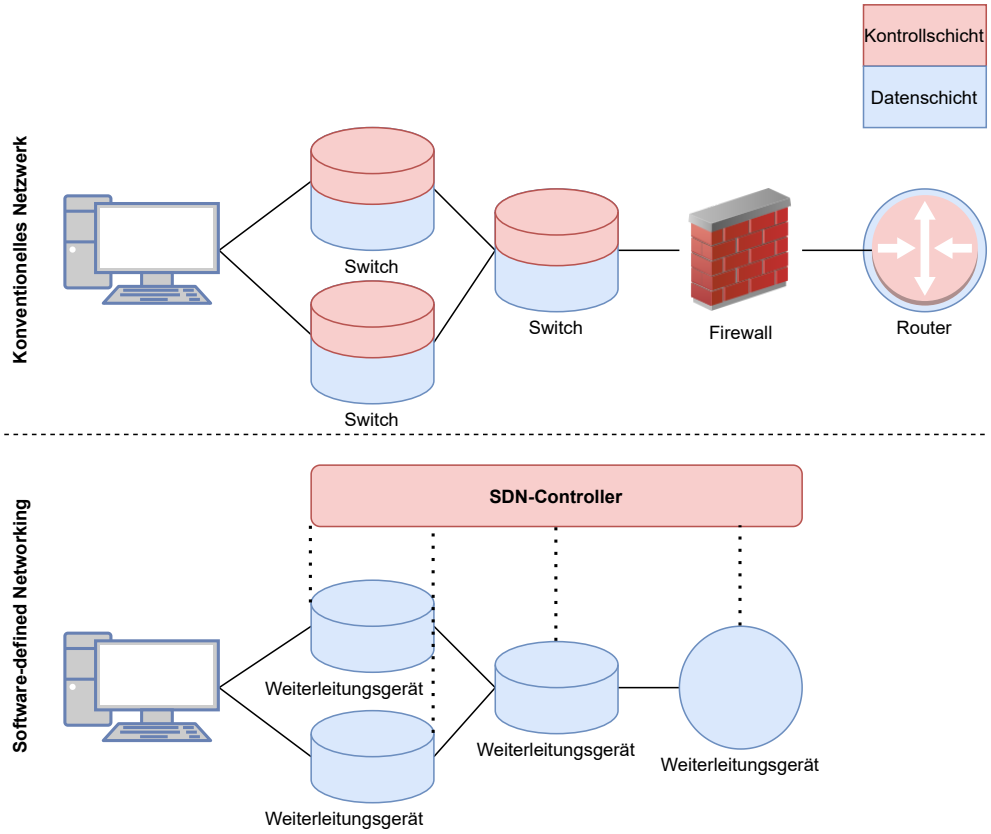

Abbildung 2.11: Konventionelles Netzwerk und SDN (Vgl. Quelle: [17] Abbildung 5)

hingegen besitzt eine zentralisierte Sicht über das Netzwerk und kann Entscheidungen darüber treffen, über welche Switches und deren Ports die Datenpakete weitergeleitet werden sollen. Die Kommunikation von SDN-Controller und Weiterleitungsgeräte wird durch das OpenFlow-Protokoll [1] ermöglicht. Durch die Separierung von Kontroll- und Datenschicht sind SDNs flexibler als konventionelle Netzwerke. Außerdem vereinfachen SDNs die Verwaltung und Skalierung und ermöglichen Programmieroptionen in diesen Netzwerken.

OpenFlow

Das OpenFlow-Protokoll ist ein Kommunikationsprotokoll, welches das Zusammenspiel von SDN-Controller und Weiterleitungsgeräte in SDN-Netzwerken ermöglicht. Es wird von der Open Network Foundation (ONF) [22] in [1] standardisiert. Mit dem OpenFlow-Protokoll wird ermöglicht, dass das Weiterleitungsverhalten von Weiterleitungsgeräten

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Tabelle 2.1: Die Felder eines Flusseintrages in einer Flusstabelle
(Quelle:[1] Tabelle 1)

gesteuert und überwacht wird. Hierbei fungieren Weiterleitungsgeräte als Server und der SDN-Controller als Client. In SDN-Netzwerken werden Datenpakete über Flüsse (flows), die zwischen Weiterleitungsgeräte definiert sind, versendet. Ein Weiterleitungsgerät (Switch) enthält mindestens eine Flusstabelle (flow table). In einer Flusstabelle werden Flusseinträge (flow entries) getätigt. Bei ankommenden Datenpaketen wird ein Matching durchgeführt, um die Zugehörigkeit des anzukommenden Datenpaketes mit einem Flusseintrag zu überprüfen. Bei einem Datenpaket, der nicht in der Flusstabelle eingetragen ist, wird dieser entweder verworfen oder zum SDN-Controller weitergeleitet. Der Aufbau eines Flusseintrages in einer Flusstabelle wird in Tabelle 2.1 dargestellt. Das Match Fields-Feld wird zum Abgleichen von ankommenden Datenpaketen verwendet. Es besteht aus dem Eingangsport und aus Paketheadern. Im Header eines Datenpaketes kann entweder die Ethernet-Quelladresse oder die IPv4-Zieladresse stehen. Das Prioritätsfeld Priority zeichnet die Priorität aus. Diese kann von null bis sieben sein. Die Priorität null wird als tabellenloser Flusseintrag bezeichnet. Das Match Fields-Feld und das Prioritätsfeld identifizieren gemeinsam einen eindeutigen Flusseintrag in einer bestimmten Flusstabelle. Das Zählerfeld counters wird immer bei abgeglichenen Datenpaketen aufgezählt. Im Instructions-Feld werden Befehle bei einem Datenpaketeingang an einen Fluss getätigt. Dabei können es sich um Befehle handeln, die den Inhalt eines Datenpaketes modifizieren oder das Datenpaket im Weiterleitungsgerät verarbeiten. Im Timeouts-Feld wird die maximale Zeitspanne angegeben, wie lange ein Flusseintrag ungenutzt sein darf, bevor es dann verworfen wird. Das Cookie-Feld wird vom SDN-Controller verwendet, um Flusseinträge nach Statistiken, Modifikationen oder Löschanfragen zu filtern. Mit dem Flags-Feld wird die Art der Verwaltung von Flusseinträgen ermöglicht. Für weitere Details hierzu wird auf [1] verwiesen.

Die Kommunikation zwischen Weiterleitungsgeräte und SDN-Controller beginnt damit, dass zunächst die Weiterleitungsgeräte sich beim SDN-Controller melden. Der SDN-Controller empfängt die Meldungen der Weiterleitungsgeräte und fragt nach den unterstützten Features der Weiterleitungsgeräte. Nachdem jedes Weiterleitungsgerät die Features dem SDN-Controller kenntlich gemacht hat, werden die Flusskonfigurationen an die Weiterleitungsgeräte mitgeteilt. In Abbildung 2.12 ist der Ablauf zwischen dem

SDN-Controller und den Weiterleitungsgeräten visualisiert, in dem ein erstes Datenpaket übertragen wird. Host A leitet das Datenpaket zu Weiterleitungsgerät 1. Danach prüft

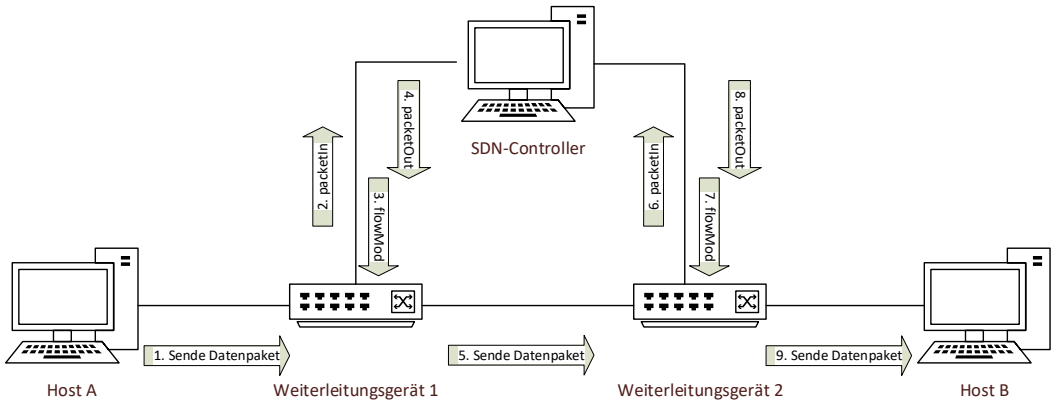

Abbildung 2.12: Die Verarbeitung von Datenpaketen in OpenFlow

Weiterleitungsgerät 1, ob ein Fluss für solch ein Datenpaket in der Flusstabelle eingetragen ist. Wenn kein Flusseintrag vorhanden ist, wird der SDN-Controller gefragt. Der SDN-Controller ermittelt eine passende Regel zur Weiterleitung des Datenpaketes und teilt dies Weiterleitungsgerät 1 mit. Weiterleitungsgerät 1 empfängt die ermittelte Regel vom SDN-Controller und trägt diese in die Flusstabelle ein. Nun kann das Datenpaket weitergeleitet werden. Jedes Weiterleitungsgerät muss sich beim SDN-Controller melden, wenn ein neues Datenpaket das erste Mal eintrifft. In SDN-Netzwerken, in denen eine sehr hohe Anzahl von Weiterleitungsgeräten vorhanden sein könnte, kann dies passieren, wodurch das erste Datenpaket eines Flusses verzögert werden kann. Falls jedoch der Flusseintrag in allen Weiterleitungsgeräten vorhanden ist, wird das jeweilige Datenpaket weitergeleitet, ohne das der SDN-Controller involviert wird.

Die OpenFlow-Switch-Spezifikation definiert eine Menge von Nachrichten. Es werden nun die relevanten OpenFlow-Nachrichten für diese Arbeit benannt:

- **flowMod**
  Mit dieser Nachricht kann der SDN-Controller die Flusstabellen von Weiterleitungsgeräte modifizieren.

- **packetIn**
  Mit dieser Nachricht benachrichtigt ein Weiterleitungsgerät dem SDN-Controller, dass eine Aktion ausgeführt wurde.

- **packetOut**
  Mit dieser Nachricht benachrichtigt der SDN-Controller dem Weiterleitungsgerät, welche Aktionen dieser durchführen muss.

2.5 Simulationsumgebung

Die folgende Erklärung zur Simulationsumgebung OMNeT++ basiert auf das Simulationshandbuch von OMNeT++ [21]. OMNeT++ ist die Abkürzung für Objective Modular Network Testbed. Es ist ein erweiterbares, modulares und Komponenten-basiertes Framework zum Erstellen von Netzwerksimulationen. OMNeT++ ist event-basiert und ermöglicht das Erstellen von Modulen mit einer Network Description Language (NED). Das Verhalten dieser Module wird in C++ programmiert. Ein Modul kann zum Beispiel ein Port, eine Anwendung oder ein Buffer sein. Aus diesen Modulen lassen sich beliebige Netzwerkkomponenten wie zum Beispiel Switches, Sensoren oder Steuergeräte zusammensetzen. Dadurch können beliebige Netzwerke erstellt und simuliert werden. OMNeT++ eignet sich daher zum Untersuchen verschiedener Netzwerkprotokolle.

In OMNeT++ ist ein Modell aus Modulen aufgebaut, die über Nachrichten miteinander kommunizieren. Es wird zwischen Simple Modules und Compound Modules unterschieden. In einem Simple Module wird das Verhalten eines Modells mit der Programmiersprache C++ implementiert. Eine Menge von Simple Modules bildet ein Compound Module. Die Anzahl der Hierarchieebenen ist unbegrenzt. Module (Simple Modules und Compound Modules) sind über Gates miteinander verbunden. Eine Verbindung wird als Connection bezeichnet. Fasst man alle Punkte zusammen, ergibt sich daraus ein Netzwerk, welches als Modell in OMNeT++ angesehen ist. Abbildung 2.13 visualsiert dies. Ein Netzwerk selbst ist wiederum ein Compound Module und kann weiter vertieft werden.

Für das Transaktionsmodell dieser Arbeit werden die aus Abbildung 2.14 beschriebenen Frameworks verwendet. Das **INET**-Framework [23] setzt Standard Ethernet- und Internettechnologien um. Das **CoRE4INET**-Framework [5] wird von der CoRE-Arbeitsgruppe [4] entwickelt. Es baut auf das INET-Framework auf und enthält Implementierung von Echtzeit-Ethernet. Diese werden dann in OMNeT++ simuliert und evaluiert. Dieses **OpenFlow**-Framework [16], welches ursprünglich an der Universität Würzburg entwickelt wurde, enthält eine Implementation von SDN mit OpenFlow. Es stellt die OpenFlow-Standardnachrichtentypen, Implementierungen von Weiterleitungsgeräte, eine SDN-Controller-

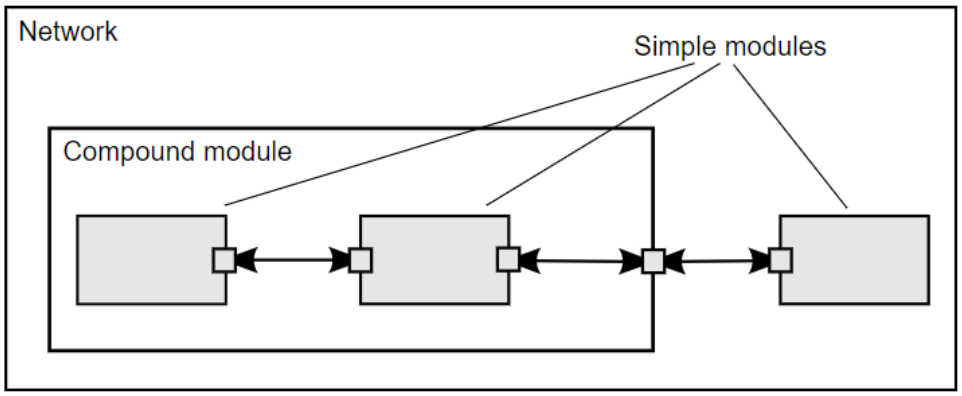

Abbildung 2.13: Das zusammengesetzte Netzwerk (Quelle: [21])

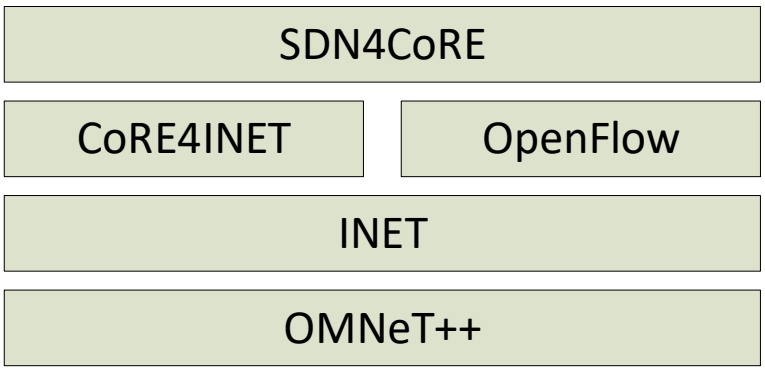

Abbildung 2.14: Die erweiterte Simulationsumgebung OMNeT++ mit den Frameworks INET, CoRE4INET, OpenFlow und SDN4CoRE

Implementierung und Schnittstellen für SDN-Controller-Anwendungen zur Verfügung. Das **SDN4CoRE**-Framework [10] ist ein Simulationsframework zur Kommunikation von Echtzeit-Ethernet mittels SDN. Es basiert auf die Frameworks INET und CoRE4INET. Außerdem nutzt es das OpenFlow-Framework und bietet zusätzliche Programmiermechanismen wie NETCONF, um die Programmierung von Echtzeit-Ethernet-Komponenten über Controller-Anwendungen zu ermöglichen.

3 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten aufgegriffen, die sich mit Transaktionen in SDN befasst haben. Außerdem werden auf gängige Ansätze der Netzwerkrekonfiguration in TSN eingegangen. Zum Schluss werden Arbeiten von SDN und TSN aufgegriffen.

3.1 Transaction-based Flow Rule Conflict Detection and Resolution in SDN [6]

In der Arbeit von Cui u.a. [6] wurde ein Mechanismus zur Erkennung und Auflösung von Konflikten zwischen Flussregeln auf Basis von Transaktionen entwickelt. Neben der Konfliktlösung dient dieser Mechanismus zur Erhaltung der Korrektheit beziehungsweise der Konsistenz des Netzwerkes. In der Arbeit beschreiben Cui u.a. [6], dass das Implementieren mehrere Flussregeln einer Netzwerkfunktion und das Entfernen einer Flussregel aus dem Befehlssatz zu einer Inkonsistenz im Netzwerk führen kann. Daher haben Cui u.a. [6] überlegt, dass der Befehlssatz der Flussregeln als eine atomare Operation angesehen werden soll. Es bedeutet, dass entweder alle oder keine Flussregeln wirksam werden. Darüber hinaus dürfen sich Flussregeln für verschiedene Netzwerkfunktionen nicht gegenseitig beeinträchtigen und müssen isoliert voneinander ablaufen. Mit dieser erworbenen Erkenntnis definierten Cui u.a. [6] die Flussregeltransaktion für einen Satz von Flussregeln, die für eine bestimmte Netzwerkfunktion vorgesehen ist.

3.2 Transactional Network Updates in SDN [7]

Aus der Arbeit von Cui u.a. [6] ist zu erkennen, dass die Transaktionseigenschaften Atomizität und Isolation für ihre Problemlösung genutzt wurden. Dieselben Transaktionseigenschaften haben Curic u.a. [7] zur Netzwerkaktualiserung in einer SDN-Umgebung

verwendet. Damit untersuchten Curic u.a. [7] verschiedene Aktualisierungstypen und Aktualiserungsraten. In ihrer Arbeit definierten Curic u.a. [7] einen Transaktionsmanager, der auf dem SDN-Controller als SDN-Anwendung läuft. Auf den Weiterleitungsgeräten definierten Curic u.a. [7] einen Ressourcenmanager. Der Transaktionsmanager koordiniert die Transaktion über das Two-Phase-Commit-Protokoll, um die globale Atomizität zu gewährleisten. Der Ressourcenmanager kümmert sich um die nebenläufigen Änderungsanforderungen, die es vom Transaktionsmanager erhält. Die Kommunikation zwischen den beiden Komponenten wird durch eine Erweiterung der südlichen Schnittstelle realisiert. Sie wird transactional Southbound Interface genannt. Curic u.a. [7] haben für die transaktionale Netzwerkaktualisierung vorhandene OpenFlow-Nachrichten erweitert, um ein Transaktionsmodell darin zu realisieren. Außerdem geben sie ein Beispiel für die Implementierung des Transaktionsmodells in SDN wieder. Aus der Arbeit von Curic u.a. [7] konnte nur die Netzwerkaktualiserung zwischen einem SDN-Controller und einem Weiterleitungsgerät entnommen werden.

Aus den Arbeiten von Cui u.a. [6] und Curic u.a. [7] kann entnommen werden, dass das Konzept der Transaktion in programmierbaren Netzwerken anwendbar ist. Jedoch betrachten diese Arbeiten nicht priorisierten Datenverkehr in den programmierbaren Netzwerken. Außerdem implementieren die Weiterleitungsgeräte aus diesen Arbeiten keine TSN-Standards wie 802.1Q (in Unterunterabschnitt 2.2) oder 802.1Qbv (in Unterunterabschnitt 2.2). In dieser Arbeit wird das Konzept der Transaktion verwendet, um Netzwerkrekonfigurationen an Weiterleitungsgeräte beziehungsweise Switches, die 802.1Qbv implementieren, durchzuführen.

3.3 Scheduled Bundles [1]

Scheduled Bundles sind eine Erweiterung des Bundle-Konzeptes und von der ONF in OpenFlow [1] spezifiziert. Bundles sind eine Folge von OpenFlow-Befehle zur Modifikation, welche vom SDN-Controller iniziiert werden. Ein Bundle wird als eine zusammengefasste OpenFlow-Operation angesehen. Der wesentliche Unterschied von Scheduled Bundles ist, dass alle Befehle eines Bundles zu einer vorher festgelegten Zeit ausgeführt werden. Dies ist in Abbildung 3.1 dargstellt. In Schritt 1 sendet der Controller eine Bundle Open-Nachricht an den Switch. Danach folgen mehrere Add-Nachrichten (Schritt 2). Jede Add-Nachricht kapselt eine OpenFlow-Nachricht ein, die beispielsweise eine **flowMod**-Nachricht sein kann. Danch wird in Schritt 3 eine Bundle Close-Nachricht gesen-

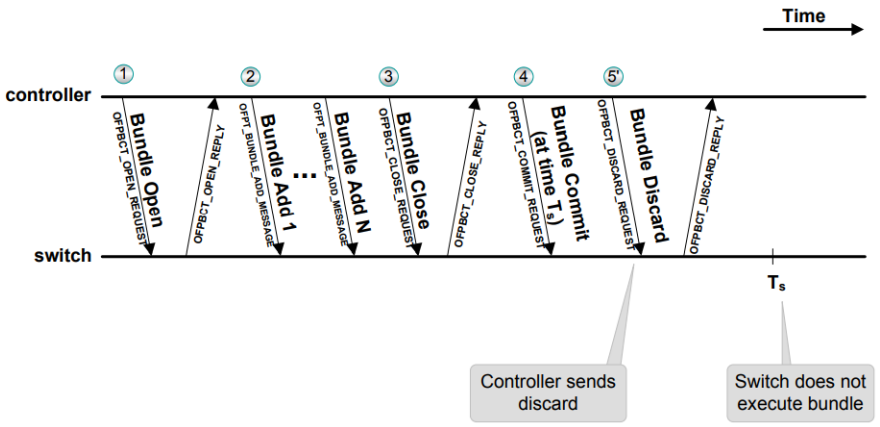

Abbildung 3.1: Das Verwerfen eines geplanten Commits
(Quelle: [1] Abbildung 7)

det. Nach der Close-Nachricht folgt das Bundle Commit (Schritt 4). Das Bundle Commit kann eine geplante Ausführungszeit T_s enthalten. Der Switch führt dann die gewünschten Befehle zum Zeitpunkt T_s aus. Wenn ein Weiterleitungsgerät nach dem Senden der Bundle-Commit-Nachricht eine Fehlermeldung erhält, sendet der SDN-Controller eine Discard-Nachricht (Schritt 5) an alle Weiterleitungsgeräte, um den geplanten Vorgang abzubrechen. Dieser Mechanismus kann für die Implementierung einer koordinierten Aktualisierung verwendet werden, bei der entweder alle Weiterleitungsgeräte den Vorgang erfolgreich einplanen oder das Bundle verwerfen ([1] Kapitel 6.9.6.2). Das Konzept der Bundles beziehungsweise Scheduled Bundles von OpenFlow eignet sich für diese Arbeit nicht, da die Abarbeitung von Befehlssätzen und die Umstellbarkeit nicht statisch sind und in einem zeitkritischen Netzwerk nicht den Zeitanforderungen entspricht.

3.4 Self-configuration of IEEE 802.1 TSN networks [9]

Gutiérrez u.a. [9] befassen sich in ihrer Arbeit mit der Selbstkonfiguration von TSN-Komponenten in einem zeitkritischen Netzwerk. In ihrer Arbeit führen sie einen Konfigurationsagenten ein, der das zeitkritische Netzwerk überwacht, Änderungen erkennt und Konfigurationen aktualisiert. Für die Konfiguration von TSN-Komponenten arbeiten Gutiérrez u.a. [9] nach IEEE 802.1Qcc-2018 [14]. Aus IEEE 802.1Qcc-2018 entnommen repräsentieren drei Modelle und zu jedem Modell jeweils einen Ansatz, wie die Konfiguration in einem zeitkritischen Netzwerk gehandhabt werden kann. In ihrer Arbeit ver-

wendeten Gutiérrez u.a. [9] das vollständig zentralisierte Konfigurationmodell aus IEEE 802.1Qcc-2018 sowie NETCONF als Konfigurationsprotokoll, um die Selbstkonfiguration zu realisieren. Die Selbstkonfiguration kann auch an zeitkritische Switches vollzogen werden, die nach dem 802.1Qbv-Standard (Unterunterabschnitt 2.2) implementiert wurden. Dafür müssen diese zeitkritischen Switches die Rekonfigurierungsfähigkeit beherrschen [9] (Abschnitt 4). In der Arbeit von Gutiérrez u.a. [9] wird der Fokus auf die Konfiguration von zeitkritischen Netzwerken gesetzt, jedoch richtet sie sich nicht auf programmierbare Netzwerke, wie es durch SDN ermöglicht wird.

3.5 SDN und TSN

In der Arbeit von Häckel u.a. [10] wird eine Programmieroption für zeitkritische Netzwerkgeräte, die priorisierten Datenverkehr verarbeiten können, eingeführt. Mit dieser Programmieroption soll die Belastbarkeit, die Sicherheit von fremden Zugriff und die Anpassungsfähigkeit der Umgebung verbessert werden. Hierbei stellen Häckel u.a. [10] Änderungen auf den drei Ebenen des SDN-Konzeptes. Zum Einen müssen die Weiterleitungsgeräte eine Programmierschnittstelle unter der Verwendung offener Standards bereitstellen. Zum Anderen muss die Logik, die für die Echtzeitfähigkeit in einem Switch verankert ist, extrahiert und in den SDN-Controller integriert werden. Letztlich müssen SDN-Anwendungen in der Lage sein Echtzeitgeräte zu programmieren und zu verwalten. Durch die Programmieroption entstehen eine Reihe von Vorteilen. Im Falle einer defekten Netzwerkleitung wird durch SDN das dynamische Umleiten von Netzwerkflüssen ermöglicht, die die Sicherheit von Kommunikationsnetzwerken in Fahrzeugen erhöht. Die Kombination von TSN und SDN können laut Häckel u.a. [10] ohne Leistungseinbußen genutzt werden.

Nayak u.a. [19] stellen eine Software-definierte Umgebung (SDE) für rekonfigurierbare Fertigungssysteme mit Echtzeiteigenschaften vor. Dabei setzen Nayak u.a. [19] den Fokus ihrer Arbeit auf die Robustheit und die Rekonfiguration durch die Verwendung exklusiver Links für Echtzeitverkehr. Außerdem entwickelten Nayak u.a. [20] ein zeitabhängiges softwaredefiniertes Netzwerk mit einem Scheduling-Prozess für zeitgesteuerten Verkehr. Der Zeitplan wird jedoch nicht in die Weiterleitungsgeräte programmiert, sondern die Hosts im Netzwerk kennen den vollständigen Zeitplan und senden Daten in ihren Zeitschlitzen. Die Weiterleitungsgeräte sind nicht echtzeitfähig und ungeplanter Querverkehr könnte das Netzwerkverhalten verändern [11].

4 Anforderungsanalyse

In diesem Kapitel werden auf die Anforderungen der CoRE RG eingegangen. Es folgt zunächst eine Beschreibung des zu entwickelnden zeitsynchronen Transaktionsmodells. Aus der Beschreibung werden dann die funktionalen und nicht-funktionalen Anforderungen definiert.

4.1 Beschreibung

Die CoRE RG fordert ein zeitsynchrones Transaktionsmodell an. Das zeitsynchrone Transaktionsmodell soll in das Framework SDN4CoRE [10] integriert und in der Simulationsumgebung OMNeT++ eingesetzt werden. Der Einsatz des zeitsynchronen Transaktionsmodells soll den Zweck erfüllen, dass in programmierbaren und zeitkritischen Netzwerken Rekonfigurationen auf transaktionaler Weise und unter zeitlichen Anforderungen durchgeführt werden. Für den weiteren Verlauf dieser Arbeit wird ein programmierbares und zeitkritisches Netzwerk als Synonym zu Netzwerk verwendet. Das zeitsynchrone Transaktionsmodell leitet zeitsynchrone Transaktionen ein. Die zeitsynchrone Transaktion muss die zeitlichen Anforderungen in einem Netzwerk berücksichtigen. So darf unter keinen Umständen aus der zeitsynchronen Transaktion resultieren, dass bestehender und hinzugefügter Datenverkehr in irgendeiner Weise beeinflusst werden. Datenpakete, die bereits gesendet werden oder hinzukommen, müssen ihren erwarteten Latenzen entsprechen. Auch dürfen keine Paketverluste an bestehendem und hinzugefügtem Datenverkehr nach der zeitsynchronen Transaktion entstehen. Die Änderungen an den Switches werden in einer XML-Datei definiert. Eine zeitsynchrone Transaktion ist folgendermaßen aufgebaut:

1. Sperroperation
   Es wird zu Beginn der zeitsynchronen Transaktion eine Sperroperation an die laufenden Konfigurationen aller angesprochenen Switches im Netzwerk mitgeteilt.

2. Kopieroperation
   Es folgt eine Kopieroperation an die gesperrten und laufenden Konfigurationen aller Switches im Netzwerk. Die Kopie der gesperrten laufenden Konfiguration lautet Kandidat-Konfiguration.

3. Sperroperation an Kandidat-Konfiguration
   Es wird eine erneute Sperroperation an die Kandidat-Konfiguration eines jeden Switches im Netzwerk mitgeteilt.

4. Änderungsoperationen
   Die durchzuführenden Änderungen sollen an die Kandidat-Konfiguration der Switches eingeführt werden. Bei einer Änderung kann es sich dabei um das Eintragen, das Löschen oder das Modifizieren von Flusseinträgen in die Flusstabelle eines Switches handeln oder um die Modifikation der Gate Control Liste auf einem Port in einem Switch. Änderungsoperationen beinhalten mindestens eine der benannten Änderungen.

5. Bestätigungsoperation
   Ein fest definierter Zeitpunkt muss ermittelt werden, in dem alle Switches gleichzeitig ihre geänderte Kandidat-Konfiguration zur neuen laufenden Konfiguration bestätigen. Es wird eine Bestätigungsanforderung mit diesem Zeitpunkt an alle Switches im Netzwerk mitgeteilt. Wenn alle Switches den Zeitpunkt der Bestätigungsanforderung wahrnehmen können, wird eine Freigabe zur Bestätigung zum Zeitpunkt an alle Switches erteilt.

6. Löschoperation
   Mit der Löschoperation wird aufgefordert, dass jeder Switch seine alte Konfiguration löschen soll.

7. Entsperroperation
   Die Entsperroperation entsperrt die neuen laufenden Konfiguration aller Switches.

Das zeitsynchrone Transaktionsmodell bricht die zeitsynchrone Transaktion bei den folgenden anwendungsseitigen Fehlern ab:

- Wenn die laufende Konfiguration eines Switches bereits gesperrt wurde.
- Wenn das Erstellen der Kandidat-Konfiguration fehlschlägt.
- Wenn eine Änderung an der Kandidat-Konfiguration fehlschlägt.

- Wenn ein Switch oder die zeitsynchrone Transaktion den fest definierten Zeitpunkt nicht einhalten kann, da diese diesen Zeitpunkt überschreiten werden.

In allen Fällen überführt das zeitsynchrone Transaktionsmodell das Netzwerk in den Zustand zurück, wie sie es vor der zeitsynchronen Transaktion hatte. Es darf immer nur eine zeitsynchrone Transaktion vom zeitsynchronen Transaktionsmodell durchgeführt werden. Eine zeitsynchrone Transaktion muss in allen Fällen immer terminieren und darf das Netzwerk oder die Switches nicht in einem inkonsistenten Zustand lassen. Das Netzwerk, in dem eine zeitsynchrone Transaktion durchgeführt wurde, bleibt so lange unverändert, bis eine neue zeitsynchrone Transaktion zu einem späteren Zeitpunkt eingeleitet wird.

In den nächsten Abschnitten werden die funktionalen und nicht-funktionalen Anforderungen des zeitsynchronen Transaktionsmodells aus der Beschreibung definiert.

4.2 Funktionale Anforderungen

Aus der Beschreibung in Abschnitt 4.1 werden folgende funktionale Anforderungen des zeitsynchronen Transaktionsmodells aufgestellt.

- **FA1 - Zeitsynchrone Transaktion**
  Das zeitsynchrone Transaktionsmodell muss zeitsynchrone Transaktionen ermöglichen. Wie aus der Beschreibung entnommen, sind in einer zeitsynchronen Transaktion die einzelnen Operationen *Sperroperation*, *Kopieroperation*, *Sperroperation an Kandidat-Konfiguration*, *Änderungsoperationen*, *Bestätigungsoperation*, *Löschoperation* und *Entsperroperation* zusammengefasst.

- **FA2 - Atomizität**
  FA1 muss atomar ausgeführt werden. Alle in einer zeitsynchronen Transaktion enthaltenen Operationen werden entweder ganz oder gar nicht ausgeführt.

- **FA3 - Konsistenz**
  FA1 muss das Netzwerk von einem validen Zustand in einen neuen validen Zustand überführen. Ein valider Zustand in einem Netzwerk ist, wenn bestehende übertragene Datenpakete in einem Netzwerk ihren erwarteten Latenzen entsprechen und ein minimaler Paketverlust herrscht.

- **FA4 - Zeitsynchronität**
  FA1 muss garantieren, dass die Switches in einem Netzwerk gleichzeitig und zeitsynchron zu einem fest definierten Zeitpunkt ihre Kandidat-Konfiguration als neue laufenden Konfiguration bestätigen.

4.3 Nicht-funktionale Anforderungen

Aus der Beschreibung in Abschnitt 4.1 werden folgende nicht-funktionale Anforderungen des zeitsynchronen Transaktionsmodells aufgestellt.

- **NFA1 - Fehlertoleranz**
  Eine zeitsynchrone Transaktion aus dem zeitsynchronen Transaktionsmodells kann mit anwendungsseitigen Fehlern umgehen und geeignete Maßnahmen einleiten, um das Netzwerk oder die Switches vor einer Inkonsistenz zu bewahren.

- **NFA2 - Verzögerung**
  Während und nach der zeitsynchronen Transaktion werden bestehender und hinzukommender Datenverkehr im Netzwerk nicht verzögert. Bei dieser nicht-funktionalen Anforderung soll beispielsweise das Ablegen von Datenpaketen in Warteschlangen vermieden werden.

- **NFA3 - Paketverlust**
  Während und nach der zeitsynchronen Transaktion darf kein Paketverlust bei bestehendem und hinzukommendem Datenverkehr entstehen.

5 Konzept

In diesem Kapitel wird das Konzept des zeitsynchronen Transaktionsmodells vorgestellt. Zunächst werden Annahmen an das zeitsynchrone Transaktionsmodell aufgestellt. Danach werden die einzelnen Phasen erläutert und ein Worst-Case-Zeitpunkt zur Commit-Ausführung errechnet. Außerdem wird auf die resultierende Finite State Machine (FSM) des zeitsynchronen Transaktionsmodells eingegangen. Letztlich werden vier Arten der Netzwerkrekonfiguration vorgestellt.

5.1 Annahmen

Folgende Annahmen werden an das zeitsynchrone Transaktionsmodell gestellt:

- Die Uhren aller Switches, die zum SDN-Controller verbunden sind, sind synchronisiert. Dadurch ist der Zeitpunkt der zeitsynchronen Transaktion auf allen Switches gleich. Das zeitsynchrone Transaktionsmodell ist nicht für das Synchronisieren der Uhren aller Switches verantwortlich.
- Der Transport der Nachrichten wird auf einem verbindungsorientierten Transportprotokoll durchgeführt. Das TCP ist ein verbindungsorientiertes Transportprotokoll und sichert mit seinem Verbindungsmanagement und seiner Segmentierung reihenfolgegesicherte und ankunftsgarantierte Nachrichten zu ([24]).
- Jegliche Fehler, die nach der Commit-Ausführung an den Switches geschehen sollten, entsprechen einem Systemfehler und werden vom zeitsynchronen Transaktionsmodell nicht behandelt.
- Es wird angenommen, dass die Switches den Standard 802.1Qbv aus Unterunterabschnitt 2.2 implementieren.

5.2 Zeitsynchrones Transaktionsmodell

Um das zeitsynchrone Transaktionsmodell zu realisieren, sind zunächst vier Phasen vorgesehen. Mit den vier Phasen sollen transaktionale Netzwerkrekonfigurationen ermöglicht werden. Für den Fall von anwendungsseitigen Fehlern wurde ein Mechanismus für das Zurücksetzen der zeitsynchronen Transaktion konzipiert. Die vier Phasen und das Zurücksetzen der zeitsynchronen Transaktion werden in den folgenden Abschnitten visualisiert und erklärt. In jeder Phase werden auch die anwendungsseitigen Fehler beschrieben, die beim Auftreten eines Fehlers vom zeitsynchronen Transaktionsmodell abgefangen werden. Jegliche Fehlerbehandlungen wie Hardware-Fehler, Nachrichtenverlust und einer unterbrochenen Netzwerkverbindung werden nicht vom zeitsynchronen Transaktionsmodell behandelt. Bei einem Verbindungsabsturz seitens TCP wurden zwei Möglichkeiten ermittelt, die als Wiederanlaufstrategie angesehen wurden. Zum einen könnte bei einer unterbrochenen TCP-Verbindung das zeitsynchrone Transaktionsmodell die gesamte zeitsynchrone Transaktion zurücksetzen. Das bedeutet, dass alle vorherigen Seiteneffekte von Operationen rückgängig gemacht werden und die Switches in den Eingangszustand vor der zeitsynchronen Transaktion zurückversetzen. Zum anderen könnte bei einer unterbrochenen TCP-Verbindung das zeitsynchrone Transaktionsmodell einen erneuten Aufbau der Verbindung nach einer vergangenen Zeit versuchen. Dadurch wird die zeitsynchrone Transaktion nicht unterbrochen, sondern zu einem späteren Zeitpunkt weiter ausgeführt. In dieser Arbeit wird ein vereinfachtes zeitsynchrones Transaktionsmodell erzielt, mit dem Fokus auf die zeitsynchrone Commit-Ausführung der neuen Konfigurationen auf den Switches. Daher wird der Ansatz verfolgt, dass bei einer unterbrochenen TCP-Verbindung in diesem Transaktionsmodell alle bisherigen Seiteneffekte zurückgesetzt werden.

Für die verwendeten Symbole und Zeichen in den folgenden Abschnitten dieses Kapitels wird auf die Legende in Abbildung 5.1 verwiesen.

5.2.1 Sperrphase

Die Sperrphase ist die erste Phase des zeitsynchronen Transaktionsmodells. Sie wurde aus der Idee des strikten Zwei-Phasen-Sperrprotokoll konzipiert. Auf das strikte Zwei-Phasen-Sperrprotokoll wird auf Unterunterabschnitt 2.1.2 eingegangen. In der Sperrphase werden die laufenden Konfigurationen von allen Switches, die mit dem SDN-Controller verbunden sind, für die zeitsynchrone Transaktion gesperrt. Der Grund für die Sperrung ist, dass keine weiteren Transaktionen auf den Switches durchgeführt werden können.

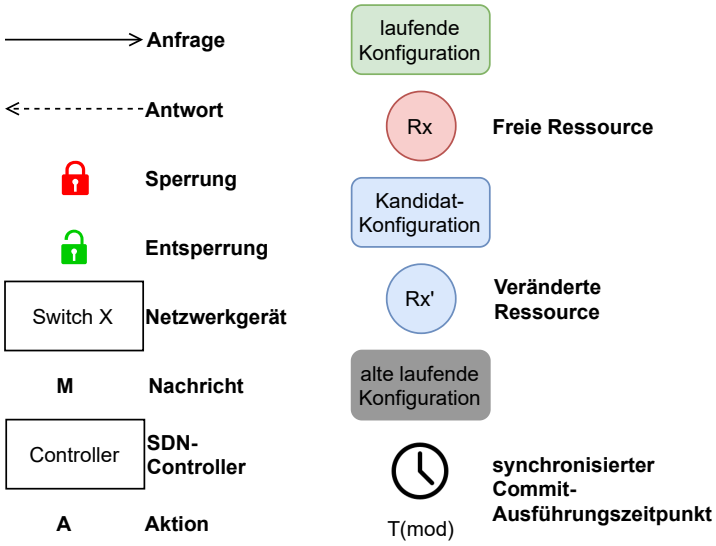

Abbildung 5.1: Die Legende für Phasen des Transaktionsmodells (Eigene Darstellung)

Außerdem erfolgt die Sperrung aller Switches mithilfe der Media Access Control (MAC)-Adresse. Dadurch wird eine Reihenfolge definiert, welche Switches zuerst und zuletzt angesprochen werden. Damit es zwischen Transaktionen zu keinem Deadlock kommt, gelten vier Bedingungen zur Deadlockvermeidung ([31] Kapitel 6.2.1). Für eine Transaktion muss ein Switch entweder zur Verfügung stehen oder bereits gesperrt sein. Darüber hinaus ist einer Transaktion erlaubt, weitere Switches zu sperren, wenn es zuvor bereits ein Switch gesperrt hat. Außerdem muss die Transaktion terminieren, um die Sperre zu lösen. Einer Transaktion ist es nicht erlaubt, gewaltsam die Sperre an einem Switch zu entziehen. Letztlich muss es eine zyklische Reihenfolge von Transaktionen geben, in der jeder auf einen Switch wartet, der die nächste Transaktion in der Kette gehört. Der Ablauf der Sperrphase ist in Abbildung 5.2, Abbildung 5.3, Abbildung 5.4 und Abbildung 5.5 visualisiert. Zur Übersicht wird die Sperrphase anhand von zwei Switches gezeigt.

In Abbildung 5.2 ist der Ausgangszustand zu sehen. Switch 1 und Switch 2 haben jeweils eine offene laufende Konfiguration und sind mit dem Controller verbunden.

In Abbildung 5.3 wird die Sperrung der laufenden Konfiguration von Switch 1 visualisiert. Zunächst definiert der Controller eine Reihenfolge für das Sperren der laufenden Konfigurationen der anzusprechenden Switches. Die Reihenfolge für das Sperren wird absteigend nach der MAC-Adresse definiert. Danach sendet der Controller die in M1 angeforderte Sperrung an Switch 1. Switch 1 empfängt die Sperranforderung vom Controller und

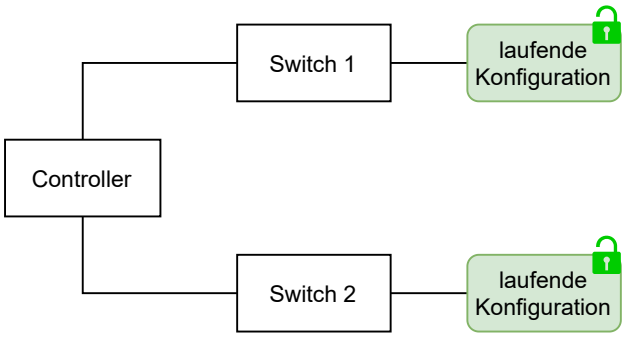

Abbildung 5.2: Der Ausgangszustand der Sperrphase (Eigene Darstellung)

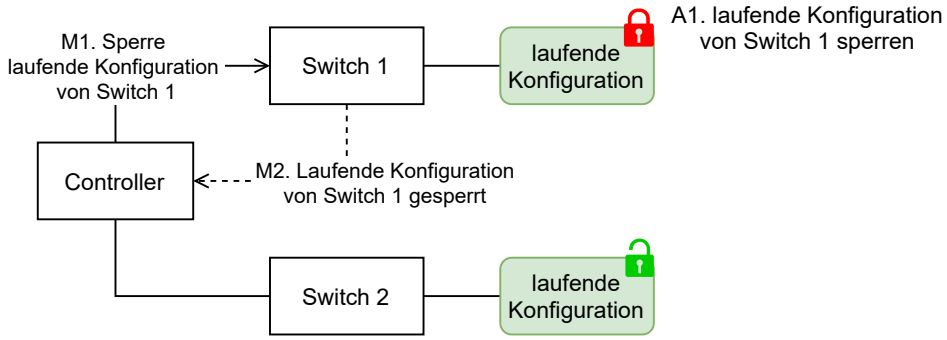

Abbildung 5.3: Das Sperren der laufenden Konfiguration von Switch 1 (Eigene Darstellung)

teilt dies seiner laufenden Konfiguration mit. Durch A1 sperrt Switch 1 seine laufende Konfiguration. Nachdem die laufende Konfiguration gesperrt wurde, teilt Switch 1 dies dem Controller mit der Bestätigungsnachricht M2 zurück. Die Sperrung der laufenden Konfiguration hat den Zweck, dass anderen Transaktionen die Sperrung einer bereits gesperrten laufenden Konfiguration eines Switches verweigert wird.

Zuerst wird auf den fehlerfreien Fall fokussiert. Nur wenn das Sperren der laufenden Konfiguration von Switch 1 erfolgreich war, wird die Sperranforderung der laufenden Konfiguration von Switch 2 eingeleitet. Dies wird in Abbildung 5.3 dargestellt. Der Controller sendet die in M3 angeforderte Sperrung an Switch 2. Switch 2 empfängt die Sperranforderung vom Controller und teilt auch dies seiner laufenden Konfiguration mit. Durch A2 sperrt Switch 2 seine laufende Konfiguration. Danach teilt Switch 2 dies dem Controller mit der Bestätigungsnachricht M4 zurück.

In Abbildung 5.5 ist das Resultat der Sperrphase visualisiert. Die laufenden Konfigura-

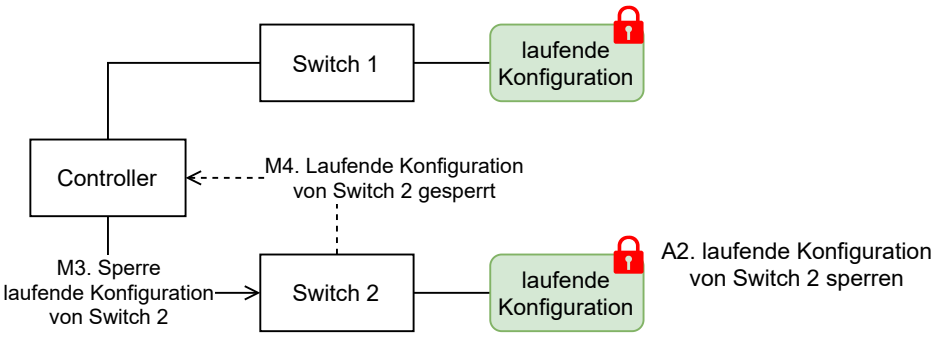

Abbildung 5.4: Das Sperren der laufenden Konfiguration von Switch 2 (Eigene Darstellung)

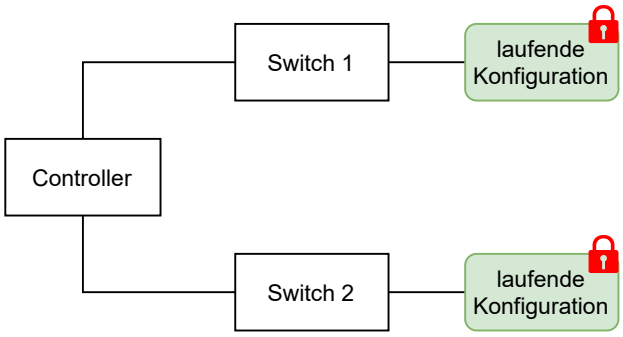

Abbildung 5.5: Das Resultat der Sperrphase (Eigene Darstellung)

tionen aller Switches wurden erfolgreich gesperrt und sind nun bereit für die Änderungsphase in Unterabschnitt 5.2.2.

Fehlerbehandlung in der Sperrphase

Falls während der Sperrphase die laufende Konfiguration eines Switches bereits gesperrt ist, gilt dies als Fehler. In diesem Fall werden die bisher gesperrten laufenden Konfigurationen aller Switches gelöst und somit die Transaktion beendet. Für wartende Transaktionen gilt, dass diese erst ausführbar sind, wenn die vorherige Transaktion terminiert. Da in dieser Phase keine Modifikation an einer laufenden Konfiguration an einem Switch getätigt wurde, wird die Entsperrphase aus Unterabschnitt 5.2.4 eingeleitet.

5.2.2 Änderungsphase

Die Änderungsphase ist die zweite Phase des zeitsynchronen Transaktionsmodells. Diese Phase tritt nur ein, wenn die laufenden Konfigurationen aller Switches erfolgreich gesperrt wurden. In der Änderungsphase wird zunächst die Kopie der laufenden Konfiguration pro Switch instanziiert und mit demselben Inhalt einer laufenden Konfiguration ausgestattet. Diese nennt man Kandidat-Konfiguration. Jede angeforderte Änderung wird nur an der Kandidat-Konfiguration durchgeführt und nebenläufig an alle Switches mitgeteilt. Der Ablauf der Änderungsphase ist in Abbildung 5.6, Abbildung 5.7, Abbildung 5.8, Abbildung 5.9 und Abbildung 5.10 visualisiert und an mehreren Switches veranschaulicht.

In Abbildung 5.6 ist der Ausgangszustand der Änderungsphase zu sehen. Jeder Switch

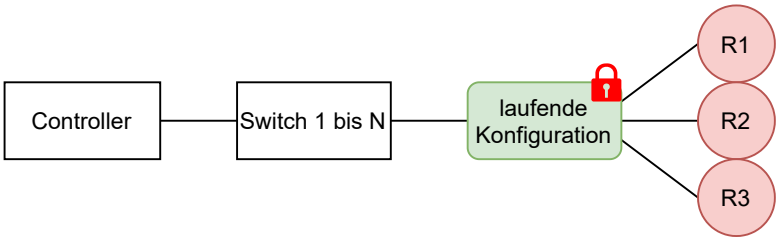

Abbildung 5.6: Der Ausgangszustand der Änderungsphase (Eigene Darstellung)

hat eine gesperrte laufende Konfiguration. Jede laufende Konfiguration besitzt mindestens eine Ressource. In Abbildung 5.6 hat jede laufende Konfiguration eines Switches drei Ressourcen (R1, R2 und R3). Eine Ressource entspricht eine Teilkonfiguration des jeweiligen Switches. Hierbei kann es sich bei einer Ressource um die Flusstabelle des Switches handeln oder um die Gate Control Liste eines Switches.

In Abbildung 5.7 wird die erste Änderung an der Kandidat-Konfiguration veranschaulicht. Der Controller teilt mit M5 allen Switches mit ihre laufenden Konfigurationen zu kopieren. Jeder angesprochene Switch empfängt die Nachricht M5 vom Controller und teilt dies seiner laufenden Konfiguration mit. Durch A3 erstellt der jeweilige Switch eine Kandidat-Konfiguration. Die Kandidat-Konfiguration wird mit demselben Inhalt der laufenden Konfiguration eines Switches ausgestattet (R1', R2' und R3'). Mit M6 teilt jeder Switch dem Controller zurück, dass die Kandidat-Konfiguration erfolgreich instanziiert wurde.

In Abbildung 5.8 wird, wie in der Sperrphase in Unterabschnitt 5.2.1, die Kandidat-Konfiguration eines jeden Switches gesperrt. Mit M7 wird vom Controller eine Sperranfor-

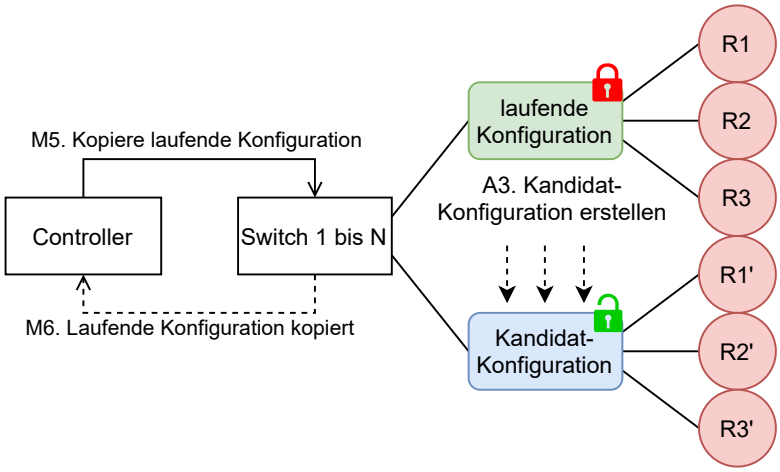

Abbildung 5.7: Das Kopieren der laufenden Konfiguration pro Switch (Eigene Darstellung)

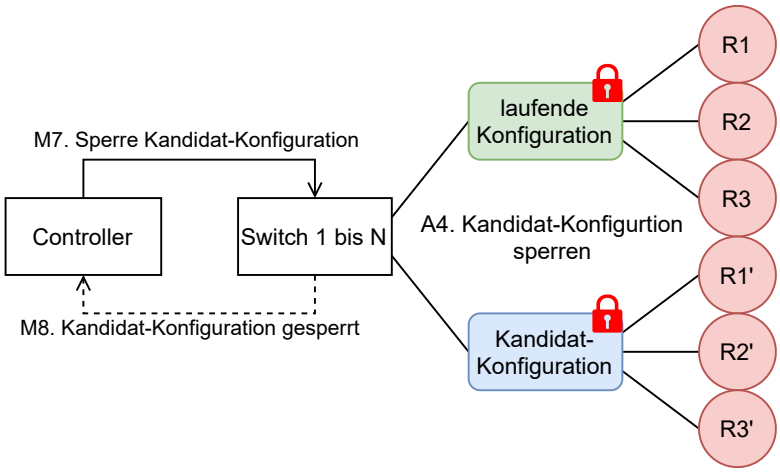

Abbildung 5.8: Die Sperrung der Kandidat-Konfiguration pro Switch (Eigene Darstellung)

derung an jedem Switch mitgeteilt. Mit A4 sperrt jeder Switch die Kandidat-Konfiguration. Danach wird mit M8 die Sperrung der Kandidat-Konfiguration eines Switches zurück an den Controller übermittelt. Die Sperranforderungen werden nebenläufig an alle Switches mitgeteilt.

In Abbildung 5.9 sendet der Controller die erste Änderungsnachricht M9 an alle Switches. Jeder Switch empfängt die Änderungsnachricht und führt mit A5 die Änderung an der Kandidat-Konfiguration des jeweiligen Switches aus. Die Ressource aus der Kandidat-

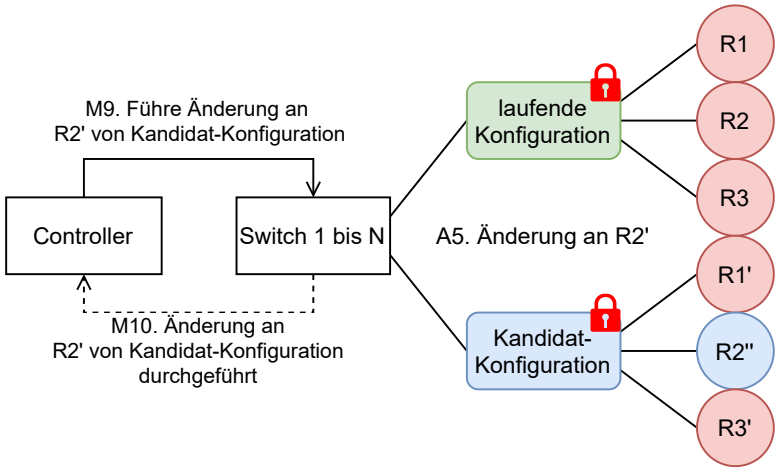

Abbildung 5.9: Die Durchführung von Änderungen an der Kandidat-Konfiguration pro Switch (Eigene Darstellung)

Konfiguration des jeweiligen Switches wird verändert (R2”). Nachdem die Änderung durchgeführt wurde, teilt der jeweilige Switch dies dem Controller mit M10 zurück. Dieser Teil der Änderungsphase wird so lange durchgeführt, bis alle durchzuführenden Änderungen erfolgreich an der Kandidat-Konfiguration vollzogen wurden. Die Änderung wird vom Controller an alle Switches nebenläufig mitgeteilt.

In Abbildung 5.10 ist das Resultat der Änderungsphase im erfolgreichen Fall. Jede

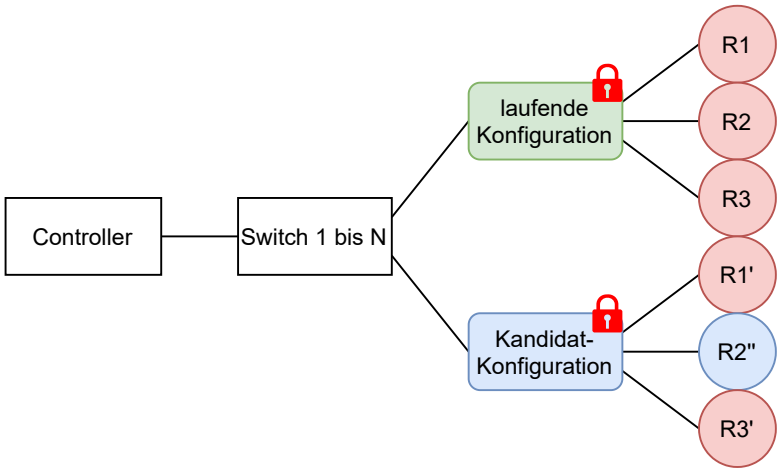

Abbildung 5.10: Das Resultat der Änderungsphase und der Ausgangszustand der Bestätigungsphase (Eigene Darstellung)

Kandidat-Konfiguration der Switches hat eine veränderte Ressource. Das Ende der Änderungsphase zeichnet gleichzeitig auch den Beginn der folgenden Bestätigungsphase aus.

Fehlerbehandlungen in der Änderungsphase

Folgende Fehlerfälle könnten in der Änderungsphase auftreten:

- Das Kopieren der laufenden Konfiguration schlägt fehl.
- Das Instanziieren der Kandidat-Konfiguration auf einem Switch schlägt fehl.
- Die Änderung an der Kandidat-Konfiguration auf einem Switch schlägt fehl.

In allen drei Fällen werden alle bisher getätigten Änderungen zurückgesetzt und alle laufenden Konfigurationen entsperrt. Das Zurücksetzen wird in Unterabschnitt 5.2.5 und das Entsperren in Unterabschnitt 5.2.4 behandelt.

5.2.3 Bestätigungsphase

Die Bestätigungsphase ist die dritte Phase des zeitsynchronen Transaktionsmodells. Diese Phase wird nur nach einer erfolgreichen Sperr- und Änderungsphase durchgeführt. In der Bestätigungsphase wird jede Kandidat-Konfiguration zur neuen laufenden Konfiguration. Die Bestätigung der neuen laufenden Konfiguration kann entweder auf einen fest definierten Zeitpunkt stattfinden oder nachdem alle Switches die Änderungen an ihre Kandidat-Konfiguration durchgeführt und dem Controller dies mitgeteilt haben. In beiden Fällen wird eine Nachricht mit entweder dem ermittelten fest definierten Zeitpunkt oder ohne diesen Zeitpunkt mitgeteilt. Für das zeitsynchrone Transaktionsmodell ist ein fest definierter Zeitpunkt in der Bestätigungsphase essenziell und wird daher in Abbildung 5.11, Abbildung 5.12, Abbildung 5.13 und Abbildung 5.14 visualisiert.

Zunächst wird der Zeitpunkt für die Commit-Ausführung ermittelt und in dem Zeitstempel $T_{(mod)}$ definiert. In Abbildung 5.11 teilt der Controller den in M11 enthaltenen Zeitstempel an allen Switches. Jeder angesprochene Switch empfängt den Zeitstempel und prüft zunächst, ob dieser Zeitstempel größer als die aktuelle Zeit auf den Switch ist. Nachdem der Zeitstempel wahrgenommen werden kann, merkt sich jeder Switch diesen Zeitstempel wie in A6 beschrieben. Danach teilt jeder Switch dem Controller mit, dass

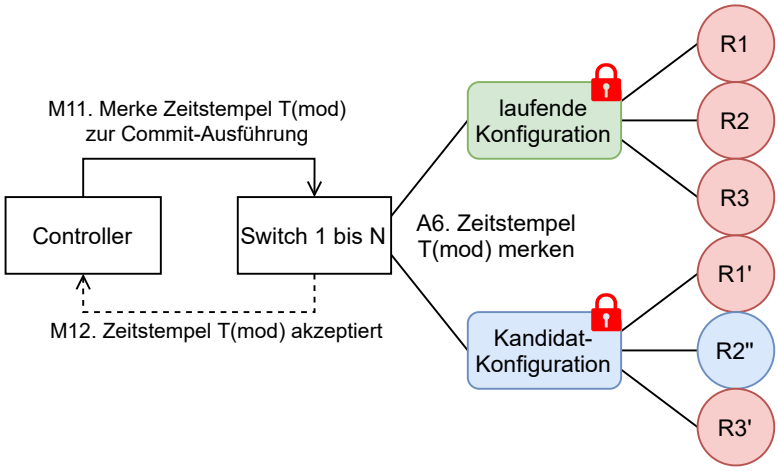

Abbildung 5.11: Das Senden des Zeitstempels (Eigene Darstellung)

der Zeitstempel akzeptiert wurde (M12). Durch die Nachricht M10 garantiert ein Switch, dass dieser den Commit zum Zeitpunkt des Zeitstempels ausführt.

Wie aus Abbildung 5.12 entnommen, gibt der Controller den Commit mit M13 an alle Switches frei. Das Versenden der Commitfreigabe an alle verbundenen Switches verläuft

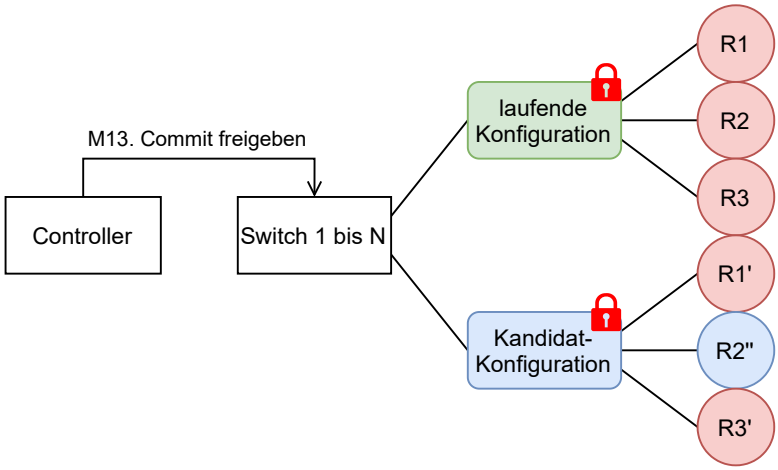

Abbildung 5.12: Die Commitfreigabe an alle Switches

nebenläufig. Jeder Switch empfängt die Commitfreigabe und wartet, bis der Zeitpunkt für das Bestätigen der neuen laufenden Konfiguration eintritt.

In Abbildung 5.13 ist die zeitsynchrone Commit-Ausführung zum Zeitpunkt $T_{(mod)}$ dargestellt. Zum Zeitpunkt $T_{(mod)}$ stellen alle Switches zeitsynchron ihre Kandidat-Konfiguration

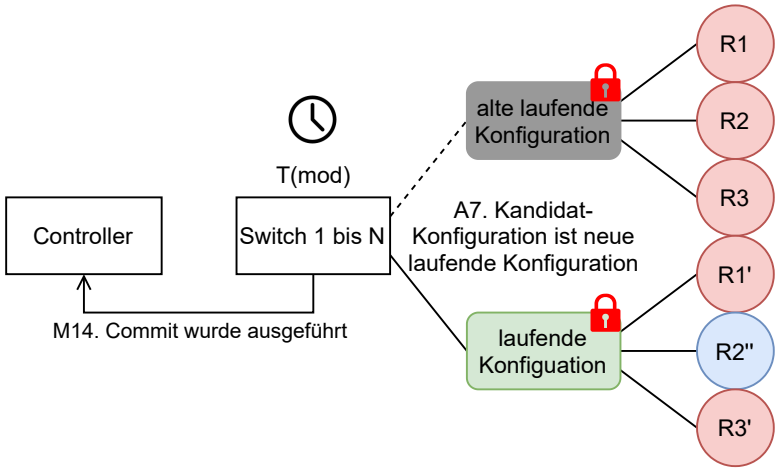

Abbildung 5.13: Die zeitsynchrone Commit-Ausführung (Eigene Darstellung)

zur neuen laufenden Konfiguration um (A7). Aus der ehemalig laufenden Konfiguration eines Switches wird die alte laufende Konfiguration. Nach der erfolgreichen Rekonfiguration bestätigt jeder Switch seine neue laufende Konfiguration (M14).

In Abbildung 5.14 ist das Resultat der Bestätigungsphase zu sehen. Jeder Switch hat die

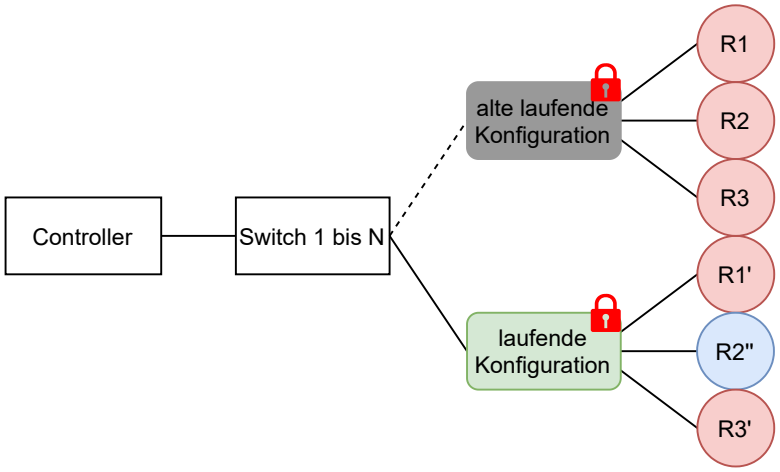

Abbildung 5.14: Das Resultat der Bestätigungsphase (Eigene Darstellung)

neue laufende und die alte laufende Konfiguration.

Fehlerbehandlung in der Bestätigungsphase

Folgende Fehlerfälle könnten in der Bestätigungsphase eintreffen:

- Das Verpassen des Zeitstempels der zeitsynchronen Transaktion
- Das Verpassen des Zeitstempels des Switches

Entweder kann es sein, dass die zeitsynchrone Transaktion den Zeitstempel verpassen wird, da die Dauer der Commitfreigabe zu lange andauern wird oder ein Switch den Zeitpunkt nicht wahrnehmen kann, da dieser den Zeitpunkt verpasst hat. In beiden Fällen werden alle bisher getätigten Änderungen zurückgesetzt und alle laufenden Konfigurationen entsperrt. Das Zurücksetzen wird in Unterabschnitt 5.2.5 und das Entsperren in Unterabschnitt 5.2.4 beschrieben.

5.2.4 Entsperrphase

Die Entsperrphase ist die vierte und letzte Phase des zeitsynchronen Transaktionsmodells. Diese Phase tritt in drei Fällen auf:

- Eine fehlgeschlagene Sperrphase
  Die laufende Konfiguration des Switches ist bereits gesperrt.
- Nach dem Zurücksetzen in Unterabschnitt 5.2.5
  Die laufenden Konfigurationen werden nach dem Zurücksetzen entsperrt.
- Nach einer erfolgreichen Sperr-, Änderungs- und Bestätigungsphase
  Die Rekonfiguration zur neuen laufenden Konfiguration aller Switches war erfolgreich.

In allen drei Fällen wird darauf geachtet, dass die zeitsynchrone Transaktion terminiert und keine Inkonsistenz auf den Switches zurücklässt. Der Ablauf der Entsperrphase nach einer erfolgreichen Sperr-, Änderungs- und Bestätigungsphase ist in Abbildung 5.15, Abbildung 5.16, Abbildung 5.17 und Abbildung 5.18 visualisiert.

In Abbildung 5.15 ist der Ausgangszustand der Entsperrphase zu sehen. Jeder Switch besitzt nun eine alte laufende Konfiguration. Es muss diese alte laufende Konfiguration entsperrt und gelöscht werden.

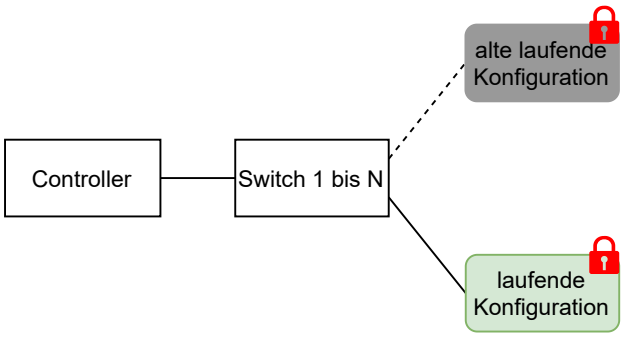

Abbildung 5.15: Der Ausgangszustand der Entsperrphase (Eigene Darstellung)

In Abbildung 5.16 teilt der Controller jedem Switch mit, dass die alten laufenden Konfigurationen gelöscht werden sollen (M17). Jeder Switch empfängt die Nachricht vom

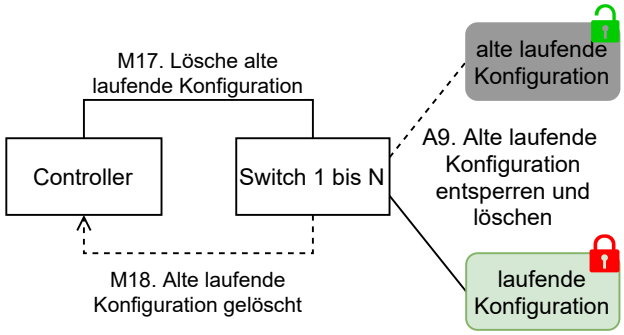

Abbildung 5.16: Das Löschen der alten laufenden Konfiguration pro Switch (Eigene Darstellung)

Controller und führt den in A9 beschriebene Befehl aus. Es wird zunächst die alte laufende Konfiguration entsperrt und danach gelöscht. Nachdem dies getan wurde, teilt jeder Switch dem Controller mit, dass die alte laufende Konfiguration gelöscht wurde (M18).

In Abbildung 5.17 teilt der Controller allen Switches mit, dass diese die neuen laufenden Konfigurationen entsperren sollen (M19). Jeder Switch empfängt die Nachricht vom Controller und entsperrt die neue laufende Konfiguration (A10). Nachdem jeder Switch seine neue laufende Konfiguration entsperrt hat, bestätigt jeder Switch dies dem Controller mit M20.

In Abbildung 5.18 ist das Resultat der Entsperrphase. Mit dem Ende der Entsperrungphase ist die zeitsynchrone Transaktion erfolgreich durchlaufen.

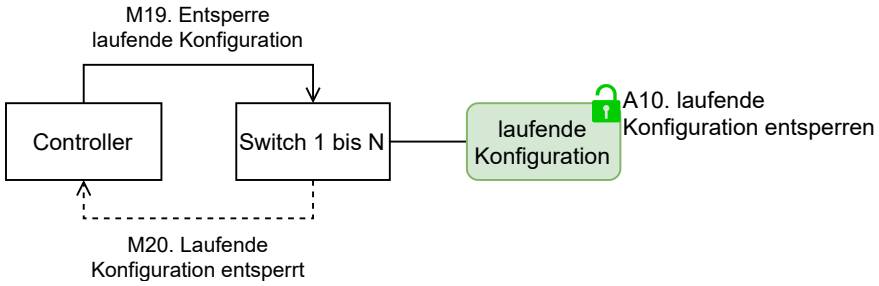

Abbildung 5.17: Das Entsperren aller Switches (Eigene Darstellung)

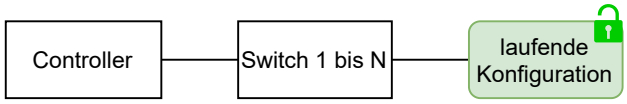

Abbildung 5.18: Das Resultat der Entsperrphase (Eigene Darstellung)

Fehlerbehandlung in der Entsperrphase

In der Entsperrphase wird keine Fehlerbehandlung getätigt. Es wurde in den Annahmen in Abschnitt 5.1 festgelegt, dass ab der Commit-Ausführung zum fest definierten Zeitpunkt das zeitsynchrone Transaktionsmodell jeden Fehler als Systemfehler ansieht.

5.2.5 Zurücksetzen

Das Zurücksetzen ist der Mechanismus, der aktiviert wird, sobald eine Fehlerbehandlung in der Änderungsphase oder in der Bestätigungsphase stattfindet. Mit dem Zurücksetzen sollen mehrere Aspekte des zeitsynchronen Transaktionsmodells berücksichtigt werden. Die zeitsynchrone Transaktion muss terminieren und darf nicht die Switches in einem inkonsistenten Zustand zurücklassen. Außerdem muss das Alles- oder Nichts-Prinzip der zeitsynchronen Transaktion bewahrt werden. Entweder waren alle bereits abgearbeiteten Operationen aus der zeitsynchronen Transaktion erfolgreich oder keiner der Operationen wird ausgeführt. Der Ablauf des Zurücksetzens ist in Abbildung 5.19, Abbildung 5.20 und Abbildung 5.21 visualisiert.

In Abbildung 5.19 ist der Ausgangszustand vom Zurücksetzen nach einer Fehlerbehandlung aus der Bestätigungsphase zu sehen. Der Controller erhielt zuvor die Nachricht, dass ein Switch den Zeitstempel zur Commit-Ausführung verpasst hat.

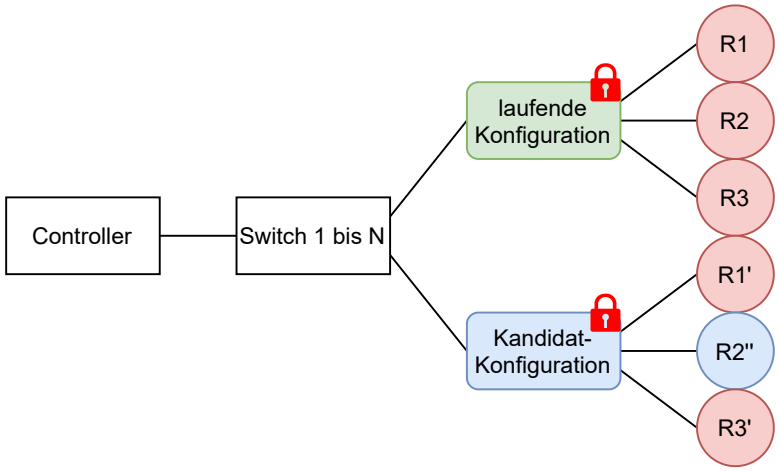

Abbildung 5.19: Der Ausgangszustand des Zurücksetzens (Eigene Darstellung)

In Abbildung 5.20 wird die Löschanforderung der Kandidat-Konfiguration dargestellt. Der Controller sendet die in M15 enthaltene Löschanforderung an alle Switches. Jeder

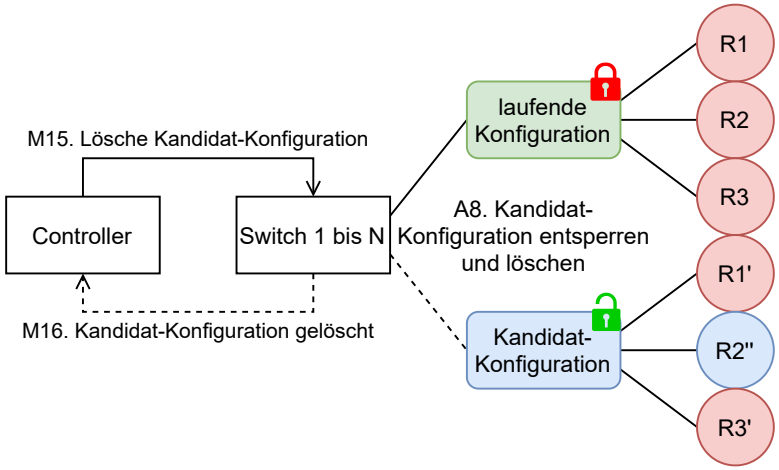

Abbildung 5.20: Das Löschen der Kandidat-Konfiguration pro Switch (Eigene Darstellung)

Switch empfängt die Nachricht vom Controller. Zunächst wird, wie in A8 beschrieben, die Kandidat-Konfiguration entsperrt und danach gelöscht. Anschließend sendet jeder Switch die in M16 enthaltene Bestätigungsnachricht, dass die Kandidat-Konfiguration gelöscht wurde.

In Abbildung 5.21 ist das Resultat vom Zurücksetzen zu sehen. Es muss nun die beste-

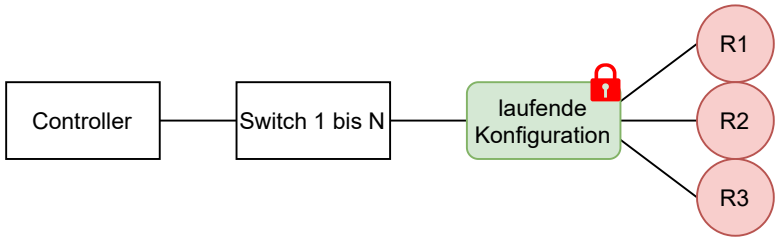

Abbildung 5.21: Das Resultat des Zurücksetzens (Eigene Darstellung)

hende laufende Konfiguration eines jeden Switches entsperrt werden. Dies wird in der Entsperrphase aus Unterabschnitt 5.2.4 durchgeführt.

5.3 Approximation zum Zeitpunkt der Commit-Ausführung

In Unterabschnitt 5.2.3 wurde der Ablauf der Bestätigungsphase erklärt. Es wird nun in diesem Abschnitt der Zeitpunkt der Commit-Ausführung mit einer Worst-Case-Berechnung abgeschätzt. Eine Abschätzung wird aus dem Grund gemacht, da ein Netzwerk dynamisch ist und keine festen Zeitspannen festgelegt sind. Datenpakete können beispielsweise während der Übertragung verzögert ankommen, da sie von anderen priorisierten Datenpaketen überholt oder gar verworfen werden. Es werden Zeitspannen definiert, die in ihrer Gesamtheit T_Z zum Zeitpunkt der Commit-Ausführung führen. In Abbildung 5.22 wurde hierzu ein Sequenzdiagramm skizziert, in denen diese Zeitspannen dargestellt sind. T_{DT} beschreibt die Zeitspanne, in der der Zeitpunkt der Commit-Ausführung rechnerisch abgeschätzt wird. T_{ST} beschreibt die Zeitspanne vom Senden der Nachricht M11 aus Abbildung 5.11 an den ersten Switch bis zum Empfangen der Nachricht M11 an den N-ten Switch. T_{CT} beschreibt die Zeitspanne, in der jeder Switch den Inhalt der Nachricht M11 speichert. T_{AT} beschreibt die Zeitspanne, in der jeder Switch die Nachricht M12 aus Abbildung 5.11 an den Controller zurücksendet. T_{CR} beschreibt die Zeitspanne, die der Controller benötigt, um die Commitfreigabe, die für N-Switches vorgesehen ist, zu prüfen. T_{SC} beschreibt die Zeitspanne vom Senden der Nachricht M13 aus Abbildung 5.12 an den ersten Switch bis zum Empfangen der Nachricht M13 an den N-ten Switch. T_{CC} beschreibt die Zeitspanne, in der jeder Switch sich auf die zeitsynchrone Commit-Ausführung bereithält, um diesen auszuführen. Bei allen benannten Zeitspannen handelt es sich um **maximale** Zeitspannen. Fasst man alle aufgelisteten Zeitspannen

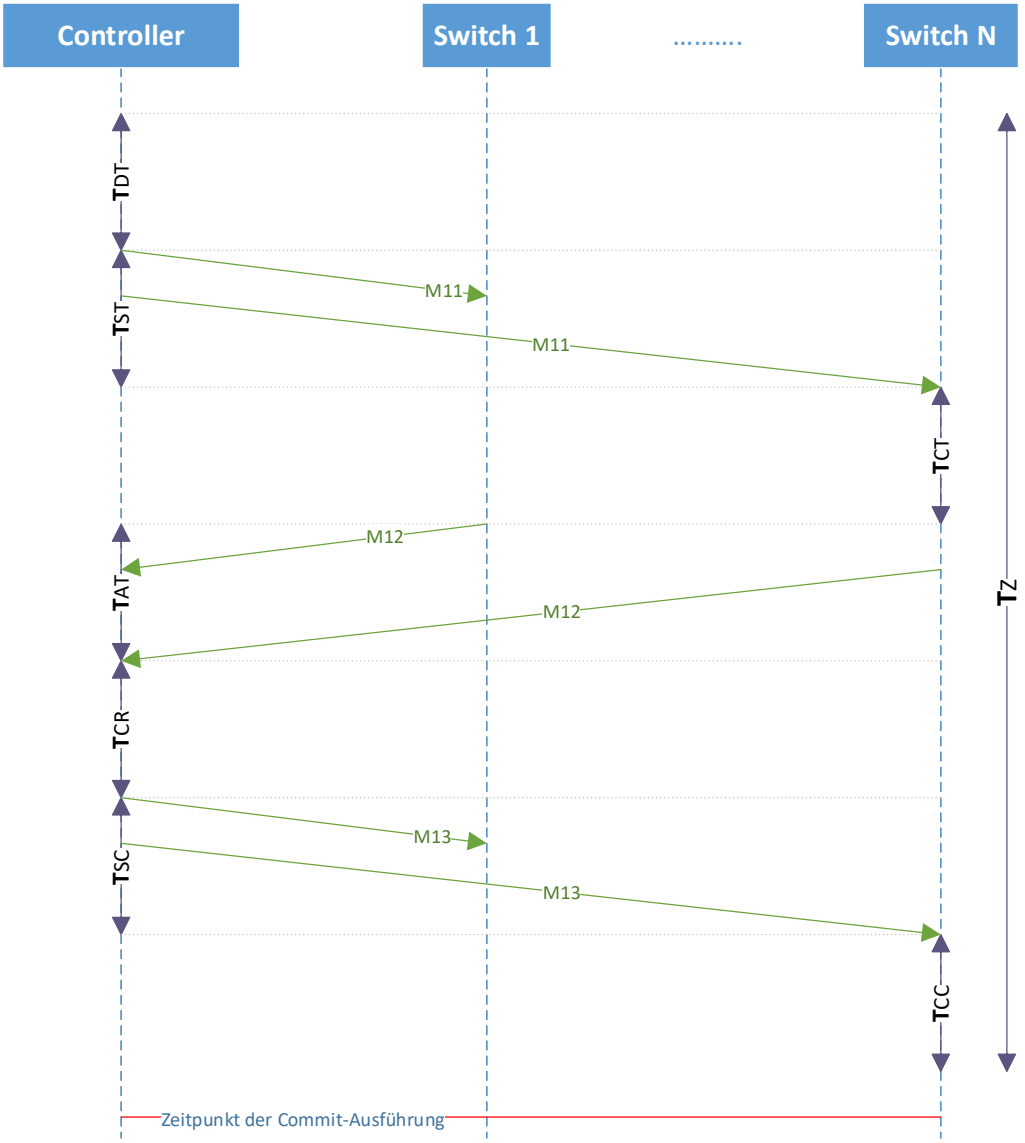

Abbildung 5.22: Das Sequenzdiagramm mit den Zeitspannen (Eigene Darstellung)

zusammen, so resultiert daraus die Zeitspanne T_Z, die in Gleichung 5.1 beschrieben ist.

$$T_Z = T_{DT} + T_{ST} + T_{CT} + T_{AT} + T_{CR} + T_{SC} + T_{CC} \tag{5.1}$$

Bei den Zeitspannen T_{ST}, T_{AT} und T_{SC} handelt es sich um maximale Latenzzeiten. Im Gegensatz dazu spiegeln die Zeitspannen T_{CT}, T_{CR} und T_{CC} maximale Verarbeitungszeiten auf dem Controller (T_{CR}) und auf N-Switches (T_{CT} und T_{CC}) wider.

Bevor die maximalen Latenzzeiten zur Bestimmung der Zeitspannen T_{ST}, T_{AT} und T_{SC} aus Abbildung 5.22 abgeschätzt werden, wird darauf eingegangen, wie eine Latenzberechnung durchgeführt wird. Es folgt die Latenzberechnung nach Steinbach [30] (Kapitel 3.4.2). Die Latenz für Ethernet besteht aus den Verzögerungen in den Switches, der Signallaufzeit und der Übertragungszeit auf den Links. Die Übertragungszeit resultiert aus der Frame-Größe und die Bit-Übertragungszeit. Außerdem sind die Kabellänge und die Anzahl an Switches, die zwischen Sender und Empfänger vorhanden sind, wichtige Kennzahlen für die Latenz. Alle beschriebenen Aspekte zur Latenz werden nun aufgelistet und in Gleichung 5.2 formuliert.

- Die Latenz für Ethernet wird mit t_L gekennzeichnet.
- Die Switchverzögerung wird mit t_{SD} gekennzeichnet. Laut Steinbach [30] seien die Switchverzögerungen implementierungsspezifisch und hängen vom Schedule ab. Steinbach [30] verwendet in seiner Arbeit eine Switchverzögerung von 2,4µs. Da in dieser Arbeit die Werte der Simulationsumgebung verwendet werden, wird eine Switchverzögerung von 3µs verwendet.
- Die Signallaufzeit wird mit t_{WD} gekennzeichnet und beträgt für Kupferkabel $\frac{1}{200000000\frac{m}{s}}$[30].
- Die Ethernet-Frame-Größe wird mit l_F gekennzeichnet und kann zwischen 512 bit (64 Byte) und 12000 bit (1500 Byte) groß sein.
- Die Bit-Übertragungszeit wird mit t_b gekennzeichnet. Laut Steinbach [30] beträgt die Bit-Übertragungszeit für eine 100 Mbit/s Leitung $0,01\frac{\mu s}{bit}$. Für eine 1 Gbit/s Leitung hingegen beträgt die Bit-Übertragungszeit $0,001\frac{\mu s}{bit}$.
- Die Kabellänge wird mit l_W gekennzeichnet.
- Die Anzahl an Hops, die zwischen Sender und Empfänger vorhanden sind, ist mit n_S gekennzeichnet.

Ethernet-Frame-Größe in Bit	Zeitspanne
1808 Bit	T_{ST}
896 Bit	T_{AT}
864 Bit	T_{SC}

Tabelle 5.1: Ethernet-Frame-Größen zu den Zeitspannen

Die Latenz t_L wird in Gleichung 5.2 aus den benannten Punkten folgendermaßen zusammengestellt.(Vgl. Steinbach [30])

$$t_L = t_{WD} * l_W + (n_S + 1) * l_F * t_b + \sum_{i=1}^{n_S} t_{SD_i} \tag{5.2}$$

Abschätzung der Latenzen Es werden die einzelnen Zeitspannen aus Gleichung 5.1 anhand der Formel zur Latenzberechnung t_L aus Gleichung 5.2 abgeschätzt. Die Größen t_{WD} und t_{SD} aus Gleichung 5.2 werden für alle Datenpakete gleich angenommen. Im Gegensatz dazu hängen die Größen l_W, n_S, l_F und t_b aus Gleichung 5.2 von verschiedenen Faktoren ab, die Einfluss auf eine maximale Latenz und der Worst-Case-Berechnung haben. Die Kabellänge l_W kann vom Controller zu den Switches unterschiedliche Längen haben. Datenpakete können bei verschiedenen Kabellängen zwischen Controller und Switches zu unterschiedlichen Verzögerungen führen. Es wird immer von einer gleichen Kabellänge zwischen Controller und Switches ausgegangen. Darüber hinaus können weitere Switches zwischen dem Sender und der Empfänger von Datenpaketen stationiert sein, weshalb die Anzahl an Hops n_S dementsprechend variiert. Zudem unterscheiden sich die Ethernet-Frame-Größen l_F der Datenpakete. Für die in dieser Arbeit zu ermittelnden Latenzen wurden Datenpaketgrößen für die Zeitspannen T_{ST}, T_{AT} und T_{SC} aus Gleichung 5.1 aus dem NetConf-RFC [8] festgelegt. Diese Datenpaketgrößen sollen eine Annäherung zu realen Datenpaketgrößen darstellen und wurden in die Simulationsumgebung integriert. Folgende Ethernet-Frame-Größen entsprechen den Zeitspannen T_{ST}, T_{AT} und T_{SC} aus Gleichung 5.1 und sind in Tabelle 5.1 einzuordnen. Außerdem hängt die Bit-Übertragungszeit von der Übertragungskapazität eines Kabels ab. Ergänzend zu den benannten Größen hängt das sogenannte Queueing-Delay t_{Queue} vom Zustand einer Leitung ab, die entweder frei oder durch bereits übertragene Datenpakete belegt sein kann und somit verzögert wird. Die Verzögerung von t_{Queue} steigt, je mehr Hops n_s zwischen einem Sender und einem Empfänger sind. Hierfür werden Datenpakete des zeitsynchronen Transaktionsmodells mit der höchsten Priorität im Netzwerk ausgestat-

tet. Dadurch können Datenpakete des zeitsynchronen Transaktionsmodells um die Dauer eines maximalen Ethernet-Frames verzögert werden.

Aus den beschriebenen Erkenntnissen werden die Latenzen der Zeitspannen T_{ST}, T_{AT} und T_{SC} folglich definiert.

$$t_L(T_{ST}) = t_{WD} * l_W + (n_S + 1) * 1808bit * t_b + \sum_{i=1}^{n_S} t_{SD_i} + (\sum_{i=1}^{n_S} t_{Queue}) \tag{5.3}$$

$$t_L(T_{AT}) = t_{WD} * l_W + (n_S + 1) * 896bit * t_b + \sum_{i=1}^{n_S} t_{SD_i} + (\sum_{i=1}^{n_S} t_{Queue}) \tag{5.4}$$

$$t_L(T_{SC}) = t_{WD} * l_W + (n_S + 1) * 864bit * t_b + \sum_{i=1}^{n_S} t_{SD_i} + (\sum_{i=1}^{n_S} t_{Queue}) \tag{5.5}$$

Für die Verarbeitungszeit eines Switches (T_{CT} und T_{CC}) und vom Controller (T_{DT} und T_{CR}) gibt es keine exakten Zeiten, da diese von der Hardware eines Switches beziehungsweise des Controllers und seiner NETCONF-Implementierung abhängig sind. Außerdem spielen auch Aspekte der Rechenleistung und Auslastung hierbei eine wichtige Rolle, weshalb dieser Fakt hiermit bekräftig wird. Es gibt Messungen aus Yoo et al. [35] zum Subtree Filtering, die darauf deuten, dass die Verarbeitung von empfangenen XML-Daten auf einem Switch eine minimale Verarbeitungszeit von 2ms verursachen. Hierbei ist die Größe der XML-Datei aus der Arbeit von Yoo et al. ausschlaggebend für das Subtree Filterng gewesen. In ihrer Arbeit wurden XML-Größen von bis zu 1500 Byte verwendet. Die XML-Konfigurationen, die in dieser Arbeit verwendet werden, liegen unterhalb dieser XML-Größe (Tabelle 5.1). Außerdem haben Lee et al. [34] in ihrer Arbeit ermittelt, dass ihre NETCONF-Implementierung auch eine Verarbeitungszeit von 2ms für XML-Daten benötigen. In der Simulationsumgebung OMNeT++ wird aufgrund der Ergebnisse aus den Arbeiten von Yoo et al. [35] und Lee et al. [34] eine maximale Verarbeitungszeit von 2ms für einen Switch angenommen. T_{CT} und T_{CC} aus Gleichung 5.1 sind jeweils mit 2ms abgeschätzt. Für die maximale Verarbeitungszeit eines Controllers wird eine Dauer von 1ms angenommen, da ein Controller eine höhere Rechenleistung als einen Switch hat. T_{DT} und T_{CR} aus Gleichung 5.1 sind jeweils mit 1ms abgeschätzt.

Phase im Transaktionsmodell	Zustände in der FSM
Sperrphase	Warte auf Sperrantwort
Änderungsphase	Warte auf Kandidatbestätigung
	Warte auf Kandidatsperrung
	Warte auf Änderungsbestätigung
	Warte auf Zeitstempelbestätigung
Bestätigungsphase	Warte auf Commitausführung
Entsperrphase	Warte auf Löschung alter Konfiguration
	Warte auf Entsperrung
Zurücksetzen	Fehlerzustand
	Warte auf Kandidatlöschung

Tabelle 5.2: Zustände der FSM in den Phasen des Transaktionsmodells

5.4 Finite State Machine

Für das zeitsynchrone Transaktionsmodell wurde eine FSM modelliert, die das Verhalten des zeitsynchronen Transaktionsmodells in Abbildung A.1 darstellt und als Unified Modelling Language (UML)-Zustandsdiagramm skizziert wurde. Es ist davon auszugehen, dass die Verbindung zwischen Controller und Switches bereits existiert. In der in Abbildung A.1 modellierten FSM wurde eine Mengennotation eingeführt, um sowohl den Zustand der zeitsynchronen Transaktion als auch die Rückmeldungen der Switches einzuordnen. Alle Mengen sind zu Beginn der zeitsynchronen Transaktion initialisiert. Die Zustände der FSM, die in Abbildung A.1 modelliert wurden, werden zu den jeweiligen Phasen des zeitsynchronen Transaktionsmodells in Tabelle 5.2 eingeordnet. Es folgt nun eine tabellarische Beschreibung der Mengen in Tabelle 5.3, die in Abbildung A.1 verwendet wurde.

5.5 Vier Arten der Netzwerkrekonfiguration

In diesem Abschnitt wird auf die vier Arten der Netzwerkrekonfiguration eingegangen. Die vier Arten der Netzwerkrekonfiguration zeichnen sich dadurch aus, wie in einem programmierbaren zeitkritischen Netzwerk Änderungen an einer bestehenden Netzwerkkonfiguration durchgeführt werden. Im Folgenden werden diese vier Arten der Netzwerkrekonfiguration aufgezählt und nacheinander erklärt.

Menge	Beschreibung
L	L ist die Menge der Switches, deren laufende Konfiguration gesperrt wurde.
K	K ist die Menge der Switches, die aus L besteht mit einer Kandidat-Konfiguration.
J	J ist die Menge der Switches, die aus K besteht und deren Kandidat-Konfiguration gesperrt wurde.
K'	K' ist die Menge der Switches, die aus J besteht und deren Kandidat-Konfiguration modifiziert wurde.
Z	Z ist die Menge der Switches, die aus K' besteht und den Zeitstempel zur Commit-Ausführung erhalten haben.
F	F ist die Menge der Switches, die aus Z besteht und den Commit zum Zeitpunkt des Zeitstempels ausgeführt haben.
Q	Q ist die Menge der Switches, die aus F besteht und deren Kandidat-Konfiguration gelöscht wurden.
E	E ist die Menge der Switches, die entweder aus L, D oder Q besteht und deren laufender Konfiguration entsperrt wurde.
G	G ist die Fehlermenge, in der sich mindestens ein Switch aus den Mengen L, K, K' und Z befindet.
D	D ist die Menge der Switches, die aus G und entweder aus L, K, K' oder Z stammen und deren Kandidat-Konfiguration gelöscht wurde.
Sw	Sw ist die Menge der verfügbaren Switches.
Ae	Ae ist die Menge der Änderungen, die ein Switch machen muss.
N	N ist Anzahl an Switches.
Sx	Sx ist der jetzt angesprochene Switch in der jeweiligen Menge.

Tabelle 5.3: Beschreibung der Mengen aus der FSM in Abbildung A.1

Die nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfiguration Die Art dieser Netzwerkrekonfiguration bestand vor der Einführung des zeitsynchronen Transaktionsmodells dieser Arbeit. Es wird mit der NETCONF-Operation **edit-config** eine Änderung an der laufenden Konfiguration eines Switches eingeleitet. Hierbei erhält die Anwendung im SDN-Controller keine Bestätigung, ob die Änderung erfolgreich oder nicht durchgeführt wurde. Dies entspricht dem Direct model Ansatz aus Abschnitt 2.3.

Die nicht-zeitsynchrone Transaktion Bei dieser Art der Netzwerkrekonfiguration wird das zeitsynchrone Transaktionsmodell verwendet, um Änderungen auf mehreren Switches nebenläufig durchzuführen. Es werden alle Phasen aus dem Konzept in Kapitel 5 vollzogen. Jedoch wird keine Worst-Case-Berechnung für die Commit-Ausführung auf allen Switches vollzogen. Die Commit-Ausführung kann pro Switch variieren und die neuen laufenden Konfigurationen zu verschiedenen Zeitpunkten bestätigen. Es gibt somit keine Garantie, dass alle Switches gleichzeitig und zeitsynchron die Commit-Ausführung zu einem definierten Zeitpunkt tätigen.

Die zeitsynchrone Transaktion Diese Art der Netzwerkrekonfiguration beschreibt Transaktionen aus dem zeitsynchronen Transaktionsmodell. Auch hier werden alle vier Phasen aus dem Konzept in Kapitel 5 vollzogen und ein fest definierter Zeitpunkt für die Commit-Ausführung mit der Worst-Case-Berechnung ermittelt. Zu diesem fest definierten Zeitpunkt führen alle Switches garantiert und zeitsynchron die Commit-Ausführung durch.

Die zeitsynchrone Transaktion zum Start der Hyperperiode Diese Netzwerkrekonfiguration ist eine Erweiterung der zeitsynchronen Transaktion. Der fest definierte Zeitpunkt wird zum nächstmöglichen Start der Hyperperiode garantiert und zeitsynchron vollzogen. Durch diesen fest definierten Zeitpunkt wird sichergestellt, dass keine Datenpakete im Umlauf des programmierbaren und zeitkritischen Netzwerkes sind. Weder sind Datenpakete bei der Übertragung, noch werden Datenpakete von Switch zu Switch weitergeleitet.

6 Implementierung und Qualitätssicherung

In diesem Kapitel werden auf die Erweiterungen eingegangen, die notwendig waren, um das in Kapitel 5 beschriebene Konzept umzusetzen. Es wird einerseits auf die Implementierung des zeitsynchronen Transaktionsmodells eingegangen. Andererseits wird auf die Qualitätssicherung des zeitsynchronen Transaktionsmodells eingegangen, in der verschiedene Tests durchgeführt und einige Anforderungen erfüllt werden.

6.1 Implementierung des zeitsynchronen Transaktionsmodells

Um das beschriebene Konzept aus Kapitel 5 umzusetzen, wurden einige Erweiterungen an das SDN4CoRE-Framework vorgenommen. In Abbildung 6.1 sind die Erweiterungen mit roter Beschriftung zusehen und werden nacheinander erklärt. Es wurde nur der Anteil der NETCONF-Implementierung des SDN4CoRE-Framework für diese Arbeit erweitert. Das SDN4CoRE-Framework bietet auch OpenFlow-Implementierungen an, die aus [10] zu entnehmen sind.

NetConfApplicationBase und NetConfDataStoreManagerBase

Die NetConfApplicationBase und die NetConfDataStoreManagerBase haben bereits implementierte NETCONF-Funktionen wie **copy-config** und **edit-config** geboten. Netzwerkrekonfigurationen wie der nicht-zeitsynchronen und nicht-transaktionalen Netzwerkrekonfiguration aus Absatz 5.5 werden mit **edit-config**-Befehlen realisiert. Für die Umsetzung von zeitsynchronen Transaktionen wurden die NetConfApplicationBase und die NetConfDataStoreManagerBase um die NETCONF-Operationen **lock**, **unlock** und

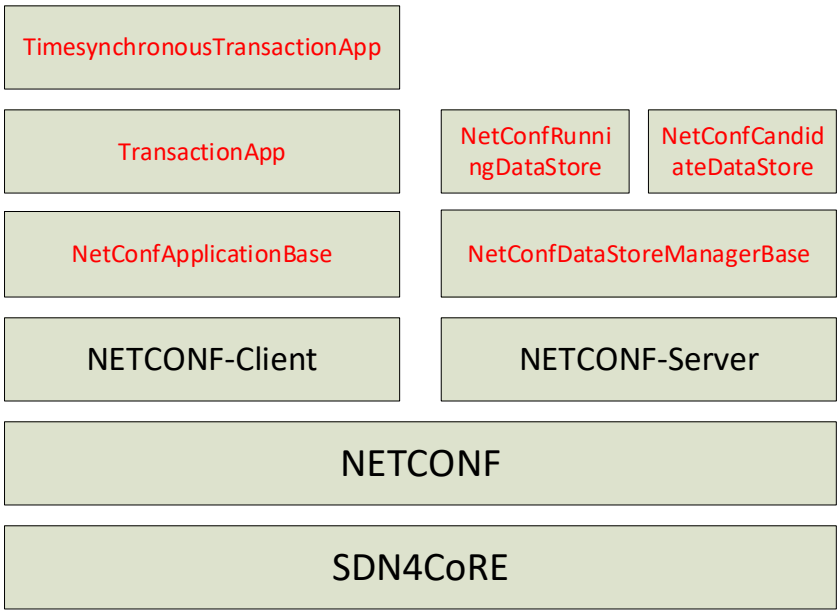

Abbildung 6.1: Die Erweiterungen an das SDN4CoRE-Framework

commit erweitert. Außerdem wurden die Datenpakete, die zwischen dem NETCONF-Client und dem NETCONF-Server ausgetauscht werden, mit realen Datenpaketgrößen ausgestattet.

NetConfRunningDataStore und NetConfCandidateDataStore

Durch die erweiterten NETCONF-Funktionen wurde das Transaktionsmodell von NETCONF aus Abschnitt 2.3 entwickelt. Es wurde ein Kandidat-Datenspeicher angelegt, welches die Kandidat-Konfiguration beinhaltet. Dadurch können Netzwerkrekonfigurationen sowohl auf dem laufenden Konfigurationsdatenspeicher beziehungsweise der laufenden Konfiguration als auch auf den Kandidat-Konfigurationsdatenspeicher beziehungsweise der Kandidat-Konfiguration angewendet werden.

TransactionApp

Die vier Phasen und das Zurücksetzen aus dem in Kapitel 5 beschriebenen Konzept wurden in der TransactionApp entwickelt. In der TransactionApp werden transaktionale

Netzwerkrekonfiguration realisiert, wie sie in Absatz 5.5 beschrieben sind. Die TransactionApp ist eine Basisimplementierung für das zeitsynchrone Transaktionsmodell, welches im folgenden Abschnitt beschrieben wird.

TimesynchronousTransactionApp

Die TimesynchronousTransactionApp spiegelt das Verhalten der modellierten FSM in Abbildung A.1 wieder. Die TimesynchronousTransactionApp nutzt Module wie **Period**, **Timer** und **Oscillator** aus dem CoRE4INET-Framework [5]. Diese Module werden für die Worst-Case-Berechnung aus Abschnitt 5.3 verwendet. Das **Period**-Modul wird benötigt, um den aktuellen Zyklus im Netzwerk zu ermitteln. Das **Oscillator**-Modul wird benötigt, um die Länge der Netzwerkperiode zu erhalten. Mit dem **Timer**-Modul erhält man einen aktuellen Zeitpunkt in der Simulationsumgebung. Die TimesynchronousTransactionApp kann zwei verschiedene fest definierte Zeitpunkte ermittelt. Zum einen kann als fest definierter Zeitpunkt ein Zeitpunkt innerhalb der Netzwerkperiode als Commit-Ausführungszeitpunkt definiert werden. Dieser fest definierte Zeitpunkt wird benötigt, um zeitsynchrone Transaktionen zu realisieren, wie sie in Absatz 5.5 beschrieben wurden. Zum anderen kann als fest definierter Zeitpunkt der nächstmögliche Start der Hyperperiode als Commit-Ausführungszeitpunkt definiert werden. Zeitsynchrone Transaktionen, die in Absatz 5.5 beschrieben wurden, werden dementsprechend realisiert.

6.2 Qualitätssicherung des zeitsynchronen Transaktionsmodells

In diesem Abschnitt wird auf die Qualitätssicherung des zeitsynchronen Transaktionsmodells eingegangen. Mit der Qualitätssicherung werden Grundfunktionen des zeitsynchronen Transaktionsmodells überprüft. Die Simulationsumgebung OMNeT++ bietet dazu im Rahmen des INET-Frameworks [23] Smoketests' und Fingerprinttests'. Mit einem Smoketest werden Netzwerke für eine bestimmte Dauer simuliert und auf Abstürze und Laufzeitfehler überprüft. Mit dem Smoketest wird sichergestellt, dass das Netzwerk nicht abstürzt. Das Verhalten des Netzwerkes wird jedoch nicht mit dem Smoketest abgedeckt.

Fingerprinttests' hingegen führen rechnerisch am Ende einer Simulation eines Netzwerkes einen Hashwert. Dieser Hashwert wird bei einer erneuten Ausführung der Simulation

verwendet, um festzustellen, ob sich das Verhalten des Netzwerkes beispielsweise durch die Modifikation von Codesequenzen geändert hat. Sowohl der Smoketest als auch der Fingerprinttest geben keine Auskunft darüber, ob das Netzwerk sich sinnvoll vor, während und nach einer zeitsynchronen Transaktion verhält.

6.2.1 Zustandsübergangs- und Zustandsbasierte Tests

Für das Verhalten des zeitsynchronen Transaktionsmodells wird ein zustandsbasierter Test eingeführt. Ein zustandsbasierter Test ist ein Black-Box-Testverfahren. Hierbei werden Testfälle aus dem Zustandsdiagramm des zeitsynchronen Transaktionsmodells abgeleitet, um zu bewerten, ob das zeitsynchrone Transaktionsmodell gültige Zustandsübergänge erfolgreich ausführt und ungültige Übergänge verhindert. Es wird eine vollständige Zustandsüberdeckung und eine vollständige Zustandsübergangsüberdeckung erzielt. Für den zustandsbasierten Test des zeitsynchronen Transaktionsmodells wurde ein Test-Netzwerk in der Simulationsumgebung OMNeT++ angefertigt. In Abbildung 6.2 sendet der timesynchronousTransactionAppTest direkte RPC-Reply Nachrichten an die timesynchronousTransactionApp, um das zeitsynchrone Transaktionsmodell in neue Zustände zu überführen. Die Testfälle werden aus dem Zustandsdiagramm in Abbildung A.1

Abbildung 6.2: Das Test-Netzwerk für das zustandsbasierte Testen

generiert. Hierfür wurde ein Übergangsbaum in Abbildung 6.3 angefertigt, in denen die Testfälle visuell dargestellt werden. Die Trigger, die zu einem Zustandsübergang überführen, wurden aus Gründen der Übersichtlichkeit durch Zahlen von 1 bis 42 ersetzt und sind aus dem Zustandsdiagramm Abbildung A.1 zu entnehmen. Für die Zustandsübergangsüberdeckung werden Testfälle aus dem Übergangsbaum aus Abbildung 6.3 definiert. Vom <initial>-Zustand bis zu einem <final>-Zustand im Übergangsbaum entspricht dies einem Testfall. Für die Zustandsüberdeckung werden auch Testfälle aus dem Übergangsbaum entwickelt, indem ein Testfall einen maximalen Durchlauf aller Zustände entspricht.

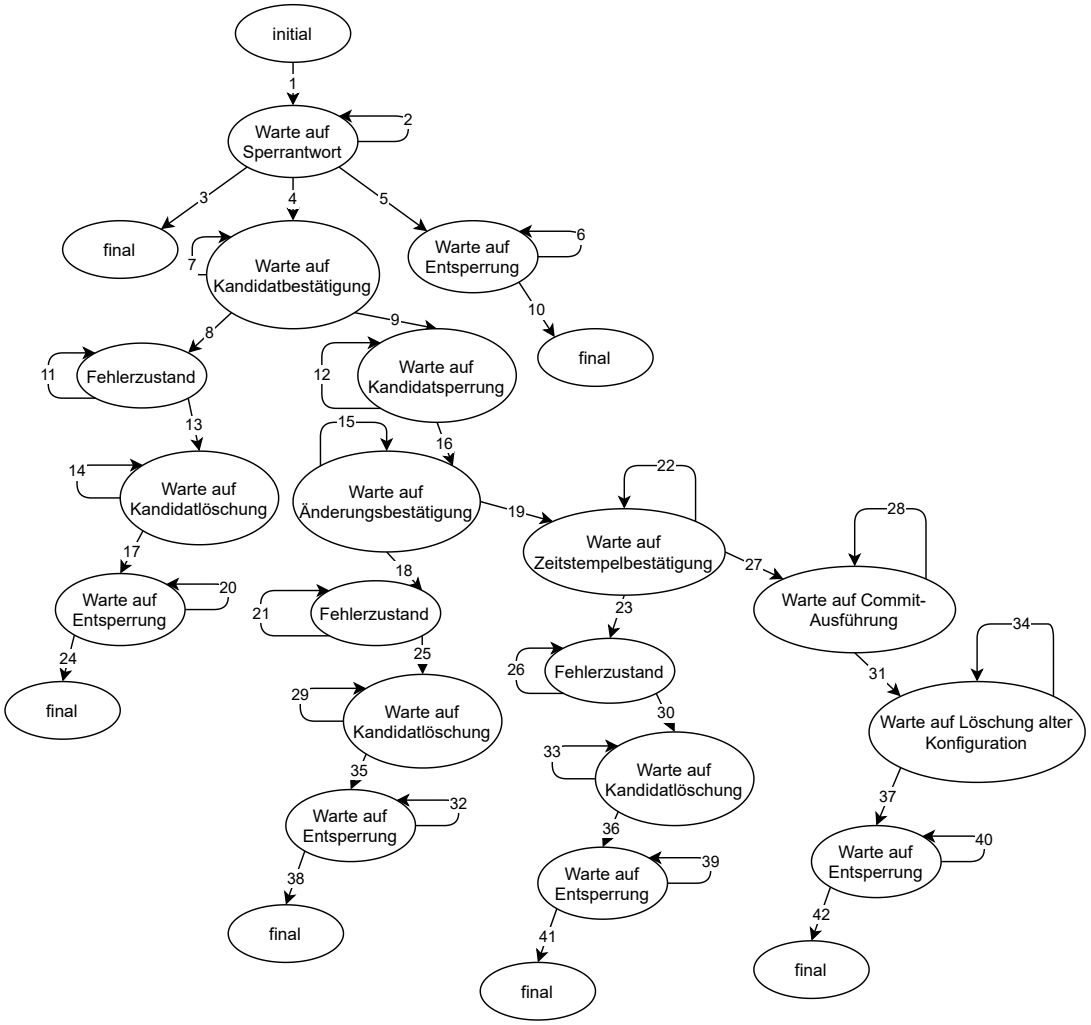

Abbildung 6.3: Der Übergangsbaum für das zustandsbasierte Testen

6.2.2 Integrationstests

Integrationstests werden zum Testen verwendet, um das Zusammenspiel von Gruppen von Software-Komponenten zu überprüfen. Für die Integrationstests des zeitsynchronen Transaktionsmodells gelten als Software-Komponenten die in Abbildung 6.1 abgebildete TimesynchronousTransactionApp, der NetConfDataStoreManagerBase sowie der NetConfRunningDataStore und der NetConfCandidateDataStore. Im Folgenden werden die Testfälle für die Integrationstests beschrieben. Für den weiteren Verlauf dieses Kapitels werden zeitsynchrones Transaktionsmodell und TimesynchronousTransactionApp als Synonym verwendet.

Erfolgreiche zeitsynchrone Transaktion

In diesem Testfall wird der erfolgreiche Ablauf einer zeitsynchronen Transaktion ausgeführt. Es werden hierfür zwei Anwendungen auf dem SDN-Controller eingeführt. Die erste Anwendung beinhaltet die TimesynchronousTransactionApp und die zweite Anwendung ist die NetConfApplicationBase, die in Abbildung 6.1 dargestellt ist. Die Änderungen der Flusstabellen und der GCLs aller Switches sind in XML-Dateien eingetragen. Die TimesynchronousTransactionApp führt gleichzeitig zwei neue Flusseinträge auf jeweils einen Switch und eine jeweils neue GCL pro Switch ein. Während die TimesynchronousTransactionApp Nachrichten austauscht, sendet die NetConfApplicationBase auch Nachrichten an die bereits angesprochenen Switches.

Mit diesem Testfall konnte die Anforderung **FA1 - Zeitsynchrone Transaktion** aus Abschnitt 4.2 erfüllt werden. Die TimesynchronousTransactionApp führte alle angeforderten einzelnen Operationen als zeitsynchrone Transaktion aus. Außerdem wurde die zeitsynchrone Transaktion der TimesynchronousTransactionApp nicht von den angefragten Nachrichten der NetConfApplicationBase verzögert und führte die zeitsynchrone Transaktion atomar aus. Hierdurch wurde die Anforderung **FA2 - Atomizität** aus Abschnitt 4.2 erfüllt. Schließlich wurde das Netzwerk und die Switches, an denen die Netzwerkrekonfiguration durchgeführt wurde, von einem validen Zustand in einen neuen validen Zustand überführt. Somit gilt die Anforderung **FA3 - Konsistenz** aus Abschnitt 4.2 als erfüllt.

Fehlerbehandlung in der Sperrphase

In diesem Testfall wird die Fehlerbehandlung in der Sperrphase aus Unterunterabschnitt 5.2.1 ausgeführt. Hierfür werden zwei Anwendungen mit verschiedenen XML-Konfigurationsdateien in den SDN-Controller eingeführt. Bei beiden Anwendungen handelt es sich um die TimesynchronousTransactionApp. Beide TimesynchronousTransactionApps beginnen zum selben Zeitpunkt mit dem Verbindungsaufbau der Kommunikation. Der Beginn beider zeitsynchronen Transaktionen wird zu verschiedenen Zeitpunkten vollzogen. Bei diesem Testfall stellte sich heraus, dass ein bereits für die zeitsynchrone Transaktion gesperrter Switch für eine andere zeitsynchrone Transaktionen nicht zur Verfügung gestellt werden kann. Nur wenn eine zeitsynchrone Transaktion terminiert, kann eine nächste zeitsynchrone Transaktion die Switches sperren. Mit diesem Testfall konnte die Anforderung **NFA1 - Fehlertoleranz** aus Abschnitt 4.3 erfüllt werden. Es wurde eine Maßnahme

eingeleitet, um eine zeitsynchrone Transaktion terminieren zu lassen, da ein Switch bereits für eine andere zeitsynchrone Transaktion verwendet wird.

Fehlerbehandlungen in der Änderungsphase

Die Testfälle dieses Abschnittes befassen sich mit den Fehlerbehandlungen aus der Änderungsphase in Unterunterabschnitt 5.2.2. Diese sind in zwei Testfällen unterschieden.

Der erste Testfall befasst sich mit dem Instanziieren der Kandidat-Konfiguration von einem beteiligten Switch. In diesem Testfall wird erzwungen, dass das Instanziieren des Kandidat-Konfiguration, welcher sich im Kandidat-Konfigurationsdatenspeicher aufhält, fehlschlagen soll. Für den Testfall werden zwei Anwendungen auf dem SDN-Controller eingeführt. Die erste Anwendung beinhaltet das TimesynchronousTransactionApp und die zweite Anwendung ist die NetConfApplicationBase. Während das TimesynchronousTransactionApp den Verbindungsaufbau vollzieht, führt die NetConfApplicationBase eine Sperranforderung an den Kandidat-Konfigurationsdatenspeicher eines Switches. Zum Zeitpunkt des Sperrens des Kandidat-Konfigurationsdatenspeichers eines Switches bricht die zeitsynchrone Transaktion der TimesynchronousTransactionApp ab. Die zeitsynchrone Transaktion leitet das Zurücksetzen aus Unterabschnitt 5.2.5 ein, wodurch alle bereits gesperrten und mit einer Kandidat-Konfiguration ausgestatteten Switches verworfen werden. Bevor es jedoch zur Entsperrung der Kandidat-Konfiguration kommt, sendet die NetConfApplicationBase eine Entsperrnachricht an den gesperrten Kandidat-Konfigurtionsdatenspeichers des Switches, um somit die Terminierung der zeitsynchronen Transaktion zu ermöglichen. Danach werden alle gesperrten laufenden Konfigurationen der Switches, wie in der Entsperrphase aus Unterabschnitt 5.2.4 beschrieben, entsperrt.

Der zweite Testfall beinhaltet die fehlerhafte Konfiguration einer XML-Datei. Hierfür wird die TimesynchronousTransactionApp mit einer fehlerhaften XML-Datei ausgestattet. In der XML-Datei wird eine undefinierte Portnummer angegeben. Die TimesynchronousTransactionApp sendet die Änderung mit der undefinerten Portnummer über den NETCONF-Client und den NETCONF-Server an den NetConfDataStoreManagerBase. Der NetConfDataStoreManagerBase überprüft die gesendete Änderung und meldet die undefinierte Portnummer als Fehler zurück. Die TimesynchronousTransactionApp erhält die Rückmeldung zu der zuvor versandten Änderung zurück und leitet das Zurücksetzen und die Entsperrphase ein.

In beiden Testfällen konnten die Anforderungen **NFA1 - Fehlertoleranz** aus Abschnitt 4.3 und **FA3 - Konsistenz** aus Abschnitt 4.2 erneut erfüllt werden. Das Netzwerk sowie die beteiligten Switches wurden in den Ursprungszustand vor der zeitsynchronen Transaktion zurückversetzt und das zeitsynchrone Transaktionsmodell war in der Lage, eine Maßnahmen gegen den anwendungsseitigen Fehler einzuleiten.

Fehlerbehandlungen in der Bestätigungsphase

Die Testfälle dieses Abschnittes befassen sich mit den Fehlerbehandlungen in der Bestätigungsphase in Unterunterabschnitt 5.2.3. Diese sind in zwei Testfällen unterschieden.

Der erste Testfall befasst sich mit dem Verpassen des Zeitstempels zur Commit-Ausführung auf einem Switch. Für das Provozieren dieses Testfalls werden an den Switches verschiedene Verarbeitungszeiten angegeben. Während der Bestätigungsphase errechnet die TimesynchronousTransactionApp den Zeitstempel zur gleichzeitigen und zeitsynchronen Commit-Ausführung auf allen beteiligten Switches. Der Zeitstempel wird danach an den NetConfDataStoreManagerBase des Switches mit der längsten Verarbeitungszeit über den NETCONF-Client und den NETCONF-Server übermittelt. Im NetConfDataStoreManagerBase wird der Zeitstempel mit der aktuellen Zeit des Switches verglichen. Die NetConfDataStoreManagerBase meldet zurück, dass der Zeitstempel zur Commit-Ausführung von diesem Switch verpasst wurde. Die Rückmeldung wird an die TimesynchronousTransactionApp übermittelt und es werden die Mechanismen Zurücksetzen aus Unterabschnitt 5.2.5 und die Entsperrphase aus Unterabschnitt 5.2.4 eingeleitet.

Der zweite Testfall beinhaltet das Verpassen des Zeitstempels von der TimesynchronousTransactionApp. Für diesen Testfall wurde eine neues Modul eingeführt, welches eine Verzögerung der aus Abbildung 5.12 stammenden M13 Nachricht von einem beteiligten Switch zum SDN-Controller verursacht. Die TimesynchronousTransactionApp erhält die verzögerte Rückmeldung von einem beteiligten Switch zurück und vergleicht den zuvor ermittelten Zeitstempel mit seiner aktuellen Zeit. Das Resultat dieses Vergleiches ist, dass die TimesynchronousTransactionApp durch die verzögerte Rückmeldung die zeitsynchrone Transaktion beendet und das Zurücksetzen und die Entsperrphase einleitet.

Auch bei diesen beiden Testfällen wurden die Anforderungen **NFA1 - Fehlertoleranz** aus Abschnitt 4.3 und **FA3 - Konsistenz** aus Abschnitt 4.2 erneut erfüllt.

7 Evaluation

In diesem Kapitel werden auf die Ergebnisse aus den Messungen der vier Arten der Netzwerkrekonfiguration eingegangen. Die vier Arten der Netzwerkrekonfiguration wurden in Abschnitt 5.5 beschrieben. Das Ziel ist es herauszufinden, ob das zeitsynchrone Transaktionsmodell den in Kapitel 4 aufgestellten Anforderungen gerecht wird. Zunächst wird der Aufbau des zu evaluierenden Netzwerkes erläutert und danach folgen die Messungen in Bezug auf Latenz und Paketverlust. Zum Schluss werden auf offene Falllstudien des zeitsynchronen Transaktionsmodells eingegangen.

7.1 Aufbau des zu evaluierenden Netzwerkes

Für die Evaluation wurde das in Abbildung 7.1 dargestellte Netzwerk verwendet. In Ab-

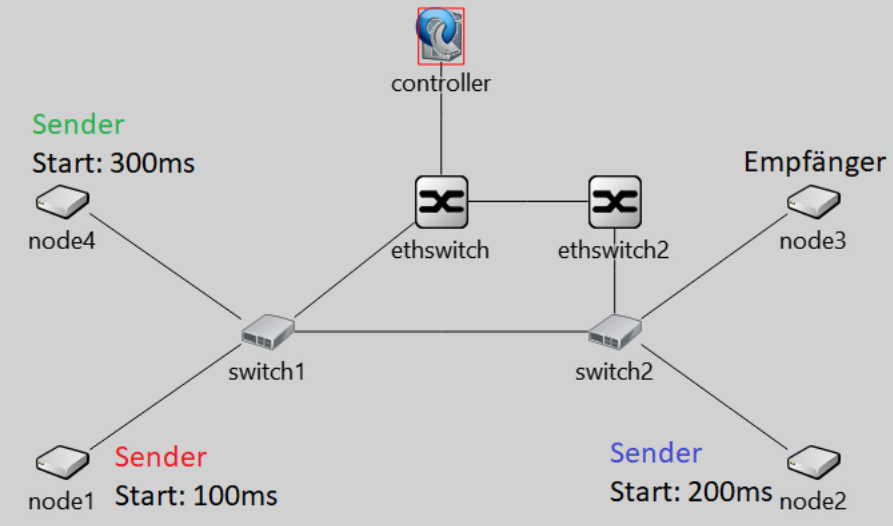

Abbildung 7.1: Das Evaluationsnetzwerk

bildung 7.1 fungieren Node 1, 2 und 4 als Sender von Datenpaketen mit der Priorität

7 und node3 als Empfänger von Datenpaketen. Für den weiteren Verlauf der Evaluation wird Datenpaket mit Priorität 7 oder Datenpakete mit Priorität 7 als Datenpaket oder Datenpakete bezeichnet. Die Sender Node 1, 2 und 4 senden jeweils Datenpakete in maximaler Größe zu verschiedenen Startzeiten an Node 3. Die Startzeiten sind aus Abbildung 7.1 zu entnehmen. Die Hyperperiode dauert 1ms. Zwischen den Knoten Node 1, 2 und 4 und den zu konfigurierenden Switches Switch 1 und 2 sind jeweils ein 10 Meter langer Kabel verlegt mit einer Übertragungsgeschwindigkeit von 100 Mbit/s. Switch 1 und 2 haben eine Verarbeitungszeit von 3µs. Datenpakete von Node 1 und 4 überqueren Switch 1 und 2, wohingegen Datenpaketen von Node 2 nur Switch 2 überqueren.

In Gleichung 7.1 wurden die erwarteten Latenzen der Datenpakete von Node 1 (L_{node1}) und 4 (L_{node4}) errechnet.

$$L_{node1} = L_{node4} = 3 * 122,45\mu s + 2 * 3\mu s = 373,35\mu s \tag{7.1}$$

In Gleichung 7.2 wurde die erwartete Latenz der Datenpakete von Node 2 (L_{node2}) errechnet.

$$L_{node2} = 2 * 122,45\mu s + 1 * 3\mu s = 247,9\mu s \tag{7.2}$$

Diese resultieren aus der Anzahl der Übertragungszeit von Datenpaketen und der Verarbeitungszeit eines Datenpaketes auf Switch 1 oder 2. Switch 1 und 2 sind über die Ethernet-Switches (ethSwitch und ethSwitch2) mit dem SDN-Controller verbunden. Es ist anzumerken, dass beide Ethernet-Switches verschiedene Verarbeitungszeiten enthalten. Der Ethernet-Switch ethSwitch hat eine Verarbeitungszeit von 8µs und ethSwitch2 hat eine Verarbeitungszeit von 1ms. Im SDN-Controller befindet sich die Anwendung, mit der die Switches und das Netzwerk rekonfiguriert werden sollen. Hierbei kann es sich um eine der vier Arten der Netzwerkrekonfiguration aus Abschnitt 5.5 handeln. Die nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfiguration sendet immer nur eine NETCONF-Operation zu einem konfigurierenden Switch. In Tabelle 7.1 sind die Änderungen aufgelistet, die jeweils auf einem Switch durchgeführt werden. Die nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfiguration beauftragt zu verschiedenen Zeitpunkten eine Änderung mit der edit-Config-Operation. Im Gegensatz dazu wurden für die nicht-zeitsynchrone Transaktion, die zeitsynchron Transaktion und die zeitsynchrone Transaktion zum Start der Hyperperiode in Tabelle 7.2 erstellt, die aus den aufgelisteten Änderungen in Tabelle 7.1 resultieren.

Kürzel	Switches	Änderungen	Beauftragung
A1	Switch 1 und Switch 2	Flusseintrag in Flusstabelle	100ms
A2	Switch 1 und Switch 2	Modifikation der GCLs	101ms
A3	Switch 2	Flusseintrag in Flusstabelle	200ms
A4	Switch 2	Modifikation der GCL	201ms
A5	Switch 1 und Switch 2	Flusseintrag in Flusstabelle	300ms
A6	Switch 1 und Switch 2	Modifikation der GCLs	301ms

Tabelle 7.1: Die Änderungen für nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfigurationen

Kürzel	Transaktion	Kürzel aus Tabelle 7.1	Beauftragung
TA1	Transaktion 1	A1 und A2	100ms
TA2	Transaktion 2	A3 und A4	200ms
TA3	Transaktion 3	A5 und A6	300ms

Tabelle 7.2: Die zusammengestellten Transaktionen aus den Änderungen von Tabelle 7.1

Berechnung des Netzwerkschedules

In Tabelle 7.3 sind die Zustände der GCL von Switch 1 und Switch 2 aus Abbildung 7.1 und wie die GCL ihre Zustände innerhalb der Hyperperiode verändern. Gleichzeitig visualisiert die Tabelle, ab welchen Transaktionen das Übertragungszeitfenster des Gates mit der Priorität 7 wie lange offen ist.

Es wird Tabelle 7.3 beschrieben. Für Switch 1 aus Abbildung 7.1 können ab der Durchführung von TA1 aus Tabelle 7.2 von 0µs bis 13µs nur Datenpakete, die nicht mit der

Switch	ab TA1	ab TA2	ab TA3	Zustand der GCL
Switch 1	0µs bis 13µs		0µs bis 13µs	o,o,o,o,o,o,o,C
	13µs bis 134µs		13µs bis 134µs	C,C,C,C,C,C,C,C
	134µs bis 148µs		134µs bis 273µs	C,C,C,C,C,C,C,o
	ab 148µs		ab 273µs	o,o,o,o,o,o,o,C
Switch 2	0µs bis 141µs	0µs bis 13µs	0µs bis 13µs	o,o,o,o,o,o,o,C
	141µs bis 260µs	13µs bis 134µs	13µs bis 134µs	C,C,C,C,C,C,C,C
	260µs bis 276µs	134µs bis 276µs	134µs bis 401µs	C,C,C,C,C,C,C,o
	ab 276µs	ab 276µs	ab 401µs	o,o,o,o,o,o,o,C

Tabelle 7.3: Die Zustände der GCLs mit Zeitspannen auf Switch 1 und 2

Priorität 7 ausgestatttet sind, weitergeleitet werden. Von 13µs bis 134µs ist ein Schutzband definiert. In Unterunterabschnitt 2.2 wird auf das Schutzband eingegangen. Von 134µs bis 148µs dürfen Datenpakete weitergeleitet werden. Ab 148µs dürfen erneut nur Datenpakete, die nicht mit der Priorität 7 ausgestattet sind weitergeleitet werden. Diese Konfiguration der GCL von Switch 1 hält so lange an, bis TA3 aus Tabelle 7.2 durchgeführt wird. Hierdurch können Datenpakete von 134µs bis 273µs übertragen werden, bevor das Gate geschlossen wird.

Für Switch 2 aus Abbildung 7.1 können ab der Durchführung von TA1 aus Tabelle 7.2 von 0µs bis 141µs Datenpakete, die nicht mit der Priorität 7 ausgestattet sind, weitergeleitet werden. Von 141µs bis 260µs ist ein Schutzband definiert. Von 260µs bis 276µs dürfen Datenpakete weitergeleitet werden. Ab 276µs dürfen erneut nur Datenpakete, die nicht mit der Priorität 7 ausgestattet sind weitergeleitet werden. Diese Konfiguration der GCL von Switch 2 hält so lange an, bis TA2 aus Tabelle 7.2 durchgeführt wird. Es können durch TA2 Datenpakete von 134µs bis 276µs weitergeleitet werden, bis das Gate geschlossen wird. Diese Konfiguration der GCL von Switch 2 hält so lange wieder an, bis TA3 aus Tabelle 7.2 durchgeführt wird. Dadurch können Datenpakete von 134µs bis 401µs übertragen werden, bis das Gate geschlossen wird.

7.2 Einfluss auf die Latenzen von bestehendem und hinzugefügtem Datenverkehr

In diesem Abschnitt werden die Einflüsse und Auswirkungen auf die Latenzen von bestehendem und hinzugefügtem Datenverkehr durch die vier Arten der Netzwerkrekonfiguration aus Abschnitt 5.5 beobachtet. Aus Abbildung 7.1 entnommen wird Datenverkehr von eines der Sender zu einem festgelegten Zeitpunkt erzeugt und über die Switches zum Ziel weitergeleitet. Die nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfiguration beauftragt seine Änderungen zu den in Tabelle 7.1 aufgelisteten Zeitpunkten. Die nicht-zeitsynchrone Transaktion, die zeitsynchrone Transaktion und die zeitsynchrone Transaktion zum Start der Hyperperiode beauftragen die Transaktionen zu den in Tabelle 7.2 aufgelisteten Zeitpunkten.

Aus den Latenzmessungen wurde die Erkenntnis gewonnen, dass die Latenzen von bestehendem und hinzugefügtem Datenverkehr je nach einer Änderung durch die nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfiguration steigen. Dies ist deutlich

in Abbildung 7.2 nach den Änderungen A3 und A4 zu erkennen. In Abbildung 7.2 sind die Latenzen der Knoten Node 1, 2 und 4 aus Abbildung 7.1 zu sehen. Node 1 verur-

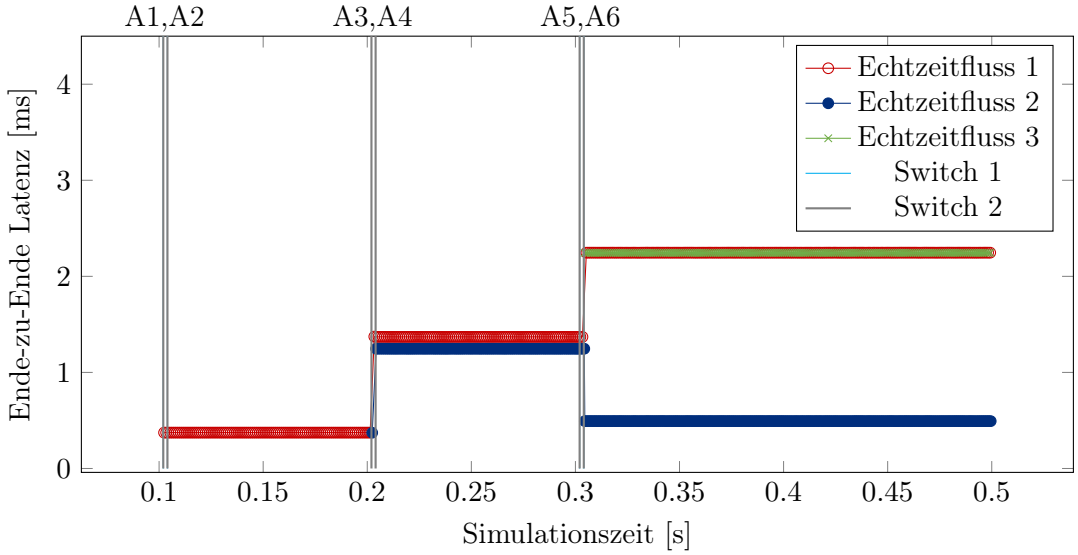

Abbildung 7.2: Die Ende-zu-Ende Latenz der nicht-zeitsynchronen und nicht-transaktionalen Netzwerkrekonfiguration

sacht die rot-markierte Latenz. Node 2 verursacht die blau-markierte Latenz und Node 4 verursacht die grün-markierte Latenz. Im Folgenden werden die Einflüsse, die durch die Änderungen A3 und A4 passieren, genauer betrachtet. Abbildung 7.3 stellt eine vergrößerte Ansicht der Zeitspanne zwischen 200ms und 210ms. Die Änderung A3 findet zum Zeitpunkt 202,016ms und die Änderung A4 findet zum Zeitpunkt 204,016ms auf Switch 2 statt. Durch A3 wird ein neuer Flusseintrag in die Flusstabelle von Switch 2 eingetragen. Dadurch können Datenpakete von Node 2 über Switch 2 weitergeleitet werden. Da die Hyperperiode 1ms beträgt, sendet jeder Sender immer ein Datenpaket innerhalb der Hyperperiode. In Abbildung 7.4 wird der Ablauf des Sendens der Datenpakete von Node 1 und 2 nach der Änderung A3 visualisiert.

- 1) Node 1 und Node 2 beginnen zum Zeitpunkt 10µs mit dem Senden eines Datenpaketes.
- 2) Das Datenpaket von Node 1 wird an Switch 1 übertragen und das Datenpaket von Node 2 wird an Switch 2 übertragen.

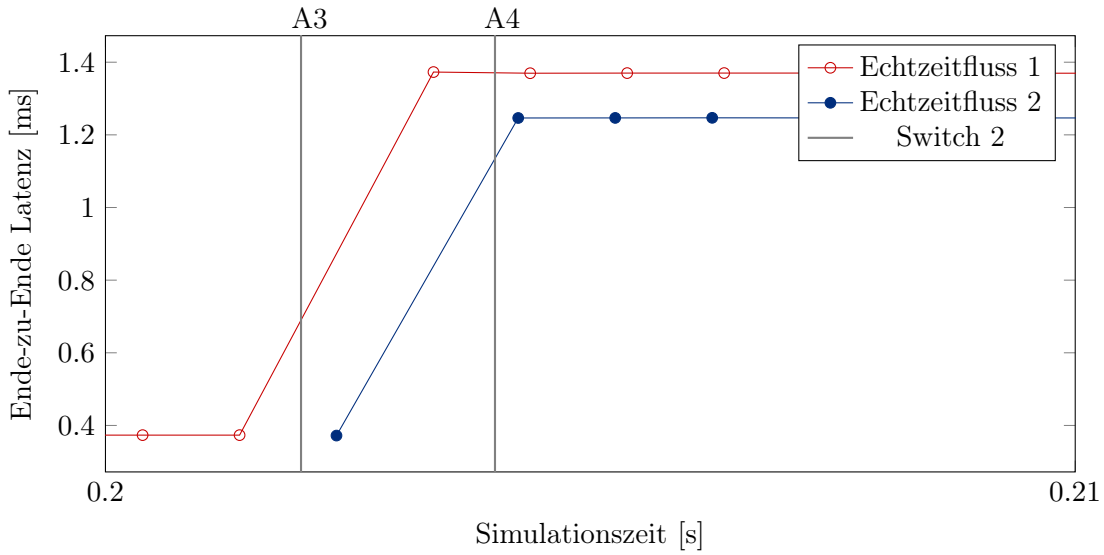

Abbildung 7.3: Die erhöhte Latenzen durch die Änderungen A3 und A4 nach der nicht-zeitsynchronen und nicht-transaktionalen Netzwerkrekonfiguration

- 3) Das Datenpaket von Node 1 wird weiter an Switch 2 übertragen während das Datenpaket von Node 2 in der Queue von Gate 7 auf Port 2 auf Switch 2 abgelegt wird, da das Gate 7 geschlossen ist.
- 4) Das Gate 7 auf Port 2 von Switch 2 ist offen und es wird nun das Datenpaket von Node 2 weitergeleitet. Da das Gate 7 nur ein Datenpaket weiterleiten kann, muss das Datenpaket von Node 1 in der Queue von Gate 7 auf Port 2 von Switch 2 verweilen. Erst zur nächsten Hyperperiode wird das Datenpaket von Node 1 weitergeleitet.
- 5) Das Gate 7 auf Port 2 von Switch 2 ist nun offen und das Datenpaket von Node 1 wird zu Node 3 weitergeleitet.

Es entsteht die angesammelte Latenz von Echtzeitfluss 2 ($AngL_{RT2}$) in Gleichung 7.3.

$$AngL_{RT2} = 122,45\mu s + 3\mu s + (260\mu s - 135,45\mu s) = 250\mu s \tag{7.3}$$

Nachdem das Gate geöffnet wurde, wird das Datenpakete von Node 2 weitergeleitet. Folgende resultierende Latenz von Echtzeitfluss 2 ($ResL_{RT2}$) wurde in Gleichung 7.4

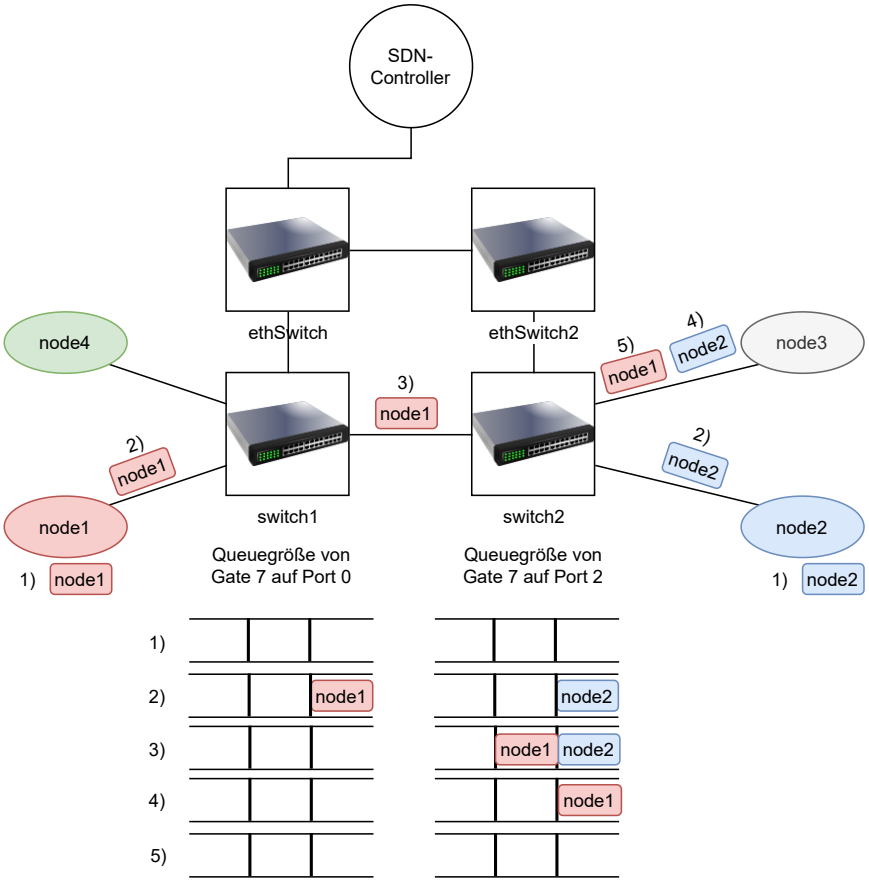

Abbildung 7.4: Die Verzögerung der Datenpaketen von Node 1 und Node 2

gebildet.

$$ResL_{RT2} = AngL_{RT2} + 122,45\mu s = 372,45\mu s \tag{7.4}$$

Das Verhältnis zwischen erwarteter Latenz aus Gleichung 7.2 und resultierender Latenz aus Gleichung 7.4 von Echtzeitfluss 2 liegt bei 124,45µs.

Für Echtzeitfluss 1 hingegen entsteht die angesammelte Latenz ($AngL_{RT1}$) in Gleichung 7.5.

$$AngL_{RT1} = 2 * 122,45\mu s + 2 * 3\mu s + (1000\mu s - 260,9\mu s) + 134\mu s = 1000\mu s \tag{7.5}$$

Das Datenpakete von Node 1 wird 1000µs bzw. 1ms in der Queue gelagert, bis das Gate wieder geöffnet wird. Folgende resultierende Latenz von Echtzeitfluss 1 ($ResL_{RT1}$) wurde

in Gleichung 7.6 gebildet.

$$ResL_{RT1} = AngL_{RT1} + 122,45\mu s = 1373,35\mu s \tag{7.6}$$

Das Verhältnis zwischen erwarteter Latenz aus Gleichung 7.1 und resultierender Latenz aus Gleichung 7.6 von Echtzeitfluss 1 liegt bei 1000µs.

Durch die Änderung A4 wird die GCL von Switch 2 verändert. Dadurch wird das Zeitfenster des Gates um die Übertragungszeit eines weiteren Datenpaketes vergrößert. Durch die resultierenden Latenzen, die nach A3 entstanden sind und durch die Veränderung der GCL, die nach A4 entstanden ist, bleiben die Latenzen von Echtzeitfluss 1 und 2 nach A4 konstant. Dies geschieht, da Datenpakete zu Beginn der Übertragung bereits in den Warteschlagen auf dem jeweiligen Port vorhanden sind.

Zusammengefasst kann über das erläuterte Phänomen, welches in Abbildung 7.3 entstanden ist, folgende Aussagen gemacht werden. Durch das unkoordinierte Verändern der Netzwerkkonfiguration durch die nicht-zeitsynchrone und nicht-transaktionale Netzwerkrekonfiguration verpassen bereits übertragene Datenpakete ihren Übertragungszeitschlitz, da neu hinzukommende Datenpakete diese übernehmen. Außerdem führen diese unkoordinierten Veränderungen dazu, dass Datenpakete in Warteschlagen abgelegt werden und nur dann weitergeleitet werden, wenn bereits übertragene Datenpakete ihr Ziel erreicht haben und das Gate wieder offen ist.

Das selbe Phänomen spiegelt sich auch in Abbildung 7.5 ab 300ms wieder. Die Änderung A5 findet auf Switch 1 zum Zeitpunkt 302,011ms und auf Switch 2 zum Zeitpunkt 302,018ms statt. Die Änderung A6 findet auf Switch 1 zum Zeitpunkt 304,011ms und auf Switch 2 zum Zeitpunkt 304,018ms statt. Durch A5 können Datenpakete von Node 4 über Switch 1 und Switch 2 weitergeleitet werden.

In Abbildung 7.6 ist die Verzögerung des Datenpaketes von Node 4 nach der Änderung A5 visualisiert. Zu dieser Zeit befinden sich 2 Datenpakete in Gate 7 auf Port 2 von Switch 2.

- 1) Node 4 beginnt mit dem Senden eines Datenpaketes zum Zeitpunkt 134µs.
- 2) Während das Datenpaket von Node 4 übertragen wird befinden sich in den Queues auf Gate 7 von Switch 1 ein Datenpaket von Node 1 und von Switch 2 zwei bereits abgelegte Datenpakete. Das verzögerte Datenpaket von Node 2 wird

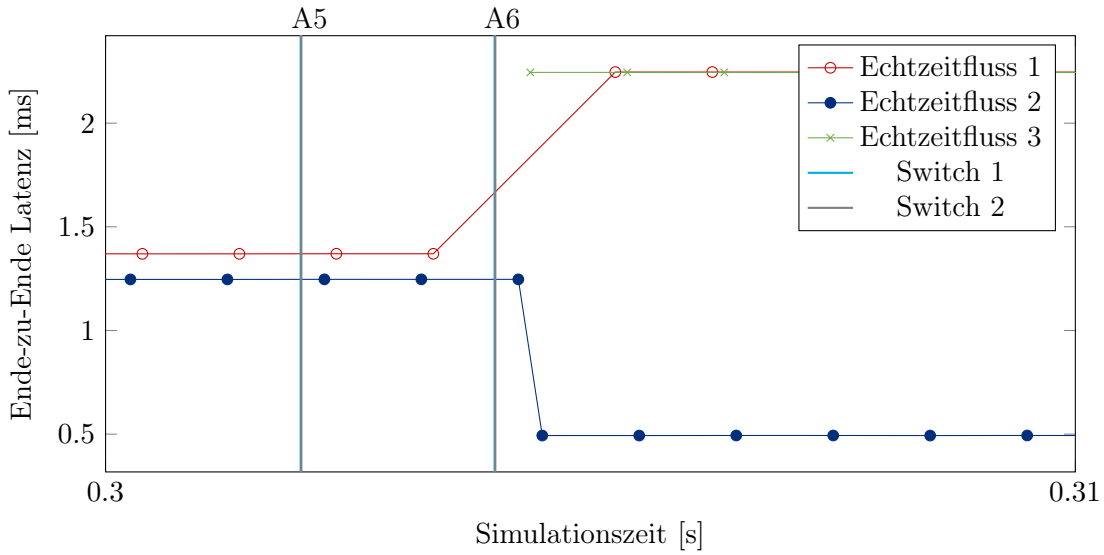

Abbildung 7.5: Die erhöhte Latenzen durch Änderungen A5 und A6 nach der nicht-zeitsynchronen und nicht-transaktionalen Netzwerkrekonfiguration

an Node 3 übertragen während ein aktuelles Datenpaket in die Queue auf Gate 7 von Switch 2 abgelegt wird.

- 3) Das Datenpaket von Node 4 wird in der Queue auf Gate 7 von Switch 1 abgelegt, da das Datenpaket von Node 1 übertragen wird und in der Queue von Gate 7 auf Switch 2 abgelegt wird. Bis zur nächsten Hyperperiode wird das Datenpaket von Node 4 in der Queue sein.
- 4) Da das Gate 7 auf Port 0 von Switch 1 offen ist wird das Datenpaket von Node 4 zu Switch 2 weitergeleitet. Bevor das Datenpaket von Node 4 am Ausgangsport von Switch 2 ankommt, hat bereits ein Datenpaket von Node 2 das Datenpaket von Node 4 überholt.
- 5) Das Datenpaket von Node 4 muss erneut um eine Hyperperiode verzögert werden, da nur zwei Datenpakete über das Gate 7 auf Port 2 von Switch 2 weitergeleitet werden dürfen.
- 6) Das Gate 7 auf Port 2 von Switch 2 wird geöffnet und das Datenpaket von Node 4 kann zu Node 3 weitergeleitet werden.

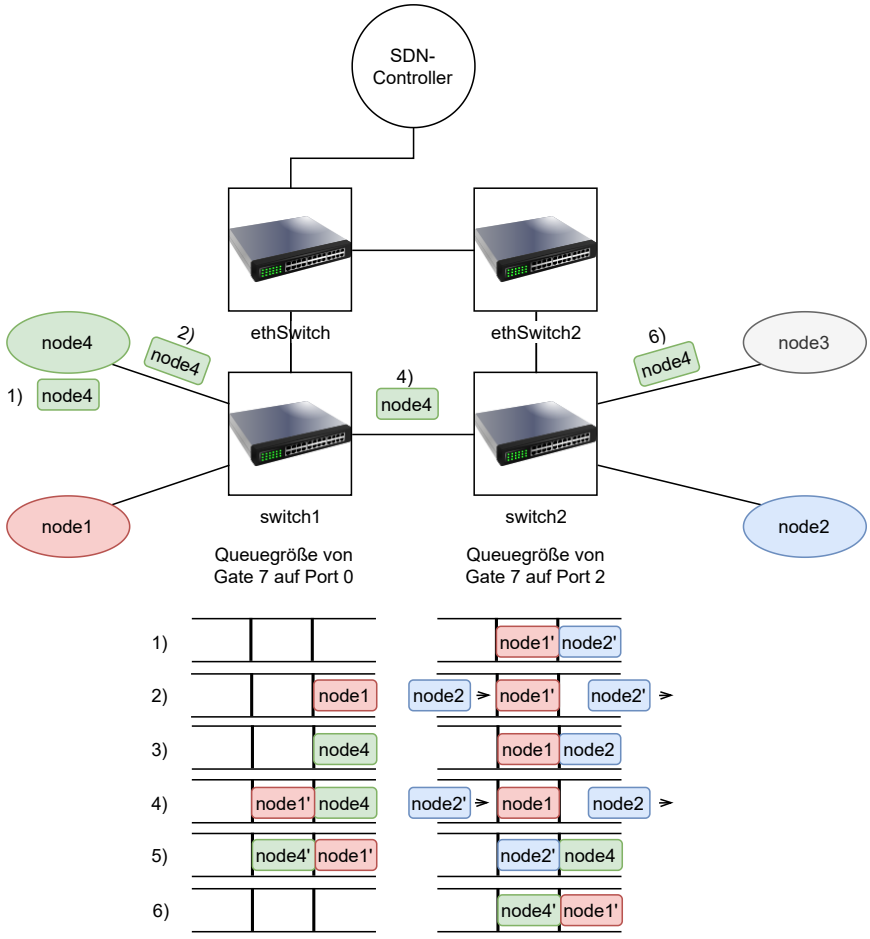

Abbildung 7.6: Die Verzögerung des Datenpaketes von Node 4

Es entsteht die in Gleichung 7.7 erste angesammelte Latenz für das Datenpaket von Node 4 ($AngL_{RT3}I$).

$$AngL_{RT3}I = 122,45\mu s + 3\mu s + (1000\mu s - 259,45\mu s) + 134\mu s = 1000\mu s \tag{7.7}$$

Die in Gleichung 7.8 zweite angesammelte Latenz für das Datenpaket von Node 4 ($AngL_{RT3}II$) wird folglich errechnet.

$$AngL_{RT3}II = AngL_{RT3}I + 122,45\mu s + 3\mu s + 997\mu s = 2122,45\mu s \tag{7.8}$$

Folgende resultierende Latenz von Echtzeitfluss 3 ($ResL_{RT3}$) wurde in Gleichung 7.9 gebildet.

$$ResL_{RT3} = AngL_{RT3}II + 122,45\mu s = 2244,9\mu s \tag{7.9}$$

Das Verhältnis zwischen erwarteter Latenz aus Gleichung 7.1 und resultierender Latenz aus Gleichung 7.9 von Echtzeitfluss 3 liegt bei 1871,55µs. Für Echtzeitfluss 1 trifft dieser Effekt auch zu. Die resultierende Latenz von Echtzeitfluss 2 hat sich nach A5 und A6 jedoch verbessert. Dies liegt daran, dass Datenpakete von Node 2 viel eher am Ausgangsport von Switch 2 gelangen und erneut den Übertragungszeitschlitz von bestehendem Datenverkehr übernehmen.

Im Gegensatz dazu entsprechen die Transaktionen aus Tabelle 7.2, die durch die nicht-zeitsynchrone Transaktion, die zeitsynchrone Transaktion und die zeitsynchrone Transaktion zum Start der Hyperperiode verursacht werden, den erwarteten Latenzen aus Gleichung 7.1 und Gleichung 7.2. Die Latenzen bleiben, wie in Abbildung 7.7, Abbildung 7.8 und Abbildung 7.9 dargestellt, bei allen drei konstant. Durch diese Erkenntnis kann gesagt werden, dass die Anforderung **NFA2 - Verzögerung** aus Abschnitt 4.3 erfüllt wird. In allen drei Transaktionsmodellen werden bestehende und hinzukommende Datenpakete im Netzwerk nicht verzögert. Der Unterschied zwischen der nicht-zeitsynchronen Transaktion, der zeitsynchronen Transaktion und zeitsynchronen Transaktion zum Start der Hyperperiode Hyperperiode liegt beim Zeitpunkt der Commit-Ausführung auf den Switches. Es ist aus Tabelle 7.4 zu entnehmen, dass die Commit-Ausführung auf allen Switches durch die zeitsynchrone Transaktion oder durch die zeitsynchrone Transaktion zum Start der Hyperperiode mit einer geringen Abweichung im Mikrosekundenbereich gleichzeitig ist. Jedoch entspricht dies nicht für die nicht-zeitsynchrone Transaktion, da bei dieser Art der Netzwerkrekonfiguration eine Abweichung im Millisekundenbereich zu sehen ist. Dadurch wird die Anforderung **FA4 - Zeitsynchronität** aus Abschnitt 4.2 von der nicht-zeitsynchronen Transaktion, der zeitsynchronen Transaktion und der zeitsynchronen Transaktion zum Start der Hyperperiode erfüllt.

Paketverlust der vier Arten der Netzwerkrekonfiguration

Das Ziel bei dieser Messung bestand darin, den Paketverlust der Switches aus Abbildung 7.1 zu ermitteln. Es wurde festgestellt, dass bei der nicht-zeitsynchronen Transaktion ein Datenpaket verloren gegangen ist. Wie aus Tabelle 7.4 zu entnehmen ist, ha-

Switch	Transaktion	Nicht-zeitsynchroner Commit	Zeitsynchroner Commit	Zeitsynchroner Commit zum Start der Hyperperiode
Switch 1	TA1	131,154ms	143,345ms	143,999ms
Switch 2	TA1	132,156ms	143,347ms	144ms
Switch 2	TA2	229,129ms	237,239ms	238ms
Switch 1	TA3	331,153ms	344,344ms	343,999ms
Switch 2	TA3	332,155ms	343,346ms	344ms

Tabelle 7.4: Die Commit-Ausführungszeitpunkte aus (nicht-)zeitsynchronen Transaktionen (zum Start der Hyperperiode)

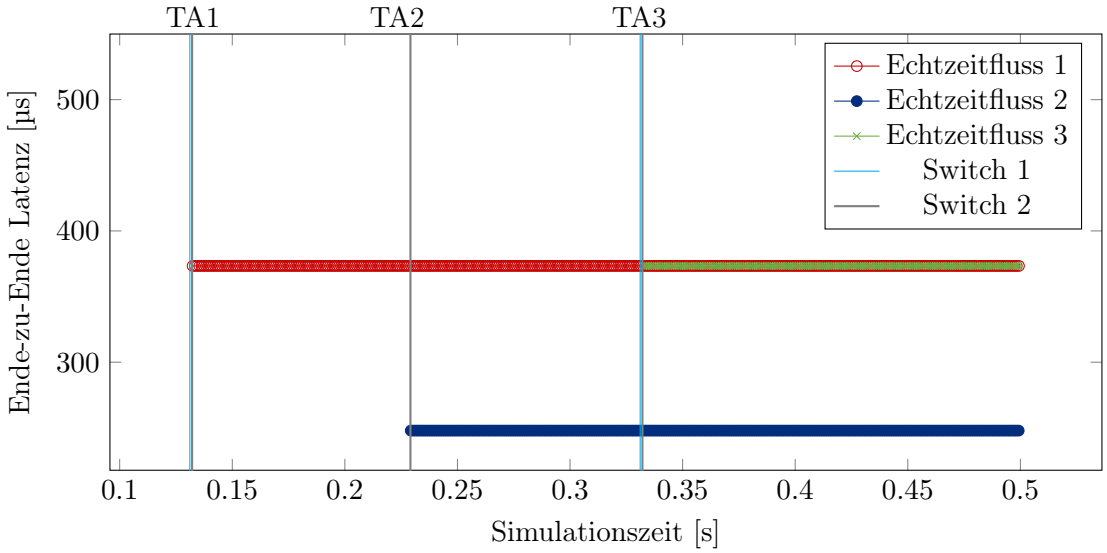

Abbildung 7.7: Die Ende-zu-Ende Latenz aus der nicht-zeitsynchronen Transaktion

ben die Switches bei der Commit-Ausführung der Transaktionen TA1 und TA3 aus der nicht-zeitsynchronen Transaktion eine Abweichung von mindestens 1ms. Die Commit-Ausführung der nicht-zeitsynchronen Transaktion findet nicht zu dem exakt selben Zeitpunkt statt, wodurch das Netzwerk in Inkonsistenz verfällt. Im Gegensatz dazu entsteht kein Paketverlust bei bestehendem sowie hinzukommendem Datenverkehr in zeitsynchronen Transaktionen. Es wird durch die Worst-Case-Berechnung entweder zeitsynchron innerhalb der Hyperperiode oder zum nächstmöglichen Start der Hyperperiode die Commit-Ausführung auf allen Switches gleichzeitig vollzogen. Hierdurch erfüllen zeitsynchrone Transaktionen die Anforderung **NFA3 - Paketverlust** aus Abschnitt 4.3.

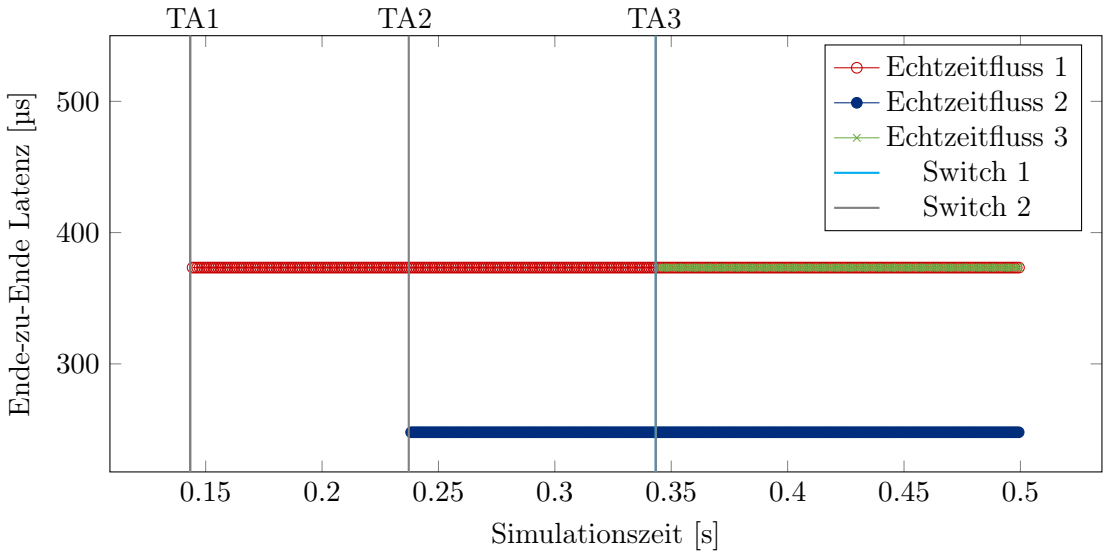

Abbildung 7.8: Die Ende-zu-Ende Latenz aus der zeitsynchronen Transaktion

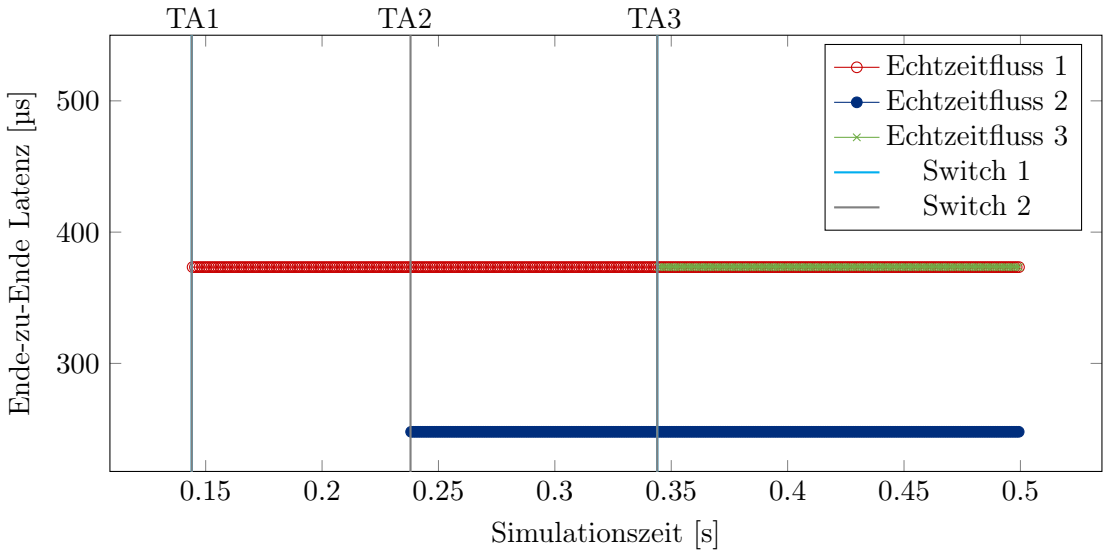

Abbildung 7.9: Die Ende-zu-Ende Latenz aus der zeitsynchronen Transaktion zum Start der Hyperperiode

7.3 Offene Fallstudien

Die bisher durchgeführten Messungen in dieser Evaluation haben sich darauf fokussiert, dass das Übertragungsfenster von Datenpaketen in der GCL vergrößert und dass Fluss-

einträge in der Flusstabelle von Switches hinzugefügt werden. Jedoch können in ferner Zukunft noch weitere Messungen zu anderen Fallstudien durchgeführt werden, um die Unterschiede von zeitsynchronen Transaktionen gegenüber von zeitsynchronen Transaktionen zum Start der Hyperperiode zu stellen. Eine mögliche Fallstudie kann das Entfernen von Flusseinträgen in der Flusstabelle und damit das Verkleinern des Übertragungsfensters der GCL der Switches sein. Das hätte zur Folge, dass Datenpakete nach einer gewissen Zeit nicht weiter übertragen werden. In dieser Fallstudie könnte überprüft werden, ob nach einer zeitsynchronen Transaktion übrige Datenpakete weitergeleitet werden oder aber bereits vollständig übertragen wurden. Ein weiteres mögliche Fallstudie kann das Verschieben von Übertragungsfenstern der GCL der Switches sein. Hierdurch würden Datenpakete zu anderen Zeitpunkten übertragen werden. Aus den beiden möglichen benannten Fallstudien könnte auch das Umleiten von Flüssen als eine weitere Fallstudie realisiert werden. Dies wäre eine Kombination der vorherigen Fallstudien, in denen Flusseinträge in den Flusstabellen entfernt und hinzugefügt, sowie in den GCLs die Übertragungsfenster verkleinert und vergrößert werden. Datenpakete hätten für eine bestimmte Zeit ein Host als Ziel, welches dann durch die zeitsynchrone Transaktion zu einem anderen Host umgeleitet werden. Neben diesen Fallstudien können weitere Messungen durchgeführt werden, in denen Switches mit anderen implementierten TSN-Standards durch zeitsynchrone Transaktionenen rekonfiguriert werden, um deren Auswirkungen und Effekte auf das Netzwerk zu beobachten.

8 Fazit und Ausblick

Dieses Kapitel schließt die vorliegende Arbeit mit Fazit und Ausblick. Zunächst folgt eine kurze Zusammenfassung der Arbeit. Danach werden auf die bisherigen Ergebnisse eingegangen. Zum Schluss werden im Ausblick mögliche weiterführende Arbeiten vorgestellt.

8.1 Fazit

In dieser Arbeit wurde ein zeitsynchrones Transaktionsmodell konzeptioniert und in der Simulationsumgebung OMNeT++ implementierten und evaluiert. Das zeitsynchrone Transaktionsmodell ist aus den Ideen und Ansätze bisheriger Protokolle konzeptioniert. Nachdem die Anforderungen an das zeitsynchrone Transaktionsmodell ermittelt wurden, konnte ein Konzept zur Umsetzung dieser Anforderungen erstellt werden. Es wurden Testfälle vorgestellt, um zu prüfen, ob das erarbeitete zeitsynchrone Transaktionsmodell die Anfoderungen erfüllen kann. Das Ziel war es, die Effekte und die Auswirkungen des zeitsynchronen Transaktionsmodells in einem programmierbaren und zeitkritischen Netzwerk zu evaluieren.
Aus den bisherigen Messungen konnte ermittelt werden, dass das zeitsynchrone Transaktionsmodell die gestellten Anforderungen in Kapitel 4 erfüllt. Das zeitsynchrone Transaktionsmodell führt zeitsynchrone Transaktionen aus, die atomar sind und das Netzwerk von einem konsistenten Zustand in einen neuen konsistenten Zustand überführen. Die Commit-Ausführung auf den Switches ist zeitsynchron und hat eine Abweichung im Mikrosekundenbereich. Die Fallstudie aus der Evaluation hat gezeigt, dass bestehender und hinzukommender Datenverkehr nicht verzögert wird und keinen Paketverlust aufweist.

8.2 Ausblick

Als Fortsetzung dieser Arbeit sind die noch offenen Fallstudien durchzuführen. Eine Fallstudie kann sein, dass bestehender Datenverkehr aus dem Netzwerk entfernt wird, wodurch Flusseinträge verworfen und Übertragungszeitschlitze auf das Gate mit der jeweiligen Priorität verkleinert werden. Darüber hinaus kann eine weitere Fallstudie sein, dass Übertragungszeitschlitze von bestehendem Datenverkehr verschoben wird, wodurch Datenpakete zu anderen Zeitpunkten übertragen werden kann. Außerdem kann eine weitere Fallstudie sein, dass das Umleiten von bestehendem Datenverkehr gemessen wird, in dem Flusseinträge und Übertragungszeitschlitze entfernt, hinzugefügt sowie verkleinert und vergrößert werden. Neben den bisher möglichen Arbeiten kann auch das Konzept des Transaktionsmodells erweitert werden, indem weitere Eigenschaften von Transaktionen etabliert werden. So wäre die Isolationseigenschaft eine Möglichkeit, das Transaktionsmodell zu erweitern, um nebenläufige zeitsynchrone Transaktionen in zeitkritischen und programmierbaren Netzwerken umzusetzen und deren Auswirkungen zu beobachten.

Literaturverzeichnis

[1] OpenFlow Switch Specification. März 2015 (ONF TS-025). – Forschungsbericht

[2] Zhang, P. ; Liu, Y. ; Shi, J. ; Huang, Y. ; Zhao, Y.: A Feasibility Analysis Framework of Time-Sensitive Networking Using Real-Time Calculus. In: *IEEE Access* 7 (2019), July, S. 90069–90081. – ISSN 2169-3536

[3] Bjorklund, M.: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) / IETF. October 2010 (6020). – RFC

[4] CoRE-Arbeitsgruppe: *Communication over Real-time Ethernet*. – URL https://core-researchgroup.de/. – Zugriffsdatum: 2021-06-29

[5] CoRE-Arbeitsgruppe: *CoRE Simulation Models for Real-time Networks*. – URL http://sim.core-rg.de. – Zugriffsdatum: 2021-06-29

[6] Cui, J. ; Zhou, S. ; Zhong, H. ; Xu, Y. ; Sha, K.: Transaction-Based Flow Rule Conflict Detection and Resolution in SDN. In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, July 2018, S. 1–9

[7] Curic, M. ; Despotovic, Z. ; Hecker, A. ; Carle, G.: Transactional Network Updates in SDN. In: *2018 European Conference on Networks and Communications (EuCNC)*, 2018, S. 203–208

[8] Enns, R. ; Bjorklund, M. ; Schoenwaelder, J. ; Bierman, A.: Network Configuration Protocol (NETCONF) / IETF. June 2011 (6241). – RFC

[9] Gutiérrez, M. ; Ademaj, A. ; Steiner, W. ; Dobrin, R. ; Punnekkat, S.: Self-configuration of IEEE 802.1 TSN networks. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, S. 1–8

[10] Häckel, Timo ; Meyer, Philipp ; Korf, Franz ; Schmidt, Thomas C.: SDN4CoRE: A Simulation Model for Software-Defined Networking for Communication over Real-Time Ethernet. In: Zongo, Meyo (Hrsg.) ; Virdis, Antonio (Hrsg.) ; Vesely, Vladimir (Hrsg.) ; Vatandas, Zeynep (Hrsg.) ; Udugama, Asanga (Hrsg.) ; Kuladinithi, Koojana (Hrsg.) ; Kirsche, Michael (Hrsg.) ; Förster, Anna (Hrsg.): *Proceedings of the 6th International OMNeT++ Community Summit 2019* Bd. 66, EasyChair, Dezember 2019, S. 24–31. – URL https://easychair.org/publications/paper/1TnZ. – ISSN 2398-7340

[11] Häckel, Timo ; Meyer, Philipp ; Korf, Franz ; Schmidt, Thomas C.: Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. Piscataway, NJ, USA : IEEE Press, April 2019, S. 1–5. – ISSN 1090-3038

[12] IEEE 802.1 Working Group: IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic / IEEE. March 2016 (Std 802.1Qbv-2015). – Forschungsbericht. – 1–57 S. ()

[13] IEEE 802.1 Working Group: IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks / IEEE. Juli 2018 (Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)). – Standard. – 1–1993 S

[14] IEEE 802.1 Working Group: IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements / IEEE. June 2018 (Std 802.1Qcc-2018). – Forschungsbericht. – 1–208 S. (Amendment to IEEE Std 802.1Q-2011)

[15] IEEE 802.3 Working Group: IEEE Standard for Ethernet / IEEE. August 2018 (Std 802.3-2018). – Forschungsbericht. – 1–5600 S. (Revision of IEEE Std 802.3-2015)

[16] Klein, Dominik ; Jarschel, Michael: An OpenFlow extension for the OMNeT++ INET framework. In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques* ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (Veranst.), 2013, S. 322–329

[17] Kreutz, D. ; Ramos, F. M. V. ; Veríssimo, P. E. ; Rothenberg, C. E. ; Azodolmolky, S. ; Uhlig, S.: Software-Defined Networking: A Comprehensive Survey. In: *Proceedings of the IEEE* 103 (2015), Januar, Nr. 1, S. 14–76. – ISSN 0018-9219

[18] Mandl, P.: *Masterkurs Verteilte betriebliche Informationssysteme.* Vieweg+Teubner Verlag, 2009

[19] Nayak, N. G. ; Dürr, F. ; Rothermel, K.: Software-defined environment for reconfigurable manufacturing systems. In: *2015 5th International Conference on the Internet of Things (IOT)*, 2015, S. 122–129

[20] Nayak, N. G. ; Dürr, F. ; Rothermel, K.: Time-sensitive Software-defined Network (TSSDN) for Real-time Applications. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. New York, NY, USA : ACM, 2016 (RTNS '16), S. 193–202. – URL http://doi.acm.org/10.1145/2997465.2997487. – ISBN 978-1-4503-4787-7

[21] OMNeT++ Simulation Manual. – Forschungsbericht. Zugriffsdatum 29.12.2020

[22] Open Network Foundation: *The Open Network Foundation.* https://www.opennetworking.org. – Zugriffdatum: 02.01.2021.

[23] OpenSim Ltd.: *INET Framework.* – URL https://inet.omnetpp.org/

[24] Postel, J.: Transmission Control Protocol / IETF. September 1981 (793). – RFC

[25] Rahm, E. ; Saake, G. ; Sattler, K.-U.: *Verteiltes und Paralleles Datenmanagement.* Springer-Verlag, 2015

[26] Saake, G. ; Sattler, K.-U. ; Heuer, A.: *Datenbanken: Konzepte und Sprachen.* MITP, 2018

[27] Schill, A. ; Springer, T.: *Verteilte Systeme.* Springer-Verlag, 2012

[28] Schubert, M.: *Datenbanken.* B. G. Teubner Verlag, 2007

[29] Schönwälder, J. ; Björklund, M. ; Shafer, P.: Network configuration management using NETCONF and YANG. In: *IEEE Communications Magazine* 48 (2010), September, S. 166–173

[30] Steinbach, Till: *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil.* Wiesbaden : Springer Vieweg, October 2018. – ISBN 978-3-658-23499-7

[31] Tanenbaum, Andrew S. ; Bos, Herbert: *Moderne Betriebssysteme*. Pearson, 2015

[32] IEEE 802.1 TSN Task Group: *Time-Sensitive Networking (TSN) Task Group*. – URL https://1.ieee802.org/tsn/

[33] Gamm, Stephanie: *Transaktionen in verteilten und mobilen Systemen*. https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2006/gamm/abstract.pdf. 2006. – Zugriffsdatum 28.12.2020.

[34] YangMin Lee and JaeKee Lee: YANG-based Data Modeling Techniques for the Content Layer of NETCONF to Improve Query Throughput. In: *Journal of Internet Technology* (2018)

[35] Yoo SM., Hong J.: Performance Improvement Methods for NETCONF-Based Configuration Management. In: *Kim YT., Takano M. (eds) Management of Convergence Networks and Services. APNOMS* (2006)

A Anhang

Bezeichnung in FSM	**Eingangssignal**	**Beschreibung**
e_Switch_gesperrt	empfange Switch X gesperrt	Die Transaktion empfängt die Nachricht, dass Switch X seine laufende Konfiguration gesperrt hat.
e_bereits_belegt	empfange Fehlermeldung bereits belegt	Die Transaktion empfängt die Nachricht, dass Switch X bereits gesperrt ist.
e_Konfig_kopiert	empfange laufende Konfiguration kopiert	Die Transaktion empfängt die Nachricht, dass Switch X die laufende Konfiguration kopiert hat.
e_Kandidat_fehlgeschlagen	empfange Fehlermeldung Kandidat fehlgeschlagen	Die Transaktion empfängt die Nachricht, dass das Kopieren der laufenden Konfiguration von Switch X fehlgeschlagen ist.
e_Aenderung_durchgefuehrt	empfange Änderung von Switch X durchgeführt	Die Transaktion empfängt die Nachricht, dass alle Änderungen von Switch X erfolgreich in der Kandidat-Konfiguration durchgeführt wurden.
e_Aenderung_fehlgeschlagen	empfange Fehlermeldung Änderung an Kandidat fehlgeschlagen	Die Transaktion empfängt die Nachricht, dass eine Änderung an der Kandidat-Konfiguration von Switch X fehlgeschlagen ist.

Tabelle A.1: Die Nachrichteneingangstabelle I

Bezeichnung in FSM	**Eingangssignal**	**Beschreibung**
e_Zeitstempel_akzeptiert	empfange Zeitstempel akzeptiert	Die Transaktion empfängt die Nachricht, dass Switch X den Zeitstempel akzeptiert hat.
e_Zeitpunkt_ueberschritten	empfange Fehlermeldung Zeitstempel überschritten	Die Transaktion empfängt die Nachricht, dass Switch X den Zeitstempel überschreiten wird.
e_Commit_ausgefuehrt	empfange Commit wurde ausgeführt	Die Transaktion empfängt die Nachricht, dass Switch X zum Zeitpunkt des Zeitstempels die neue Konfiguration bestätigt hat.
e_alte_Konfig_geloescht	empfange alte Konfiguration gelöscht	Die Transaktion empfängt die Nachricht, dass Switch X die alte laufende Konfiguration gelöscht hat.
e_Switch_entsperrt	empfange Switch X entsperrt	Die Transaktion empfängt die Nachricht, dass Switch X die neue laufenden Konfiguration entsperrt hat.
e_Switch_Kandidat_sperren	empfange Kandidat-Konfigurationsdatenspeicher sperren	Switch X empfängt von der Transaktion die Nachricht, dass dieser die Kandidat-Konfiguration sperren soll.

Tabelle A.2: Die Nachrichteneingangstabelle II

Bezeichnung in FSM	Eingangssignal	Beschreibung
e_alte_Konfig_-geloescht	empfange alte Konfiguration gelöscht	Die Transaktion empfängt die Nachricht, dass Switch X die alte laufende Konfiguration gelöscht hat.
e_Switch_entsperrt	empfange Switch X entsperrt	Die Transaktion empfängt die Nachricht, dass Switch X die laufende Konfiguration entsperrt hat.
e_Kandidat_geloescht	empfange Kandidat-Konfiguration gelöscht	Die Transaktion empfängt die Nachricht, dass die Kandidat-Konfiguration von Switch X gelöscht wurde.
e_Switch_sperren	empfange Switch X sperren	Switch X wird aufgefordert die laufende Konfiguration zu sperren.
e_Konfig_kopieren	empfange laufende Konfiguration kopieren	Switch X wird aufgefordert die laufende Konfiguration zu kopieren.
e_Aenderung_-durchfuehren	empfange Änderung durchführen	Switch X wird aufgefordert die Änderung an der Kandidat-Konfiguration durchzuführen.

Tabelle A.3: Die Nachrichteneingangstabelle III

Bezeichnung in FSM	Eingangssignal	Beschreibung
e_Zeitstempel	empfange Zeitstempel	Switch X empfängt den Zeitstempel zur Commit-Ausführung.
e_Commit_freigeben	empfange Commit freigeben	Switch X empfängt die Commit-Freigabe von der Transaktion.
e_Kandidat_-loeschen	empfange Kandidat-Konfiguration löschen	Switch X wird aufgefordert die Kandidat-Konfiguration zu löschen.
e_alte_Konfig_-loeschen	empfange alte Konfiguration löschen	Switch X wird aufgefordert die alte laufende Konfiguration zu löschen.
e_Switch_entsperren	empfange Switch entsperren	Switch X wird aufgefordert die neue laufende Konfiguration zu entsperren.
e_Switch_Kandidat_gesperrt	empfange Kandidat-Konfigurationsdatenspeicher gesperrt	Die Transaktion empfängt die Nachricht von Switch X, dass die Kandidat-Konfiguration gesperrt wurde.

Tabelle A.4: Die Nachrichteneingangstabelle IV

Bezeichnung in FSM	Ausgangssignal	Beschreibung
s_Switch_sperren	sende Switch X sperren	Die Transaktion fordert von Switch X die laufende Konfiguration zu sperren.
s_Konfig_kopieren	sende laufende Konfiguration kopieren an alle Switches	Die Transaktion fordert von allen Switches die laufende Konfiguration zu kopieren.
s_Aenderung_durchfuehren	sende Änderung durchführen an alle Switches	Die Transaktion forder von allen Switches die Änderungen an der Kandidat-Konfiguration durchzuführen.
s_Zeitstempel	sende Zeitstempel an alle Switches	Die Transaktion sendet den Zeitstempel zur Commit-Ausführung an alle Switches.
s_Commit_freigeben	sende Commit freigeben an alle Switches	Die Transaktion gibt allen Switches die Freigabe zum Commit.

Tabelle A.5: Die Nachrichtenausgangstabelle I

Bezeichnung in FSM	Ausgangssignal	Beschreibung
s_alte_Konfig_loeschen	sende alte Konfiguration löschen an alle Switches	Die Transaktion fordert von allen Switches die alte laufende Konfiguration zu löschen.
s_Switch_entsperren	sende Switch entsperren an alle Switches	Die Transaktion fordert von allen Switches die ihre neue oder alte laufende Konfiguration zu entsperren.
s_Kandidat_loeschen	sende Kandidat-Konfiguration löschen an alle Switches	Die Transaktion fordert von allen Switches die Kandidat-Konfiguration zu löschen.
s_Konfig_kopiert	sende laufende Konfiguration kopiert	Der Switch teilt der Transaktion mit, dass die laufende Konfiguration kopiert wurde.
s_Kandidat_fehlgeschlagen	sende Fehlermeldung Kandidat fehlgeschlagen	Der Switch teilt der Transaktion mit, dass die Kandidat-Konfiguration fehlschlug.
s_Switch_Kandidat_sperren	sende Kandidat-Konfiguration sperren an Switch X	Die Transaktion fordert von Switch X, dass die Kandidat-Konfiguration gesperrt werden soll.

Tabelle A.6: Die Nachrichtenausgangstabelle II

Bezeichnung in FSM	Ausgangssignal	Beschreibung
s_Aenderung_durchgefuehrt	sende Änderung durchgeführt	Der Switch sendet die Nachricht, dass die Änderung an der Kandidat-Konfiguration durchgeführt wurde.
s_Aenderung_fehlgeschlagen	sende Fehlermeldung Änderung an Kandidat fehlgeschlagen	Der Switch sendet die Nachricht, dass die Änderung an der Kandidat-Konfiguration fehlgeschlagen ist.
s_Zeitstempel_akzeptiert	sende Zeitstempel akzeptiert	Der Switch sendet die Nachricht, dass der Zeitstempel akzeptiert wurde.
s_Zeitpunkt_ueberschritten	sende Fehlermeldung Zeitstempel überschritten	Der Switch sendet die Nachricht, dass der Zeitstempel überschritten wurde.
s_Switch_gesperrt	sende Switch X gesperrt	Der Switch sendet die Nachricht, dass dieser die laufende Konfiguration gesperrt hat.
s_Commit_ausgefuehrt	sende Commit wurde ausgeführt	Der Switch sendet die Nachricht, dass dieser den Commit ausgeführt hat.

Tabelle A.7: Die Nachrichtenausgangstabelle III

Bezeichnung in FSM	**Ausgangssignal**	**Beschreibung**
s_Kandidat_geloescht	sende Kandidat-Konfiguration gelöscht	Der Switch sendet die Nachricht, dass die Kandidat-Konfiguration gelöscht wurde.
s_alte_Konfig_-geloescht	sende alte Konfiguration gelöscht	Der Switch sendet die Nachricht, dass die alte laufende Konfiguration gelöscht wurde.
s_Switch_entsperrt	sende Switch entsperrt	Der Switch sendet die Nachricht, dass die neue laufende Konfiguration entsperrt wurde.
s_bereits_belegt	sende Fehlermeldung bereits belegt	Der Switch sendet die Nachricht, dass die laufenden Konfiguration bereits belegt beziehungsweise gesperrt ist.
s_Switch_Kandidat_gesperrt	sende Kandidat-Konfiguration gesperrt an Transaktion	Switch X sendet an die Transaktion, dass die Kandidat-Konfiguration gesperrt wurde.

Tabelle A.8: Die Nachrichtenausgangstabelle IV

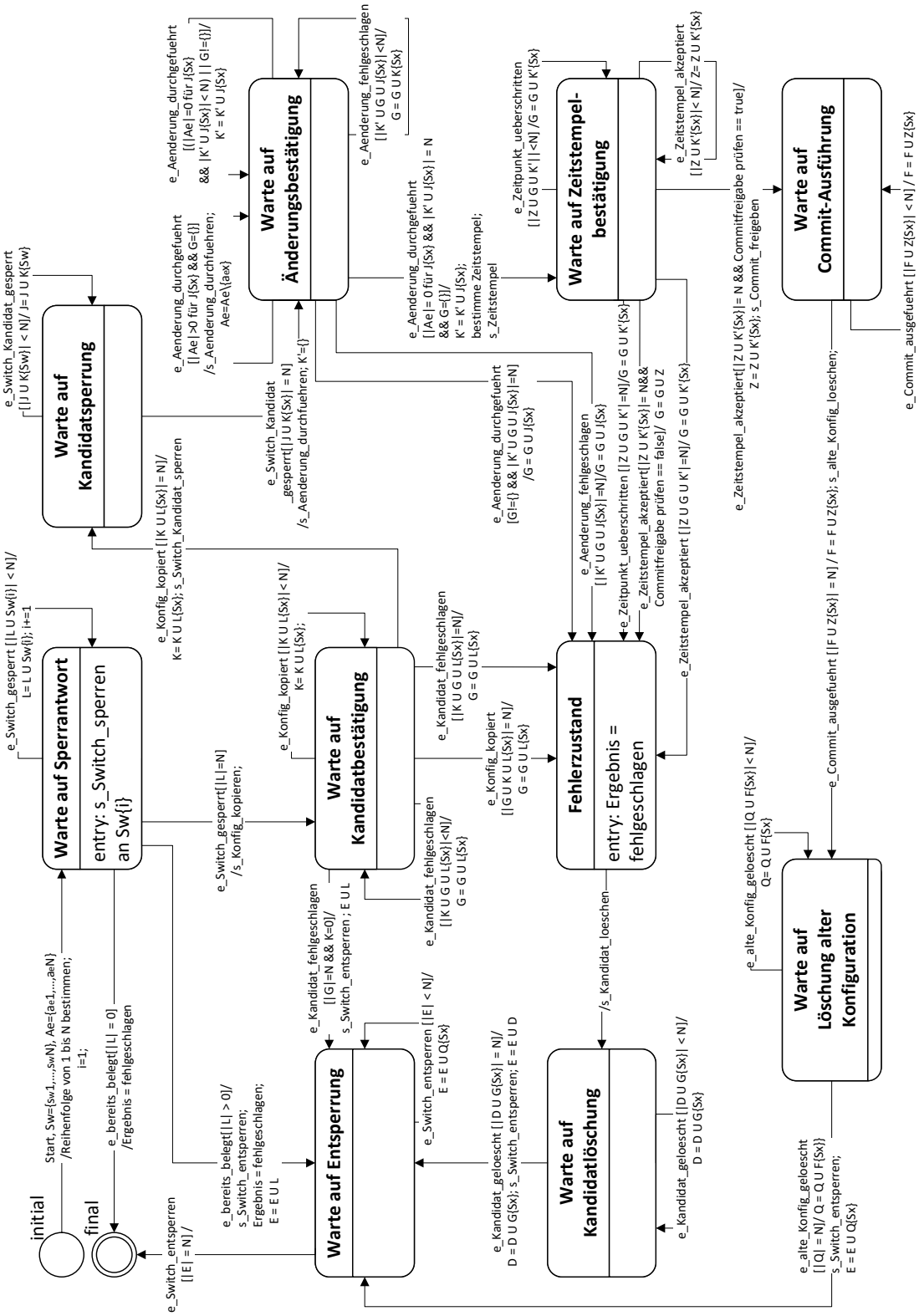

Abbildung A.1: Der endliche Automat der TimesynchronousTransactionApp

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Ein Transaktionsmodell zur Netzwerkkonfiguration mit Zeitanforderungen am Beispiel von Fahrzeugen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original