Universität Rostock

Traditio et Innovatio

Routing und Scheduling im zeitgesteuerten Ethernet

Mathematische Modellierung, Optimierung und systematische Leistungsbewertung

Dissertation

VORGELEGT VON:

Eike Björn Schweißguth,

geboren am 11.04.1990 in Oldenburg

zur Erlangung des akademischen Grades eines

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

https://doi.org/10.18453/rosdok_id00005160

UNIVERSITÄT ROSTOCK | FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK

Institut für Angewandte Mikroelektronik und Datentechnik

Albert-Einstein-Str. 26, 18059 Rostock

GUTACHER: **Dirk Timmermann**
Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Angewandte Mikroelektronik und Datentechnik
Albert-Einstein-Str. 26, 18059 Rostock

Kurt Rothermel
Universität Stuttgart
Fakultät für Informatik, Elektrotechnik und Informationstechnik
Institut für Parallele und Verteilte Systeme
Universitätsstr. 38, 70569 Stuttgart

Gero Mühl
Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik
Albert-Einstein-Str. 22, 18059 Rostock

EINGEREICHT AM: 26.11.2024
VERTEIDIGT AM: 24.04.2025

Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit, einschließlich Tabellen und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Rostock, den 26.11.2024

_____

Eike Björn Schweißguth

Zusammenfassung

Echtzeitfähige Netzwerke dienen der Kommunikation in kritischen Anwendungsbereichen, in denen Nachrichten zuverlässig innerhalb vorgegebener Latenzschranken ihre Empfänger erreichen müssen. Um derartige Anforderungen erfüllen zu können, wurde standardkonformes Ethernet in den letzten Jahren um hierfür notwendige Funktionen erweitert, wobei die entsprechenden Teilstandards unter dem Namen Time-Sensitive Networking zusammengefasst werden. Für Anwendungen mit höchsten Anforderungen an Latenz und Jitter in der Kommunikation wurden dabei in IEEE 802.1Qbv Mechanismen definiert, die das zeitgesteuerte Senden von Frames ermöglichen. Algorithmen zum Ermitteln einer sinnvollen Netzwerkkonfiguration, insbesondere eines Sendezeitplans für das zeitgesteuerte Senden, wurden dabei allerdings nicht standardisiert. Dies hat zu einem gesteigerten Forschungsinteresse an derartigen Algorithmen, auch als Scheduler bezeichnet, geführt.

Die Vielzahl von Publikationen in diesem Gebiet hat jedoch bisher einen aussagekräftigen Vergleich der darin veröffentlichten Scheduler vermissen lassen. Dies ist u.a. darauf zurückzuführen, dass bisher kein systematischer Entwurf von Testdaten für diese Scheduler stattfindet, sodass Tests mit unzureichender Abdeckung bezüglich der Netzwerk- und Verkehrsparameter durchgeführt werden. In dieser Arbeit werden daher Testcases zur systematischen Leistungsbewertung der Scheduler entworfen. Die verbesserte Testabdeckung gegenüber bisherigen Untersuchungen ermöglicht es, Stärken und Schwächen einzelner Scheduler identifizieren zu können. Die Veröffentlichung der Testdaten als Datensatz stellt deren Wiederverwendbarkeit für zukünftige Schedulervergleiche sicher, während eine präzise Beschreibung der Erzeugung der Testdaten die Nachvollziehbarkeit und Erweiterbarkeit adressiert.

Des Weiteren werden in der Arbeit Scheduler auf Basis mathematischer Modelle entworfen. Konkret wird dabei Integer Linear Programming eingesetzt, um das Planungsproblem des zeitgesteuerten Netzwerkverkehrs abzubilden und Lösungen zu finden. Entwurfsziel des Schedulers war es, den Lösungsraum bezüglich Routing und Scheduling möglichst vollständig zu erfassen, um Aussagen bezüglich der Planbarkeit gegebener Netzwerkszenarien und der Güte von gefundenen Lösungen treffen zu können, welche beim Vergleich von Schedulern als Referenzwerte nutzbar sind. Eine umfassende Optimierung dieses modellbasierten Ansatzes macht ihn außerdem auch hinsichtlich der Leistungsfähigkeit (insbesondere bezüglich Rechenzeiten) zu einem praktisch anwendbaren Verfahren zur Planung zeitgesteuerter Netzwerke. Durch die Optimierung der Modelle konnte die durchschnittliche Rechenzeit in den systematischen Testcases um mehr als 90 % reduziert werden. Spezielle Modellanpassungen ermöglichen außerdem die Erstellung von Schedules, die mit standardkonformen Bridges nach IEEE 802.1Q kompatibel sind.

Abstract

Real-time networks offer communication services for critical applications, where messages need to be delivered to their destinations reliably and within given latency bounds. In recent years, additional mechanisms were specified in the Ethernet standards in order to satisfy these requirements. The new substandards associated with these extensions are collectively referred to as Time-Sensitive Networking standards. Among these, IEEE 802.1Qbv defines mechanisms for the time-triggered transmission of frames, which enables communication with the lowest possible latencies and jitter. Yet, algorithms for finding sensible network configurations, especially regarding the schedule for time-triggered transmissions, were not defined as part of the standards. This situation led to significant research efforts regarding such scheduling algorithms.

However, the large amount of publications in this area has lacked an expressive comparison of the presented schedulers so far. In this regard, one of the major problems is the absence of a systematic design of test data for the developed schedulers, which has led to evaluations with an insufficient coverage of network und traffic parameters. Therefore, this work presents test cases for a systematic scheduler benchmarking, which offers an improved test coverage and thereby allows the identification of strengths and weaknesses of tested schedulers. The publication of these test cases as dataset ensures reusability for future scheduler comparisons, whereas a precise description of their creation addresses transparency and extensibility.

As another major contribution, this work presents schedulers based on mathematical models. Specifically, Integer Linear Programming is used to represent the planning problem of time-triggered network traffic and find appropriate solutions. With respect to these models, the primary design goal was a comprehensive coverage of the routing and scheduling design space, which allows giving answers regarding the feasibility of networking scenarios and the quality of solutions. Thereby, these models provide reference data for scheduler comparisons. Additionally, comprehensive optimizations were applied to this model-based approach, leading to a high performance (i.e. low runtimes) and ensuring practical applicability for scheduling time-triggered networks. By means of these optimizations, the average runtimes for the systematic test cases were reduced by more than 90 %. Furthermore, implemented model extensions enable the creation of schedules that are compatible with standards-compliant IEEE 802.1Q bridges.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASAP As Soon As Possible 104

ATS Asynchronous Traffic Shaper 42

AVB Audio Video Broadcast 23, 24

BE Best Effort 32

BOX Boxplot 60

BPI Performanceindikator 58

BTR Backtracking 199

CBS Credit-Based Shaper 34

CI Konfidenzintervall 60

CNC Centralized Network Configuration 39

CP Constraint Programming 105

CT Cut-Through 52

CUC Centralized User Configuration 40

E2E Ende-zu-Ende 23

FDB Filtering Database 47

FIFO First In First Out 163

FRER Frame Replication and Elemination 37

GCL Gate Control List 42

gPTP generalized Precision Time Protocol 38

ID Identifikation 45

IE Industrial Ethernet 23

IEEE Institute of Electrical and Electronics Engineers 23

IFG Inter-Frame Gap 52

IIS Irreducible Inconsistent Subsystem 185

ILP Integer Linear Programming 25

IP Internet Protocol 45

IPV Internal Priority Value 48

IRT Isochronous Real-Time 34

JRaS Joint Routing and Scheduling 110

kSP k Shortest Path 157

LAN Local Area Network 36

LB Load Balanced 125

LLDP Link Layer Discovery Protocol 33

LP Linear Programming 95

MAC Medium Access Control 36

MED Median 60

MFTT multi-frame Time-Triggered 200

MIP Mixed Integer Programming 101

MRP Multiple Registration Protocol 38

MTU Maximum Transmission Unit 44

NCC No-Cross-Cycle 214

NW No-Wait 202

OMT Optimization Modulo Theory 198

OPC UA OPC Unified Architecture 27

OT Operational Technology 23

PSFP Per-Stream Filtering and Policing 38

QI Qualitätsindikator 58

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Vernetzung von Anlagen in der Fabrikautomation stellt besondere Anforderungen an die Kommunikationsinfrastruktur. Neben einer hohen Zuverlässigkeit müssen insbesondere konkrete *Ende-zu-Ende* (E2E)-Latenzen bei der Kommunikation garantiert werden. Je nach Anwendungsbereich können Störungen dieser Kommunikation, wie beispielsweise Paketverluste oder -verzögerungen, zu fatalen Folgen führen: Neben Produktionsausfällen durch das Anhalten von Prozessen kann es zur Beschädigung von Anlagen oder gar Gefährdung von Personen kommen. Für diese Form der Machine-to-Machine-Kommunikation unter sogenannten *harten Echtzeitanforderungen* kommen daher seit den 1980er Jahren speziell zu diesem Zweck entwickelte Feldbusse zum Einsatz. Diese von IT-Netzwerken separierten *Operational Technology* (OT)-Netzwerke ermöglichen die Kommunikation zwischen OT- und IT-Systemen nicht oder nur über spezielle Gateways. Die vertikale Integration von Produktionsprozessen wird somit erschwert, während der Aufbau separater Netzwerke für IT und OT die Kosten der Fabrikvernetzung steigert. Seit den 2000er Jahren werden Feldbusse daher zunehmend von *Industrial Ethernet* (IE)-Systemen abgelöst [1]. Diese spezialisierten Kommunikationssysteme basieren bezüglich Verkabelung, Steckertypen und physikalischer Übertragung auf standardkonformem Ethernet nach IEEE 802.3 und sollen die besonderen Anforderungen der OT-Vernetzung erfüllen, während gleichzeitig IT-Kommunikation im selben Netzwerk stattfinden kann. Somit werden konvergente IT/OT-Infrastrukturen ermöglicht. Problematisch ist jedoch, dass auf Basis dieser grundlegenden Idee zahlreiche, untereinander nicht kompatible Umsetzungen (Profinet, EtherCAT, TTEthernet, usw.) entworfen wurden, welche für die Kommunikation unter harten Echtzeitanforderungen spezielle Hardware des jeweiligen Systems erfordern [E1]. Dies führt neben Herstellerabhängigkeiten zu einer eingeschränkten Flexibilität, da ein Wechsel der Infrastruktur mit einem kostenintensiven Hardwareaustausch verbunden ist.

Nachdem die Verbreitung von IE-Systemen den Bedarf nach einer konvergenten IT/OT-Infrastruktur auf Basis von Ethernet gezeigt hat, wurde von der IEEE ein Standardisierungsprozess ins Leben gerufen, um die Funktionen von Ethernet nach IEEE 802.3 und IEEE 802.1 entsprechend der besonderen *Quality of Service* (QoS)-Anforderungen von OT-Netzwerken zu erweitern. Durch die Weiterentwicklung der Standards sollen Herstellerabhängigkeiten reduziert und konvergente IT/OT-Infrastrukturen zu gesenkten Kosten verfügbar werden. Die in Bezug auf die Echtzeitkommunikation entwickelten Erweiterungen von Ethernet werden auch unter den Begriffen *Audio Video Broadcast*

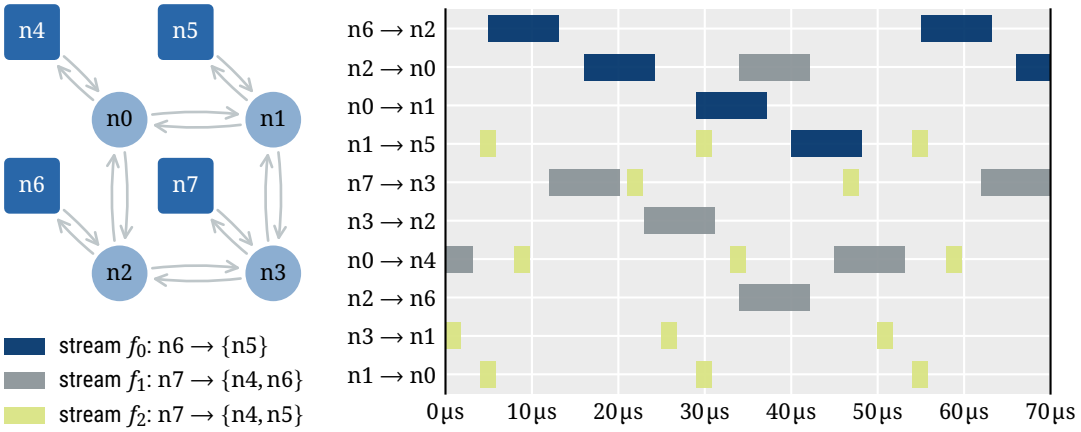


Abbildung 1.1: Planung zeitgesteuerter Netzwerke: Gegeben sind die Netzwerktopologie und eine Liste erforderlicher Echtzeitstreams (links). Ein Planungsalgorithmus muss den Sendezeitplan entwerfen (rechts). Hier: Beispiel mit Ringtopologie (4 Switches, 4 Endgeräte) sowie 3 periodischen Echtzeitstreams (Unicast und Multicast) mit einem geforderten Sendeintervall von 50 µs für die Streams f_0 und f_1 sowie 25 µs für f_2.

(AVB) und *Time-Sensitive Networking* (TSN) zusammengefasst. Konkret beschreiben diese Erweiterungen des Ethernet-Standards Mechanismen, die eine strikte Priorisierung kritischer Datenstreams sowie garantierte Kommunikationslatenzen ermöglichen. Wie schon aus verschiedenen IE-Systemen (insbesondere Profinet und TTEthernet) bekannt, können kritische Daten dabei je nach QoS-Anforderung entweder als priorisierter, ratenlimitierter Stream oder in dedizierten Zeitschlitzen übertragen werden, wobei letztere Übertragungsform geringste Latenzen und die Vermeidung von Jitter ermöglicht. Um die Einhaltung vorgegebener E2E-Latenzen gegenüber Anwendungen *garantieren* zu können, ist jedoch bei beiden Übertragungsformen eine Planung der kritischen Datenstreams erforderlich. Hierbei müssen die Streameigenschaften (z.B. Datenrate, Sendeintervall) und Latenzanforderungen aller kritischen Datenstreams noch vor Beginn der Kommunikation durch eine Managementinstanz erfasst werden, sodass diese eine passende Netzwerkkonfiguration ermitteln und in den Geräten hinterlegen kann. Die TSN-Standards beinhalten zu diesem Zweck sowohl konkrete Vorschläge für die Managementarchitektur als auch für die Protokolle zum Erfassen von Streamanforderungen und die Konfiguration der Netzwerkgeräte. Nicht Teil des Standardisierungsprozesses sind hingegen Planungsalgorithmen, die in der Managementinstanz ausgeführt werden, um anhand der gegebenen Anforderungen eine adäquate Netzwerkkonfiguration zu bestimmen.

Bezüglich dieser Planungsalgorithmen überschneidet sich die Problemstellung zwischen TSN und herkömmlichen IE-Systemen mit vergleichbaren QoS-Mechanismen. Dies betrifft insbesondere die Planung des *zeitgesteuerten Verkehrs*, also der Datenübertragung innerhalb von vorab reservierten Zeitschlitzen. Um bestimmte E2E-Latenzen zu garantieren, werden den Streams dieser Verkehrsklasse bei der Planung sowohl feste Routen durch das Netzwerk als auch Sendezeitschlitze auf jedem Netzwerklink der gewählten Route zugewiesen. Die Planung der Sendezeitschlitze wird auch als Scheduling bezeichnet. Das Planungsproblem einschließlich einer sinnvollen Lösung ist in Abbildung 1.1 anhand eines einfachen Beispiels veranschaulicht. Obwohl IE-Systeme mit vergleichbaren Mechanismen bereits seit vielen Jahren eingesetzt werden, sind die genutzten Planungsverfahren

nicht umfassend dokumentiert und in der Literatur finden sich aus der Zeit vor der Entwicklung von TSN nur wenige auf dieses Problem spezialisierte Arbeiten (z.B. [2, 3]). Darin werden wesentliche Aspekte des vorliegenden Planungsproblems noch nicht hinreichend adressiert. Nicht ausreichend bearbeitet sind beispielsweise (a) die gemeinsame Betrachtung des Lösungsraumes bezüglich Routing und Scheduling, (b) die Skalierbarkeit der Lösung für die Planung großer Netzwerke, und (c) der Vergleich verschiedener Lösungsansätze. Seitdem die Mechanismen für das zeitgesteuerte Senden jedoch als Teil von TSN standardisiert wurden und der zugehörige Standard (IEEE 802.1Qbv) verabschiedet wurde, ohne Vorgaben bezüglich der Planungsalgorithmen zu machen, werden die entsprechenden Fragen in einer Vielzahl von Forschungsarbeiten diskutiert. Ein Überblick dieser Arbeiten findet sich in [4]. Gleichermaßen befasst sich auch diese Arbeit mit den genannten Fragestellungen, wobei die Kernthemen der Arbeit die *ganzheitliche Modellierung des Routings und Schedulings als Integer Linear Program* (ILP)[1] sowie der *objektive und reproduzierbare Vergleich von Lösungsansätzen* sind. Im Folgenden werden die veröffentlichten wissenschaftlichen Beiträge zusammengefasst und eingeordnet, welche die Basis für diese Monographie bilden, bevor die Ziele und Struktur der Monographie konkretisiert werden.

1.1 Übersicht und Einordnung der eigenen Arbeiten

An dieser Stelle sollen Beiträge mit direkter Relevanz für die hier vorliegende Arbeit zusammengefasst werden, während auf mitverfasste Arbeiten [E1, E3, E4, E5, E6, E7, E8, E9, E10, E2] angrenzender Themen nicht näher eingegangen wird. Die folgenden Beiträge wurden auf internationalen ACM/IEEE-Konferenzen vorgestellt und werden hier chronologisch nach Veröffentlichungszeitpunkt präsentiert.

Application-Aware Industrial Ethernet Based on an SDN-Supported TDMA Approach [E11]

In dieser Arbeit wird eine zentral verwaltete Echtzeitkommunikation auf Basis von Ethernet und *Software-Defined Networking* (SDN) [5] vorgestellt und prototypisch implementiert. Das Konzept sieht vor, dass der zentrale SDN-Controller die Netzwerktopologie über Discovery-Mechanismen erkennt und Anforderungen der kommunizierenden Echtzeitanwendungen über eine sogenannte Northbound API mitgeteilt bekommt. Anhand dieser Informationen berechnet der Controller Routen und Sendezeitschlitze, vergleichbar mit dem Beispiel aus Abbildung 1.1. Die ermittelten Routen werden in den Netzwerkswitches über das populäre OpenFlow-Protokoll [6] konfiguriert, während der Sendezeitplan den Anwendungen über ein eigens entwickeltes Protokoll mitgeteilt wird. Die grundlegende Idee dieser Arbeit war es, durch die Nutzung des SDN-Konzepts und des teilweise in gewöhnlichen Data-Center-Switches verfügbaren OpenFlow-Protokolls eine Alternative zu existierenden IE-Systemen zu entwerfen, welche die bekannten Hersteller- und Hardwareabhängigkeiten dieser Systeme vermeidet. Zugleich sollte durch die Nutzung von SDN die Flexibilität bei der Konfiguration gesteigert werden. Damit stimmten die wesentlichen Ziele mit denen der TSN-Standardisierung überein, wobei die Entwicklung in [E11] unabhängig von TSN erfolgte, da die meisten relevanten Funktionen von TSN zu diesem Zeitpunkt noch nicht als Standard verabschiedet und dementsprechend unbekannt waren. Die nahezu zeitgleiche

[1] ILP wird in dieser Arbeit zur Abkürzung von Integer Linear Programming und Integer Linear Program genutzt.

Standardisierung von TSN hat allerdings zum einen die Bedeutung des in [E11] vorgeschlagenen Konzepts untermauert und zum anderen eine im Vergleich zu OpenFlow-Switches geeignetere Hardwarebasis für das angestrebte Echtzeitnetzwerk in Aussicht gestellt. Darüber hinaus wurde im gleichen Zeitraum bekannt, dass die Algorithmen zur Verkehrsplanung nicht Teil des Standardisierungsprozesses sind und daher als offene Forschungsfrage verbleiben.

ILP-Based Joint Routing and Scheduling for Time-Triggered Networks [E12]

Die anschließende Arbeit [E12] fokussiert sich auf die Planung des zeitgesteuerten Datenverkehrs, da diese nicht durch die TSN-Standards definiert wird. Obwohl sich die Arbeit am erwarteten Funktionsumfang von TSN-Switches orientiert, wird hierbei ein allgemeineres Routing- und Schedulingproblem formuliert und als ILP modelliert und gelöst. Die Verallgemeinerung wird vorgenommen, um gegebenenfalls auch in anderen zeitgesteuerten Netzwerken wie beispielsweise bekannten IE-Systemen anwendbar zu sein. Gegenüber älteren Arbeiten [2, 3, 7], die das Planungsproblem im Kontext der herkömmlichen Bus- und IE-Systeme bearbeiten, unterscheidet sich das vorgeschlagene Lösungsverfahren durch die gemeinsame Betrachtung von Routing und Scheduling. Diesbezüglich zeigt [E12], dass diese gemeinsame Betrachtung gegenüber Schedulingverfahren auf Basis eines fest vorgegebenen Routings Vorteile bei der Lösbarkeit von Probleminstanzen sowie in der Güte der gefundenen Lösungen aufweisen kann. Einige der etwa zeitgleich entstandenen Arbeiten berücksichtigen zwar ebenfalls den Einfluss des Routings, unterscheiden sich jedoch durch genutzte Methoden (z.B. Tabu-Search in [8]) oder Vereinfachungen (z.B. kein linkspezifisches Scheduling in [9], kein Switch-Buffering in [10]).

Automatic Configuration of a TSN Network for SDC-Based Medical Device Networks [E13]

Um die praktische Anwendbarkeit der Ansätze aus [E11, E12] zu demonstrieren, wird in [E13] ein Konzept sowie eine prototypische Implementierung zur echtzeitfähigen Vernetzung von Medizingeräten vorgestellt. Die kommunizierenden Medizingeräte unterstützen in diesem Fall die Kommunikationsprotokolle und eine Beschreibung abrufbarer Daten nach IEEE 11073: *Service-Oriented Device Connectivity* (SDC), einer Familie von Standards zur Gerätekommunikation und Datenmodellierung in der Medizintechnik. Die echtzeitfähige Vernetzung soll wiederum durch ein TSN sichergestellt werden. In [E13] wird durch die Beschreibung einer passenden Managementinstanz (vergleichbar mit einem SDN-Controller) das bisher fehlende Bindeglied zwischen SDC und TSN geschaffen. Dabei nutzt die Managementinstanz die Protokolle und Datenmodelle von SDC, um die Kommunikationsanforderungen der Medizingeräte zu erfassen und eine entsprechende Liste kritischer Datenstreams zu erstellen. Diese wird anschließend dem in [E12] entworfenen Planungsalgorithmus übergeben, um eine passende Konfiguration für ein zeitgesteuertes Netzwerk zu erstellen. Auch wenn das Konzept aus [E13] in dieser Arbeit nicht näher diskutiert wird, handelt es sich bei diesem mitverfassten Beitrag um einen wichtigen Nachweis der praktischen Anwendbarkeit der in dieser Arbeit präsentierten Planungsalgorithmen.

ILP-Based Routing and Scheduling of Multicast Realtime Traffic in Time-Sensitive Networks [E14]

Im Zuge von Industrie 4.0 [11] und dem Industrial Internet of Things wird eine Zunahme der vernetzten Geräte in industriellen Anwendungen erwartet. Dabei verschiebt sich auch die Art

der Kommunikation von einfachen Unicastverbindungen zu einer umfassenderen Verteilung von Informationen, beispielsweise durch Publish/Subscribe-Systeme. In Kommunikationsstandards für industrielle Anwendungen, wie der *OPC Unified Architecture* (OPC UA) werden daher zugehörige Kommunikationsprofile definiert [56]. Aufgrund dieser Entwicklungen müssen die Planungsalgorithmen zukünftiger Echtzeitnetzwerke skalierbar sein und Multicaststreams unterstützen. In [E14] werden daher wesentliche Teile der ILP-Modellierung aus [E12] überarbeitet, um die Planung von Multicaststreams zu ermöglichen und Rechenzeiten zu verbessern. Zur Performancesteigerung wird dabei insbesondere die Repräsentation des Problems als ILP optimiert. Damit unterscheidet sich der Ansatz grundlegend von vielen Planungsverfahren die im gleichen Zeitraum entwickelt wurden und entweder Heuristiken einsetzen [12] oder den Suchraum einschränken, beispielsweise durch den Ausschluss von Switch-Buffering [13]. Des Weiteren zeigt [E14], inwiefern das Optimierungsziel eines ILP genutzt werden kann, um einen sinnvollen Tradeoff zwischen Lösungsgüte und Rechenzeiten zu erzielen.

TSN Scheduler Benchmarking [E15]

Die vorherigen Arbeiten an Schedulern für zeitgesteuerte Netzwerke haben wiederholt die Frage aufgeworfen, wie die eigenen ILP-Modelle im Vergleich zu einer Vielzahl anderer Vorschläge aus der Literatur zu bewerten sind. Bei der umfassenden Analyse veröffentlichter Arbeiten zeigt sich, dass es in der Literatur keinerlei Konsens über die zu verwendenden Testszenarien zur Leistungsbeurteilung von Schedulern gibt. Darüber hinaus unterscheiden sich die Scheduler in Funktionsumfang und zugrundeliegenden Problemvereinfachungen, was die Vergleichbarkeit weiter erschwert. Bezüglich des ersten Problems werden in [E15] Testszenarien vorgeschlagen und als Datensatz veröffentlicht [E16, E17], welche durch ihren systematischen Aufbau eine Leistungsbeurteilung von Schedulern mit angemessenem Testumfang ermöglichen. Zur weiteren Einordnung von Schedulern werden außerdem Kriterien zur Klassifizierung vorgeschlagen, die deren Funktionsumfang und mathematische Methoden umfassen. Derartige Unterscheidungsmerkmale von Schedulern werden auch in der etwa zeitgleich entstandenen Arbeit [4] präsentiert. Eine Methodik zur Leistungsbeurteilung von Schedulern wird darin jedoch nicht diskutiert.

Die Relevanz der eigenen Arbeiten wird unter anderem durch eine Vielzahl von Zitationen (insbesondere von [E12, E14]) untermauert. Bezüglich [E12] ist dies insbesondere mit der Bearbeitung einer durch den TSN-Standardisierungsprozess hochrelevanten Fragestellung und den im Vergleich zu anderen Arbeiten frühen Veröffentlichungszeitpunkt zurückzuführen. Das vorgeschlagene ILP-Modell ist dabei im Kontext des hier eingeführten Schedulingproblems das erste mit gemeinsamer Modellierung von Routing und linkspezifischem Scheduling einschließlich Switch-Buffering. Des Weiteren eignet sich der ILP-basierte Lösungsansatz durch die präzise, formale Beschreibungsform und die gute Verfügbarkeit von ILP-Solvern bestens zur Nachimplementierung. Die spätere Arbeit [E14] zeichnet sich darüber hinaus durch die Kombination von Multicastrouting und Switch-Buffering aus, welche sich in modellbasierten Lösungsansätzen bis dahin nur in der im gleichem Jahr veröffentlichten Arbeit [14] findet. Von dieser unterscheidet sich die eigene Arbeit wiederum durch die Performanceverbesserungen mittels Optimierung des ILP-Modells. Eine ausführlichere Abgrenzung zu den Arbeiten aus der Literatur wird in Kapitel 10.1.2 vorgenommen. Der Entwicklungsverlauf der

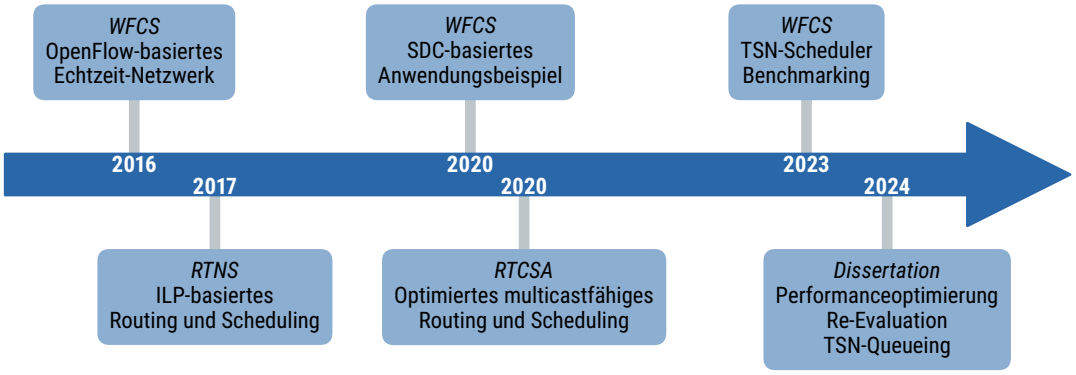


Abbildung 1.2: Entwicklungsverlauf der eigenen Arbeiten.

eigenen Arbeiten ist in Abbildung 1.2 zusammengefasst, wobei auch die im Rahmen dieser Arbeit vorgenommenen Erweiterungen gezeigt sind. Diese werden im Folgenden erläutert.

1.2 Zielsetzung der Arbeit

Diese Monographie erweitert und überarbeitet die Inhalte aus [E15] zu einem verbesserten Benchmarking und einer detaillierteren Schedulerklassifizierung, die Inhalte aus [E12, E14] zu einer optimierten ILP-basierten Lösung des Routing- und Schedulingproblems, und ergänzt das ILP-Modell um unveröffentlichte Komponenten zur Sicherstellung der TSN-Konformität. Diese drei primären Ziele werden im Folgenden genauer beschrieben. Dabei wird das zuvor beschriebene und in Abbildung 1.1 veranschaulichte Planungsproblem[2] bezüglich Routing und Scheduling von zeitgesteuerten Datenstreams auch als *Time-Triggered Routing and Scheduling Problem* (TT-RSP) bezeichnet. Planungsverfahren, die TT-RSP lösen, werden außerdem als *Scheduler* bezeichnet.

Etablierung eines Benchmarkings und einer Klassifizierung für Scheduler

Einer der wichtigsten Aspekte bei der Entwicklung von Schedulern ist die objektive Bewertung von Leistungsfähigkeit und Funktionsumfang. In der Literatur sind jedoch bisher keine objektiven und wiederverwendbaren Bewertungsmaßstäbe etabliert. Die Bewertung der Leistungsfähigkeit veröffentlichter Scheduler erfolgt in der Regel durch Messung der Rechenzeit ausgewählter Testszenarien. Diese Testszenarien weisen jedoch in den bisher in der Literatur gezeigten Evaluationen einen eingeschränkten Umfang auf und unterscheiden sich zwischen Publikationen. Die unterschiedlichen Testszenarien führen zu einer mangelhaften Vergleichbarkeit, während der unzureichende Testumfang offen lässt, inwiefern gezeigte Ergebnisse auf andere Netzwerke und Streameigenschaften übertragbar sind. Des Weiteren unterscheiden sich veröffentlichte Scheduler in getroffenen Annahmen und unterstützten Funktionen (z.B. Buffering von Paketen in Switches, heterogene Sendeintervalle der Datenstreams). In dieser Arbeit werden daher umfassende Testszenarien für das Benchmarking von Schedulern entworfen und eine Klassifizierung für den Funktionsumfang und die angewandten Methoden von Schedulern eingeführt.

[2] Eine formale Beschreibung des Problems folgt in Kapitel 3.

Erstellung einer Referenzlösung für TT-RSP auf Basis von Integer Linear Programming

Neben einer objektiven Evaluationsmethodik ist es bei der Entwicklung neuer Scheduler essentiell, eine direkte Vergleichsmöglichkeit für die Bewertung der Leistungsfähigkeit eines neu entwickelten Schedulers zu besitzen. Hierfür kann beispielsweise eine existierende, performante Lösung als Referenz herangezogen werden. Bisher in der Literatur gezeigte Lösungsverfahren sind diesbezüglich nur bedingt geeignet, da mindestens eine der folgenden Schwächen zutrifft: (a) Unterschiedliche Vereinfachungen bei der Problemmodellierung (z.B. homogene Sendeintervalle, kein Buffering in Switches) erschweren die Vergleichbarkeit von Lösungsverfahren. (b) Implementierungen werden nicht veröffentlicht und der Einsatz von Algorithmen, die nicht formal und präzise beschrieben werden, erschwert die Nachimplementierung. (c) Aufgrund unzureichender Optimierung wird das volle Potential der eingesetzten Methoden nicht ausgeschöpft, sodass die resultierende Performance nicht repräsentativ für die eingesetzten Methoden ist. In dieser Arbeit wird daher eine Lösung für TT-RSP vorgeschlagen, die sich durch die formale Beschreibung (ILP) für eine präzise Nachimplementierung eignet. Dabei wird eine ganzheitliche Problemmodellierung angestrebt, bei der auf Vereinfachungen soweit wie möglich verzichtet wird. Durch eine Vielzahl von Optimierungen des Modells soll darüber hinaus ein Modell geschaffen werden, das die Leistungsfähigkeit eines ILP-basierten Lösungsverfahrens angemessen repräsentiert. Obwohl die Anzahl ähnlicher Arbeiten in den letzten Jahren stark zugenommen hat (siehe Kapitel 10.1.2 und [4]), existiert nach bestem Wissen kein anderes modellbasiertes Lösungsverfahren, das einen vergleichbaren Funktionsumfang besitzt und eine ebenso umfassende Optimierung durchlaufen hat, wie die in dieser Arbeit vorgestellten Modelle.

TSN-spezifische Anpassung des ILP-basierten Schedulers

Das Lösungsverfahren für die allgemein gehaltene Problemstellung von TT-RSP wird abschließend um Randbedingungen des Queuing-Modells von TSN erweitert, um zu garantieren, dass gefundene Lösungen TSN-kompatibel sind. Zusammen mit den umfassenden Performanceverbesserungen führt diese Erweiterung zu einem Lösungsverfahren für zeitgesteuerte Netzwerke, das für den Einsatz in der Managementinstanz eines realen TSN-Netzwerkes geeignet ist.

Auch wenn diese Arbeit auf [E12, E14, E15] basiert und viele der Modelle und Analysen aufgreift, sei an dieser Stelle angemerkt, dass im Zuge dieser Arbeit sämtliche Beschreibungen überarbeitet und alle Evaluationen erneut durchgeführt wurden. Die wichtigsten Überarbeitungen werden hier zusammengefasst und begründet:

- Benchmarkingszenarien wurden an die Leistungsfähigkeit und den umfangreicheren Funktionsumfang der getesteten Scheduler angepasst.
- Die erneute Evaluation der zuvor veröffentlichten Lösungsverfahren von TT-RSP mit den erst später definierten Benchmarkingszenarien bietet wesentlich detaillierte Einblicke in die Leistungsfähigkeit der Verfahren als die ursprünglichen Veröffentlichungen.
- Die erneute Evaluation hat es außerdem ermöglicht, alle Verfahren zwecks Vergleichbarkeit auf derselben Hardware- und Softwareplattform zu testen.

- Die Einführung einer einheitlichen Symbolik über alle mathematischen Formulierungen hinweg dient der besseren Nachvollziehbarkeit.
- Die Aufrechterhaltung der Kompatibilität der früher veröffentlichten Modelle mit später entwickelten Verbesserungen ermöglicht weitreichende Vergleiche zwischen verschiedenen Entwicklungsstufen der entworfenen Lösungsverfahren.

Des Weiteren handelt es sich bei der Optimierung und TSN-spezifischen Anpassung der ILP-Modelle um umfassende Erweiterungen der veröffentlichten Modelle. Nach bestem Wissen des Autors ist der in dieser Arbeit gezeigte Performancevergleich verschiedener Modellierungsmöglichkeiten von TT-RSP als ILP außerdem der bisher umfangreichste dieser Art. Dabei wurden sowohl Testszenarien [E18] als auch Ergebnisse [E19] als Datensatz veröffentlicht. Das entwickelte Verifikationsmodell für gefundene Lösungen von Probleminstanzen von TT-RSP ist in der Literatur zu zeitgesteuerten Netzwerken bisher ebenfalls nicht in vergleichbarer Weise gezeigt worden.

1.3 Struktur der Arbeit

In Kapitel 2 wird die notwendige Netzwerkarchitektur für eine zuverlässige Kommunikation unter harten Echtzeitanforderungen veranschaulicht und ein Überblick über TSN gegeben, welches in dieser Arbeit als Referenz für die in Praxis zu erwartenden Fähigkeiten von Netzwerkgeräten dient. Die Planung des zeitgesteuerten Netzwerkverkehrs (TT-RSP) wird in Kapitel 3 formalisiert. Der erste Schwerpunkt der Arbeit, das systematische Benchmarking von Lösungsverfahren für TT-RSP, wird in Kapitel 4 präsentiert. Das Benchmarking ist der Erläuterung von Lösungsverfahren für TT-RSP vorangestellt, da es während der Beschreibung von Lösungsverfahren in den folgenden Kapiteln kontinuierlich zum Einsatz kommt. Kapitel 5 gibt eine kurze Einführung in Integer Linear Programming, welches die Basis für die entworfenen Lösungsverfahren liefert. In Kapitel 6 wird zunächst die Wahl von Integer Linear Programming als bevorzugte Methode zum Lösen von TT-RSP begründet, bevor entsprechende Unicast- und Multicastmodelle erarbeitet werden. Kapitel 7 zeigt weitere Modellvarianten auf, die der Performancesteigerung dienen, und erweitert die Modelle um Randbedingungen bezüglichen des Queuing-Modells von TSN. Ein Gesamtüberblick über die Entwicklungsstufen des ILP-Modells folgt in Kapitel 8 zusammen mit einer Evaluation von Multicastszenarien, welche in den vorherigen Kapiteln nicht berücksichtigt wurden. Kapitel 9 befasst sich mit numerischen Problemen, die im Rahmen der ILP-Modellierung betrachtet werden müssen und stellt die genutzten Verifikationsalgorithmen vor, welche die Korrektheit aller im Rahmen dieser Arbeit analysierten Lösungen sicherstellen. Kapitel 10 führt eine Schedulerklassifizierung ein und bietet auf dieser Basis einen umfassenden Vergleich der eigenen Lösungsverfahren mit Alternativen aus der Literatur. Dieser Vergleich wird erst am Ende der Arbeit vorgenommen, da hierfür ein detailliertes Verständnis der ILP-basierten Lösungsverfahren von Vorteil ist. Eine Zusammenfassung der wesentlichen Erkenntnisse der Arbeit und ein Überblick über offene Forschungsfragen wird abschließend in Kapitel 11 gegeben.

Kapitel 2

Echtzeitnetzwerke

Die Kernthemen dieser Arbeit liegen beim Entwurf und der Evaluation von Planungsalgorithmen für zeitgesteuerte Netzwerke. Für eine korrekte Spezifikation des betrachteten Planungsproblems ist jedoch die gesamte Netzwerkarchitektur, in der die entworfenen Algorithmen zum Einsatz kommen sollen, zu berücksichtigen. Nur anhand dieser Netzwerkarchitektur können beispielsweise die für die Algorithmen zu erwartenden Eingangsdaten (z.B. Netzwerktopologie, Eigenschaften von Datenstreams) und die zu erzeugenden Ergebnisse (z.B. Konfigurationsdaten für Netzwerkgeräte) abgeleitet werden. In diesem Kapitel soll daher ein Überblick über die Netzwerkarchitektur und den zugehörigen Konfigurationsprozess gegeben werden, welcher zur Erfüllung harter Echtzeitanforderungen bei der Kommunikation notwendig ist. Hierzu wird zunächst das abstrakte Konzept eines Echtzeitnetzwerks vorgestellt, bevor TSN als eine konkrete Realisierung im Detail beleuchtet wird.

2.1 Konzeptionelle Darstellung eines Echtzeitnetzwerks

Der Begriff *Echtzeit* wird in der Netzwerkkommunikation je nach Kontext unterschiedlich streng ausgelegt, was zu maßgeblichen Unterschieden bei den Ansprüchen an die Kommunikationsinfrastruktur führt. In der Literatur wird daher üblicherweise zwischen folgenden Ausprägungen unterschieden:

Harte Echtzeitanforderungen liegen vor, wenn die gesendeten Nachrichten einer Anwendung innerhalb einer vorgegeben Zeitschranke (*Deadline*) beim Empfänger ankommen müssen, da ein Überschreiten der Deadline, ebenso wie der Verlust einer Nachricht, fatal sein kann. Mögliche Folgen im Fehlerfall sind beispielsweise die Beschädigung von Produktionsanlagen oder die Gefährdung von Personen.

Weiche Echtzeitanforderungen liegen vor, wenn der Verlust von Nachrichten oder das Verpassen von vorgegebenen Zeitschranken zu einer Qualitätsminderung auf Anwendungsebene führen kann. Dieser Fall liegt beispielsweise in der Audio- und Videokommunikation vor. Permanente Schäden sind in diesem Fall jedoch nicht zu erwarten.

Die Entwurfsstrategie zwischen Netzwerken für die jeweiligen Anwendungsbereiche unterscheidet sich drastisch, da es für weiche Echtzeitanforderungen ausreichend ist, die *durchschnittliche* Kom-

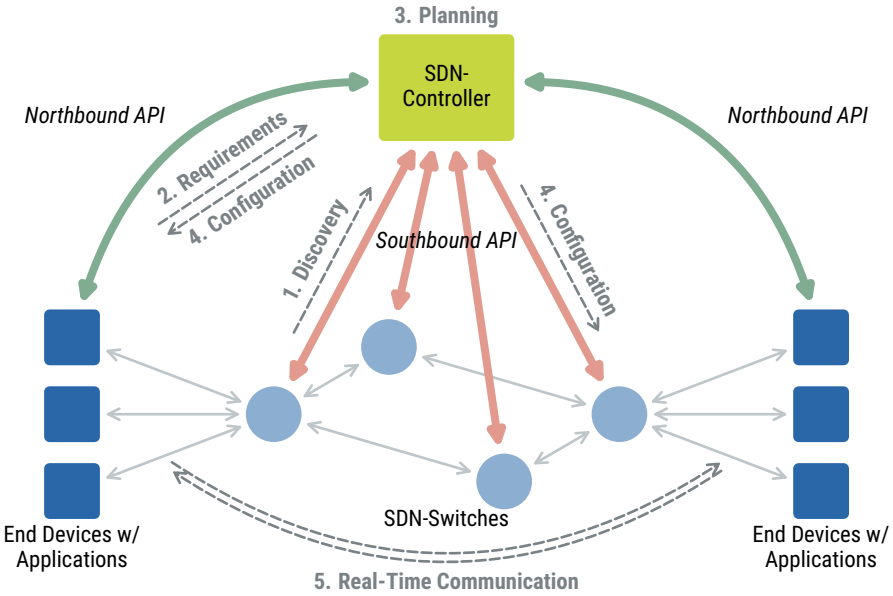


Abbildung 2.1: Konfigurationsprozess eines Echtzeitnetzwerks.

munikationsqualität zu optimieren, während für harte Echtzeitanforderungen stets eine Analyse des *Worst-Case Delay* (WCD) erforderlich ist. Die WCD-Analyse dient dem Nachweis, dass die Nachrichtenübertragung unter allen Umständen innerhalb der vorgegebenen Zeitschranke erfolgen kann. Dabei werden in der Regel nur reguläre Betriebszustände berücksichtigt, wohingegen der Ausfall von Systemkomponenten gesondert betrachtet wird. In dieser Arbeit bezeichnet der Begriff „Echtzeitfähigkeit“, sofern nicht genauer eingegrenzt, stets die Erfüllung harter Echtzeitanforderungen.

Bei herkömmlichen Ethernetnetzwerken ohne spezielle Erweiterungen im Sinne eines IE bzw. TSN handelt es sich um sogenannte *Best Effort* (BE)-Netzwerke. Innerhalb dieser werden Nachrichten von Anwendungen versendet, ohne dass zuvor überprüft wird, ob das Netzwerk ausreichend Ressourcen zur Verfügung stellen kann, um die Übertragung erfolgreich abzuschließen. In modernen Ethernetnetzwerken, die durch Switches segmentiert werden, ergibt sich die Ressourcenlimitierung aus den Bandbreiten der Netzwerklinks und Buffergrößen innerhalb der Switches. Da alle mit dem Netzwerk verbundenen Anwendungen zu beliebigen Zeitpunkten und ohne zentrale Koordination Nachrichten versenden, ist die Auslastung einzelner Netzwerklinks und die damit einhergehende Verzögerung von Nachrichten durch Buffering in Switches nicht vorhersagbar. Nach diesem Kommunikationsprinzip kann somit keine harte Echtzeitfähigkeit erzielt werden.

Der vorherrschende Lösungsansatz für dieses Problem ist die Einführung eines zentralen Controllers zur Koordination des zeitkritischen Netzwerkverkehrs. Dieses auf dem Software-Defined Networking (SDN) aufbauende Konzept wird beispielsweise in [E11] konkret beschrieben. Darin ist ein dedizierter Konfigurationsprozess für zeitkritische Datenstreams vorgesehen, bei dem Anwendungsanforderungen identifiziert und angemessene Ressourcenreservierungen im Netzwerk vorgenommen werden. Dieser ist in Abbildung 2.1 dargestellt und umfasst die folgenden Schritte:

1. *Topologieerkennung*: Der zentrale Controller erkennt die vollständige Netzwerktopologie. Zwischen den Netzwerkelementen ist die Erkennung von Verbindungen beispielsweise unter

Nutzung des *Link Layer Discovery Protocol* (LLDP) möglich. Erkannte Verbindungen sowie weitere Systemzustände (z.B. Bandbreite von Netzwerklinks) werden dem Controller über eine entsprechende Konfigurationsschnittstelle zwischen Controller und Netzwerkelementen mitgeteilt, der sogenannten *Southbound API*.

2. *Erfassung von Anwendungsanforderungen*: Die benötigten Echtzeitstreams der Anwendungen müssen dem Controller mitgeteilt werden. Hierfür wird eine Schnittstelle zwischen Controller und Anwendungen genutzt, welche auch als *Northbound API* bezeichnet wird. In diesem Schritt werden für jeden Echtzeitstream konkrete QoS-Anforderungen definiert.
3. *Planungsverfahren bezüglich Echtzeitstreams*: Anhand der nun bekannten Netzwerktopologie und angeforderten Echtzeitstreams ermittelt der Controller die notwendigen Ressourcenreservierungen und Gerätekonfigurationen zur Einhaltung aller QoS-Anforderungen.
4. *Konfiguration von Endgeräten und Netzwerkelementen*: Die Ressourcenreservierungen werden in den Netzwerkelementen konfiguriert (via Southbound API) und Anwendungen erhalten ebenfalls ggf. notwendige Konfigurationsparameter (via Northbound API).
5. *Echtzeitkommunikation*: Nach erfolgreichem Abschluss des Konfigurationsprozesses können Anwendungen die Kommunikation in den zuvor angeforderten Echtzeitstreams beginnen. Vorgenommene Ressourcenreservierungen garantieren die Einhaltung der in 2. formulierten QoS-Anforderungen.

Die konkrete Ausgestaltung von Northbound API und Southbound API ist implementierungsabhängig. Während die Northbound API stark vom Anwendungsbereich abhängt, gibt es für die Southbound API populäre Vertreter wie das *Simple Network Management Protocol* (SNMP) und OpenFlow [6]. In Bezug auf die in dieser Arbeit diskutierten Planungsverfahren für Echtzeitstreams sind dabei insbesondere folgende Eigenschaften des Konzepts von Bedeutung:

- Der Controller hat zum Zeitpunkt der Planung vollständiges Wissen über die Netzwerktopologie und geforderte Echtzeitstreams einschließlich aller relevanten Parameter.
- Die Planung findet vor dem Beginn der Echtzeitkommunikation als sogenannte *Offline-Planung* statt. Die Ausführungszeit des Planungsverfahrens muss somit grundsätzlich keinen strengen zeitlichen Anforderungen genügen und nicht vorhersagbar sein.

Das beschriebene Konzept deckt zunächst ausschließlich statische Szenarien ab, in denen während der Echtzeitkommunikation keine Veränderungen an Netzwerktopologie und erforderlichen Echtzeitstreams auftreten. Eine Erweiterung zu dynamisch veränderlichen Netzwerken ist jedoch möglich, indem beispielsweise während des Betriebs Echtzeitstreams aus der aktuellen Konfiguration entfernt oder zu dieser hinzugefügt werden. Dabei muss sichergestellt werden, dass bestehende Echtzeitstreams durch entsprechende Rekonfigurationsvorgänge nicht beeinflusst werden. Die grundlegende Idee der Offline-Planung bleibt auch bei der Rekonfiguration erhalten. Dies bedeutet, dass auch beim Hinzufügen von Echtzeitstreams eine Anmeldung am Controller, Planung und Ressourcenreservierung erfolgt, bevor die Kommunikation beginnt.

Bei der Planung und Ressourcenreservierung für Echtzeitstreams kommen in Ethernetnetzwerken zwei Grundprinzipien zum Einsatz:

Bandbreitenorientierte Ansätze charakterisieren Echtzeitstreams mittels *Burst* und maximaler *Senderate* (Bandbreite) und führen Reservierungen in Switches in Form ratenbasierter Konzepte durch. Bei der Netzwerkübertragung einer Nachricht über mehrere Hops erlauben diese Ansätze eine WCD-Analyse für jeden Hop beispielsweise mittels *Network Calculus* [15, 16, 17]. Somit lassen sich bei der Planung maximale E2E-Latenzen der Streams ermitteln und bezüglich der Einhaltung gegebener Latenzschranken verifizieren.

Zeitgesteuerte Ansätze befassen sich vorwiegend mit periodischer Kommunikation, bei der sich Echtzeitstreams mittels *Sendeintervall* und *Nachrichtengröße* charakterisieren lassen. Reservierungen in Switches erfolgen in Form von Sendezeitschlitzen. Die WCD-Analyse bezüglich E2E-Latenz eines Streams ist in diesem Fall trivial: Die Position einer Nachricht im Netzwerk ist durch den Sendezeitplan (Schedule) zu jedem Zeitpunkt bekannt, die E2E-Latenz ergibt sich somit aus dem Beginn des Zeitschlitzes auf dem ersten Netzwerklink und dem Ende des Zeitschlitzes auf dem letzten Netzwerklink der geplanten Route. Das Senden entsprechend einem vordefinierten Schedule erfordert jedoch eine Zeitsynchronisation aller an der zeitgesteuerten Kommunikation beteiligten Geräte.

Während beide Ansätze zur Erfüllung harter Echtzeitanforderungen geeignet sind, gibt es wesentliche Unterschiede bei den erzielbaren Übertragungseigenschaften und dem dafür notwendigen Aufwand. Bandbreitenorientierte Lösungen sind in der Regel einfacher realisierbar, da im Gegensatz zu zeitgesteuerten Ansätzen keine Zeitsynchronisation aller an der Kommunikation beteiligten Geräte erforderlich ist. Andererseits wird eine vollständig jitterfreie Kommunikation, d.h. eine gleichbleibende E2E-Latenz über lange Zeiträume, im Allgemeinen nur durch eine zeitgesteuerte Kommunikation ermöglicht. Darüber hinaus lassen sich nur bei der zeitgesteuerten Kommunikation die geringsten E2E-Latenzen garantieren, da innerhalb von Switches das Queuing von Frames unterschiedlicher Echtzeitstreams und die damit einhergehende Verzögerung vollständig vermieden werden kann.

Die praktische Realisierung beider Mechanismen findet sich sowohl in herkömmlichen IE-Systemen als auch den neuen TSN-Standards. Bandbreitenorientierte Ansätze finden beispielsweise durch *Rate-Constrained* (RC)-Traffic in TTEthernet [18, 8] und durch den *Credit-Based Shaper* (CBS) in TSN [57] besondere Unterstützung seitens der Switches. Zeitgesteuerte Ansätze gibt es als *Time-Triggered* (TT)-Traffic in TTEthernet, *Isochronous Real-Time* (IRT) in Profinet [19, 2] und in Form von *Scheduled Traffic* in TSN [58]. Auch ohne spezielle Hardwareunterstützung ist eine Echtzeitkommunikation grundsätzlich möglich. So nutzt [E11] beispielsweise einfache OpenFlow-Switches und realisiert die zeitgesteuerte Kommunikation ausschließlich durch Festlegung der Sendezeitpunkte von Frames direkt am Sender, während [16] einen bandbreitenorientierten Ansatz allein durch prioritätsbasierte Weiterleitung von Frames innerhalb von Switches beschreibt. In beiden Fällen, d.h. mit und ohne spezielle Hardwareunterstützung, muss dabei eine Planung, WCD-Analyse, und Gerätekonfiguration erfolgen, die vergleichbar mit dem hier bereits dargelegten Konfigurationsprozess ist. In der Praxis werden Systeme mit speziell auf Echtzeitfähigkeit ausgelegten Hardwarefunktionen bevorzugt, da sie beispielsweise die folgenden zusätzlichen Funktionen bieten:

- Zeitgesteuerte Frameweiterleitung innerhalb von Switches (andernfalls wie z.B. in [E11] nur zeitgesteuertes Senden direkt am Sender eines Streams)
- Verbesserte Isolation von Frames unterschiedlicher Verkehrsklassen
- Unterbrechung von Frameübertragungen des niedrig priorisierten Verkehrs zwecks Übertragung von höher priorisiertem Verkehr
- Verwerfen von Frames eines Echtzeitstreams, sofern diese nicht der ursprünglichen Spezifikation entsprechen, um negative Auswirkungen auf andere Echtzeitstreams zu verhindern

Im Allgemeinen bieten solche Systeme also detailliertere Möglichkeiten der Verkehrsformung sowie eine verbesserte Robustheit gegen Fehlerfälle, wie beispielsweise fehlerhaftes Sendeverhalten einzelner Anwendungen.

Diese Arbeit diskutiert die im dritten Konfigurationsschritt des hier vorgestellten Konzepts angesiedelten Planungsverfahren. Dabei liegt der Fokus bei der Planung von zeitgesteuertem Verkehr, wobei TSN als wichtigste Zieltechnologie aufgefasst wird. Im Folgenden wird daher erläutert, wie das beschriebene Gesamtkonzept aus Netzwerkarchitektur, Konfigurationsprozess und zeitgesteuerten Sendevorgängen innerhalb von TSN konkret umgesetzt wurde.

2.2 Time-Sensitive Networking (TSN)

Die im Verlauf dieser Arbeit diskutierten mathematischen Modelle sind in ihrer Anwendbarkeit nicht auf eine bestimmte Technologie beschränkt, sondern dienen grundsätzlich der effizienteren Planung von zeitgesteuertem Verkehr. Für die Entwicklung entsprechender Algorithmen ist es allerdings hilfreich, eine reale Netzwerktechnologie als Referenzsystem zu berücksichtigen. Anhand dieser Referenz können dann sinnvolle Annahmen über die zu erwartenden Eigenschaften und Funktionen realer Systeme getroffen werden. Außerdem können die für die Algorithmen zu erwartenden Eingangsdaten (z.B. Netzwerktopologie, Eigenschaften von Datenstreams) und die zu erzeugenden Ergebnisse (z.B. Konfigurationsdaten für Netzwerkgeräte) spezifiziert werden. Insgesamt soll so die praktische Anwendbarkeit der neu entwickelten Lösungsverfahren gewährleistet werden.

Beim Time-Sensitive Networking (TSN) handelt es sich um eine Reihe von Standards, die die etablierte Ethernet-Technologie (im Wesentlichen standardisiert durch IEEE 802.3 und IEEE 802.1) um Mechanismen erweitern, die zur Übertragung zeitkritischer Datenstreams notwendig sind. Aufgrund der Standardisierung durch die IEEE innerhalb der IEEE-802.1-Standardfamilie ist TSN herstellerunabhängig und die Standarddokumente sind mittlerweile[1] frei verfügbar. Daher ist TSN als Referenzsystem besonders geeignet und die entsprechenden Standarddokumente wurden im Rahmen dieser Arbeit genutzt, um Annahmen über die Funktionen und Eigenschaften realer Netzwerkhardware zu treffen. Dementsprechend werden die für diese Arbeit relevanten Mechanismen von TSN in diesem Kapitel vorgestellt.

[1] Dies war zu Beginn dieser Arbeit in 2016 nicht der Fall, da sich viele der TSN-Standards noch im *Draft*-Status befanden und nur fertiggestellte Standards (keine Drafts) nach Ablauf von sechs Monaten frei zugänglich gemacht werden [59].

2.2.1 Terminologie

Im Rahmen dieser Arbeit wird davon ausgegangen, dass der Leser mit der üblichen Terminologie aus der Netzwerktechnik vertraut ist. An dieser Stelle sollen jedoch einige Bezeichnungen aus den Standarddokumenten eingeführt werden, welche nicht dem allgemeinen Sprachgebrauch entsprechen oder weniger gebräuchlich sind.

Bridge
: Eine Bridge verbindet mehrere *Local Area Networks* (LANs) miteinander. Die Kernkomponente zur Frameweiterleitung zwischen mehreren Ports ist dabei das *Medium Access Control* (MAC)-Relay.

Talker
: Sender von Streams werden auch als Talker bezeichnet.

Listener
: Empfänger von Streams werden auch als Listener bezeichnet.

Stream
: Eine unidirektionale Kommunikation von einem Talker zu einem oder mehreren Listenern wird als Stream bezeichnet. Sofern konkrete Anforderungen an die maximale Kommunikationslatenz vorliegen, wird dieser als *time-sensitive* bezeichnet.

Traffic Shaper
: Mechanismen, die das Zeitverhalten bei der Frameweiterleitung zielgerichtet beeinflussen, werden als Traffic Shaper bezeichnet. Dies kann beispielweise durch Ratenlimitierungen oder durch zeitgesteuertes Senden erfolgen.

Bei einer Bridge handelt es sich um ein Netzwerkelement, das der Frameweiterleitung in einem lokalen Netzwerk dient und daher vorherrschend (aber nicht ausschließlich) Frame-Headerfelder verarbeitet, die den Protokollen der ISO/OSI-Schicht 2 zuzuordnen sind. Eine Bridge wird im allgemeinen Sprachgebrauch vorwiegend als (Layer-2-)Switch bezeichnet, obwohl dieser Begriff in den entsprechenden Standards nicht verwendet wird. Außerdem wird im Standard davon gesprochen, dass eine Bridge mehrere LANs miteinander verbindet. Dies ist ebenfalls abweichend vom allgemeinen Sprachgebrauch, in dem in der Regel alle durch Bridges (Switches) verbundenen Endgeräte als *ein* LAN aufgefasst werden. In den folgenden Kapiteln dieser Arbeit werden entsprechend dem Standard alle Verbindungselemente im Netzwerk als Bridge bezeichnet und bekannte unidirektionale Datenflüsse als Stream. Dabei werden vorwiegend Streams mit konkreten Latenzanforderungen diskutiert, also solche, die in den vorangegangenen Kapiteln auch als Echtzeitstream bezeichnet wurden und *time-sensitive* im Sinne des Standards sind.

2.2.2 Übersicht über TSN

Die IEEE befasst sich im Rahmen des Projektes IEEE 802 mit der Standardisierung von Technologien und Protokollen für lokale Netzwerke. Das darin verankerte IEEE-802.1-Projekt fokussiert sich dabei

auf Funktionen, die dem Netzwerkmanagement dienen. Deren Standardisierung ist wiederum in eine Vielzahl von Standarddokumenten innerhalb von IEEE 802.1 unterteilt, wie z.B. die Erkennung der Netzwerktopologie (IEEE 802.1AB) oder umfassende Funktionen zur Ressourcenreservierung, der Verkehrsformung, und der Priorisierung von Datenstreams (IEEE 802.1Q). Für die Entwicklung und Standardisierung neuer Funktionen zur besseren Unterstützung von Echtzeitanwendungen wurde von der zuständigen *IEEE 802.1 Working Group* zunächst die *Audio Video Broadcast Task Group* ins Leben gerufen, aus der später die *Time-Sensitive Networking Task Group* hervorging und der entsprechende Begriff des Time-Sensitive Networking geprägt wurde. Die von der Arbeitsgruppe standardisierten Funktionen erweitern entweder bestehende Standards der IEEE-802.1-Familie oder sie werden als neue Standards innerhalb der Familie definiert. Die Namensgebung der Standards innerhalb von IEEE 802.1 folgt dabei den in [59] genannten Regeln:

- Auf IEEE 802.1 folgen zwischen 1 und 4 Buchstaben
- Großbuchstaben kennzeichnen eigenständige Standards
- Kleinbuchstaben folgen ggf. auf Großbuchstaben und kennzeichnen Ergänzungen (*Amendments*) zu existierenden Standards, welche nach der Fertigstellung in die nächste Version des betroffenen Standards aufgenommen werden

Gültige Namen sind beispielsweise:

- IEEE 802.1Q (eigenständiger Standard)
- IEEE 802.1Qbv (Ergänzung zu IEEE 802.1Q)
- IEEE 802.1CB (eigenständiger Standard)

Die folgende Auflistung gibt einen ersten Überblick über die zum Verständnis dieser Arbeit hilfreichen Standards bzw. Amendments:

IEEE 802.3 [60] definiert Protokoll und Frameformat von Ethernet, Zugriffsverfahren auf ein gemeinsames Übertragungsmedium (MAC) und Übertragungstechniken bis hin zur physikalischen Schicht (z.B. *Fast Ethernet* und *Gigabit Ethernet*).

IEEE 802.1Q [61] spezifiziert die Funktionsweise von *Bridges*, welche der Verbindung mehrerer LANs dienen. Der Standard befasst sich mit Mechanismen der Frameweiterleitung (auch *MAC Relaying*) sowie der logischen Gruppierung von Geräten eines Netzwerks in *Virtual Local Area Networks* (VLANs). Da viele der TSN-Standards die Frameweiterleitung betreffen, werden bzw. wurden sie als Erweiterung zu IEEE 802.1Q formuliert.

IEEE 802.1AB [62] spezifiziert LLDP, welches dem Austausch von Gerätefunktionen und Verbindungsinformationen innerhalb eines LAN dient. In einem Echtzeitnetzwerk mit zentralem Controller kann es zur Topologieerkennung genutzt werden.

IEEE 802.1CB [63] standardisiert Funktionen zur redundanten Übertragung von Frames über mehrere Pfade im Netzwerk (FRER, von *Frame Replication and Elemination*). Ebenfalls enthalten sind Funktionen zur Streamidentifikation, welche es ermöglichen, die Zugehörigkeit von Frames zu

bestimmten Echtzeitstreams zu erkennen (z.B. bei Eintreffen in einer Bridge) und die für diesen Stream vordefinierte Verarbeitung anzuwenden (z.B. zeitgesteuerte Weiterleitung).

IEEE 802.1AS [64] definiert das *generalized Precision Time Protocol* (gPTP), welches Mechanismen zur Zeitsynchronisation von Bridges und Endgeräten umfasst. Die hiermit zur Verfügung gestellte gemeinsame Zeitbasis vernetzter Geräte ist unabdingbar für die Einhaltung eines globalen Schedules.

IEEE 802.1Qcc [65] beschreibt mögliche Konfigurationsarchitekturen für TSN und erweitert damit assoziierte Protokolle zur Konfiguration und Ressourcenreservierung, wie beispielsweise das *Stream Reservation Protocol* (SRP) und das *Multiple Registration Protocol* (MRP). Dies umfasst zentralisierte Konfigurationsmechanismen, die zur Realisierung des in Kapitel 2.1 vorgestellten Konzepts genutzt werden können.

IEEE 802.1Qbv [58] erweitert die Sende- und Weiterleitungsmechanismen von Bridges und Endgeräten um zeitgesteuerte Frameübertragungen und definiert zugehörige Konfigurationsparameter für Bridges. Die Realisierung erfolgt durch das zeitgesteuerte Öffnen und Schließen von mehreren Queues am jeweiligen Ausgangsport. Dies ermöglicht es, die in einem Schedule festgelegten Sendezeitschlitze innerhalb von Bridges und Endgeräten abzubilden.

IEEE 802.3br [66] definiert Mechanismen auf MAC-Ebene, um die Übertragung von Frames zu unterbrechen, sobald höher priorisierte Frames zur Übertragung bereitstehen (*Frame Preemption*). In Bezug auf TSN ermöglicht dies beispielsweise die bevorzugte Behandlung zeitgesteuerter Streams, da diese die Frameübertragungen anderer Verkehrsklassen unterbrechen können.

IEEE 802.1Qbu [67] definiert Konfigurationsparameter und Funktionsweisen für Bridges in Bezug auf die in IEEE 802.3br eingeführte Frame Preemption.

IEEE 802.1Qci [68] erweitert die Frameverarbeitung in Bridges um *Per-Stream Filtering and Policing* (PSFP). Dieses ermöglicht unter anderem eine feingranulare Zuweisung von Frames zu verschiedenen Queues an Bridge-Ausgangsports, welche wiederum Sendezeitschlitze mittels IEEE 802.1Qbv abbilden. Des Weiteren können Frames bei unerwarteter Ankunftszeit verworfen werden, was beispielsweise bei fehlerhaftem Verhalten von Endgeräten Folgefehler bezüglich des geplanten Schedules verhindern kann.

An dieser Stelle sei angemerkt, dass alle der oben genannten Amendments zu IEEE 802.1Q mittlerweile in IEEE 802.1Q aufgenommen worden sind. Da die ursprünglichen Dokumente der Amendments nach der Aufnahme in den Standard nicht weiter im Rahmen von Aktualisierungen gepflegt werden, sind diese obsolet. Als Referenz sollte stattdessen nur die weiterhin durch Revisionen gepflegte, aktuelle Version von IEEE 802.1Q [61] herangezogen werden. Die obsoleten Amendments und deren Bezeichnungen werden hier insbesondere deshalb eingeführt, weil es sich dabei um die in der Literatur vorherrschenden Bezeichnungen handelt. Bei Verweisen zu konkreten Inhalten werden jedoch im Folgenden stets die entsprechenden Kapitel des aktuelleren IEEE 802.1Q [61] genannt.

2.2.3 Erläuterung ausgewählter TSN-Standards

Von den zuvor genannten Standards sollen hier jene näher betrachtet werden, die das Netzwerkmanagement sowie das zeitgesteuerte Senden von Frames betreffen. Durch die Darstellung des Netzwerkmanagements von TSN soll ein besseres Bild davon vermittelt werden, wie das vorgestellte Konzept eines Echtzeitnetzwerks mit Hilfe TSN-konformer Geräte praktisch realisiert werden kann. Eine detaillierte Betrachtung der Mechanismen für das zeitgesteuerte Senden ist wiederum für die getroffenen Annahmen in der mathematischen Modellierung des Schedulingproblems von Bedeutung.

2.2.3.1 Konfigurationsarchitektur (IEEE 802.1Qcc)

Die Ausführungen in [61, § 46] (ursprünglich IEEE 802.1Qcc) befassen sich mit der Konfiguration der im Rahmen von TSN neu hinzugekommenen Funktionen von Ethernetnetzwerken. Dabei werden drei Konfigurationsmodelle vorgeschlagen. Ein Modell der vollständig verteilten Konfiguration setzt dabei primär auf das Selbstmanagement einzelner Bridges. Es ist aufgrund des Fehlens eines zentralen Controllers nur bedingt für die Offline-Planung von zeitgesteuertem Verkehr geeignet und wird daher hier nicht weiter betrachtet.

Das in Abbildung 2.2 gezeigte Modell mit zentralisierter Netzwerkkonfiguration stellt dagegen eine konkrete Realisierungsmöglichkeit des in Kapitel 2.1 vorgestellten Konzepts eines Echtzeitnetzwerks dar. In diesem werden die auf den Endgeräten laufenden Anwendungen, welche das Netzwerk zur Kommunikation verwenden, als *User* aufgefasst. Wird von einer der Anwendungen ein Echtzeitstream benötigt, so kann der Talker bzw. Listener dies unter Angabe der konkreten Anforderungen über das sogenannte *User/Network Interface* (UNI) beim Netzwerk anmelden [61, § 46.1.1]. Dabei agiert die Bridge, an dem das Endgerät angeschlossen ist, als Proxy, welcher die erhaltene Anfrage an die sogenannte *Centralized Network Configuration* (CNC) weiterleitet. Bei der CNC handelt es sich um

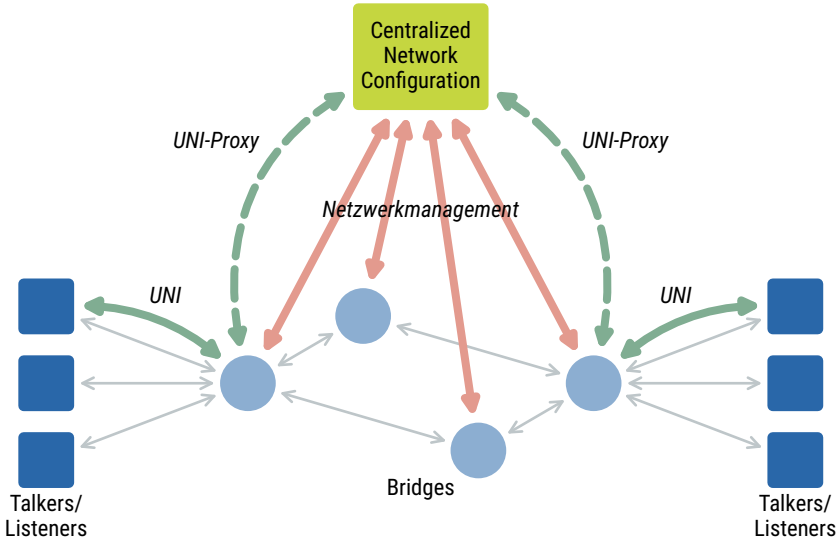


Abbildung 2.2: Konfigurationsmodell von TSN mit zentralem Netzwerkcontroller [61, § 46.1.3.2].

einen zentralen Netzwerkcontroller, der beispielsweise als Softwareanwendung auf einem Endgerät oder einer bestimmten Bridge betrieben wird. Mit Hilfe eines Netzwerkmanagement-Protokolls kann die CNC darüber hinaus eine Topologieerkennung durchführen, Bridgefähigkeiten erfassen und die Konfiguration der Bridges vornehmen. Insgesamt sind nach diesem Modell also die Voraussetzungen geschaffen, um ein Echtzeitnetzwerk entsprechend dem Ablauf aus Kapitel 2.1 zu konfigurieren. Der erfolgreiche Abschluss der Konfiguration wird über das UNI gegenüber den Anwendungen signalisiert, bevor diese ihre zeitkritische Kommunikation beginnen.

Es sei angemerkt, dass [61, §46.1.3.2] für UNI, UNI-Proxy und Netzwerkmanagement keine Protokolle vorschreibt, allerdings sinnvolle Optionen vorschlägt:

- UNI: SRP nach [61, §35]
- UNI-Proxy: MRP External Control nach [61, §12.32.4]
- Netzwerkmanagement: SNMP, NETCONF [69], RESTCONF [70]

Darüber hinaus weist der Standard darauf hin, dass Bridges ihre Konfiguration in einem nichtvolatilen Speicher ablegen können, um auch während Ausfällen der CNC eine valide Konfiguration über Neustarts hinweg zu behalten. So soll die Abhängigkeit zur CNC reduziert werden.

Das dritte, in Abbildung 2.3 gezeigte Konfigurationsmodell schlägt eine noch weitreichendere Zentralisierung vor, indem kommunizierende Anwendungen eine sogenannte *Centralized User Configuration* (CUC) erhalten. Motiviert wird dies im Standard damit, dass es in vielen Anwendungsbereichen von TSN bereits eine komplexe Aufgabe sein kann, die kommunizierenden Endanwendungen zu

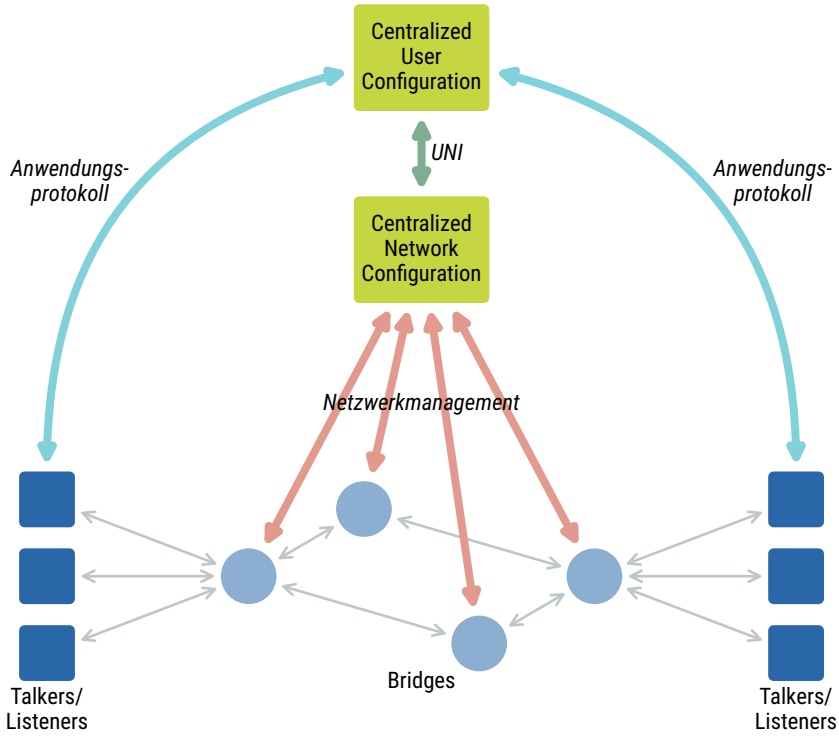


Abbildung 2.3: Vollständig zentralisiertes Konfigurationsmodell von TSN [61, §46.1.3.3].

konfigurieren und die konkreten Anforderungen der Echtzeitstreams abzuleiten. In einem solchen Fall kann die CUC eingesetzt werden, um Konfiguration und Verbindungsaufbau der Anwendungen zu unterstützen und dabei zugleich die Anforderungen benötigter Echtzeitstreams zu erfassen. Die Anforderungen werden dann von der CUC zur CNC übermittelt. Das UNI verschiebt sich dabei zwischen CUC und CNC, während die Kommunikation zwischen Anwendungen und CUC durch ein anwendungsspezifisches Protokoll abgewickelt werden soll, welches im Standard bewusst nicht näher eingegrenzt wird. Abweichend vom Modell ohne CUC wird bezüglich des UNI der Einsatz von RESTCONF in Betracht gezogen. Als Beispiel für diese Konfigurationsarchitektur sei hier die prototypische Implementierung eines Medizingerätenetzwerks in [E13] genannt, bei dem SDC als anwendungsspezifisches Protokoll zum Einsatz kommt.

2.2.3.2 Zeitgesteuertes Senden (IEEE 802.1Qbv)

Ziel des in den vorangegangenen Kapiteln beschriebenen Planungs- und Konfigurationsprozesses ist, Schedules entsprechend Abbildung 1.1 zu erstellen und in äquivalenten Bridgekonfigurationen abzubilden. Um die notwendigen Bridgekonfigurationen nachvollziehen zu können, muss die Funktionsweise der zeitgesteuerten Frameweiterleitung in Bridges betrachtet werden.

An einer Bridge werden eintreffende Frames auf Basis der aktuellen Konfiguration und nach Auswertung der relevanten Headerfelder gefiltert (d.h. ggf. verworfen) und in einer Queue an einem oder mehreren Ausgangsports eingereiht. Der genaue Filter- und Queuingvorgang wird später erläutert. Nach [61, § 8.6.8.4] (ursprünglich IEEE 802.1Qbv) besitzt eine standardkonforme Bridge hierfür bis zu acht Queues an jedem Ausgangsport, welche der Weiterleitung unterschiedlicher Verkehrsklassen mit fest zugewiesenen Prioritäten dienen. Wie in Abbildung 2.4 gezeigt, folgt auf jede dieser Queues

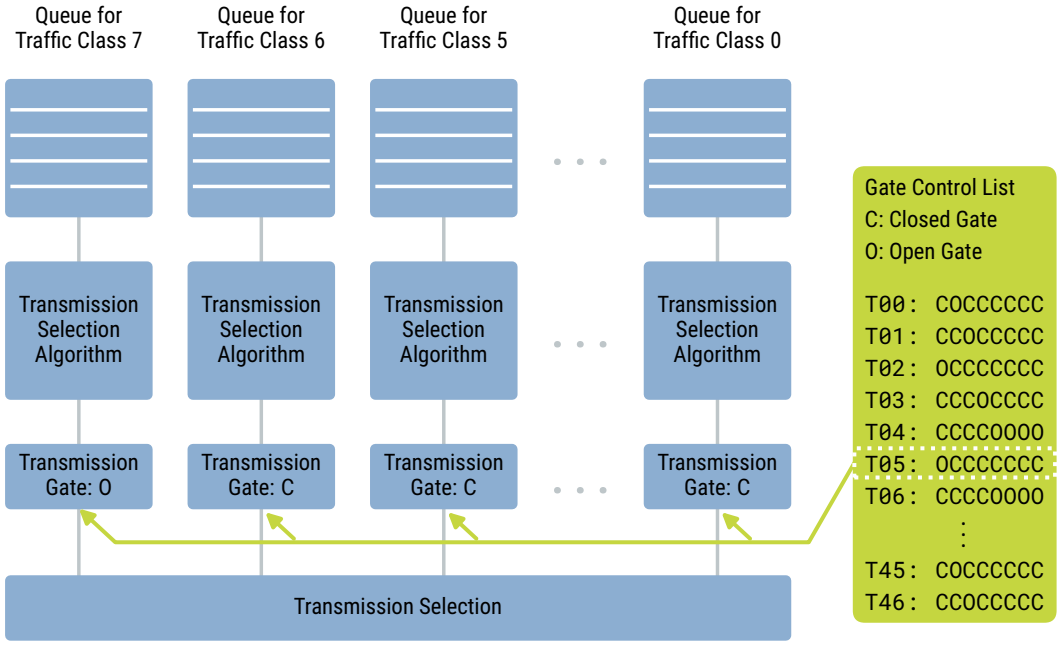


Abbildung 2.4: Queuing-Modell am Ausgangsport einer Bridge nach IEEE 802.1Qbv [61, § 8.6.8.4].

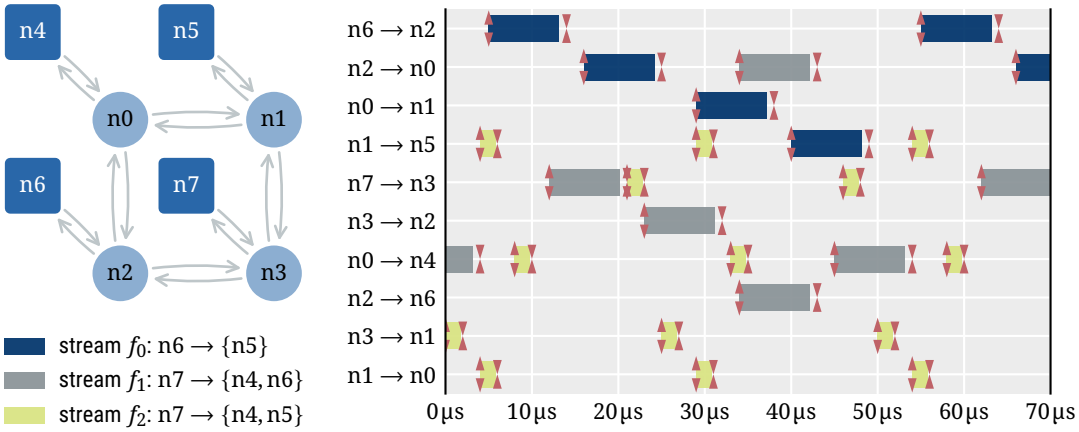


Abbildung 2.5: Gate-Events des Beispiel-Schedules aus Kapitel 1.

eines Ausgangsports ein *Transmission Selection Algorithm* (TSA). Beim TSA handelt es sich um einen für jede Queue einzeln konfigurierbaren Traffic Shaper, welcher entscheidet, ob Frames der jeweiligen Queue aktuell zum Senden bereitstehen. Im Falle des CBS ist dies beispielsweise nur genau dann der Fall, wenn die Queue nicht leer und der Credit nicht negativ ist. Neben dem CBS kann außerdem der *Asynchronous Traffic Shaper* (ATS) zum Einsatz kommen oder, im einfachsten Fall, der *Strict-Priority*-Algorithmus nach [61, §8.6.8.1], bei dem die Queue immer dann als sendebereit gilt, wenn sie nicht leer ist. Im Falle von Bridges, die zeitgesteuerten Verkehr nach [61, §8.6.8.4] unterstützen, folgt auf den TSA ein *Transmission Gate*, welches geöffnet (O) oder geschlossen (C) sein kann. Nach den Transmission Gates werden die verschiedenen Queues in der *Transmission Selection* zusammengeführt, welche bestimmt, aus welcher Queue der Frame für den nächsten Sendevorgang am Ausgangsport entnommen wird. Dabei wird immer die höchstpriorisierte Queue ausgewählt, bei der zugleich Sendebereitschaft durch den TSA signalisiert wird und das zugehörige Gate geöffnet ist.

Zur Realisierung von zeitgesteuertem Verkehr werden alle Gates durch das periodische Durchlaufen einer *Gate Control List* (GCL) entsprechend dem gewünschten Schedule geöffnet bzw. geschlossen. Die GCL existiert separat für jeden Ausgangsport einer Bridge [61, §12.29.1] und ist eine einfache Auflistung von Zeitpunkten (`T00`–`T46` in Abb. 2.4) und den in diesem Moment zu setzenden Gate-Zuständen aller Queues des jeweiligen Ausgangsports. Die GCL-Länge ist implementierungsabhängig begrenzt. Für Queues, die der Weiterleitung von zeitgesteuertem Verkehr dienen, kommt in der Regel der Strict-Priority-Algorithmus als TSA zum Einsatz und das Gate wird zum geplanten Zeitpunkt exklusiv geöffnet, d.h., alle anderen Gates werden geschlossen. Somit lässt sich ein zeitgesteuertes Senden von Frames einer bestimmten Queue realisieren. Da durch die Synchronisation von Bridges (z.B. nach IEEE 802.1AS) eine einheitliche Zeitbasis im Netzwerk zur Verfügung steht, kann durch diese GCL-Einträge ein globaler Schedule abgebildet werden. Als Beispiel wurde der Schedule aus Kapitel 1 in Abbildung 2.5 um passende Gate-Open- und Gate-Close-Events ergänzt. Zur Vereinfachung wird hier davon ausgegangen, dass alle Links[2] nur eine einzige Queue (z.B. die der

[2]Es sei angemerkt, dass hier nicht zwischen Links und Ausgangsports unterschieden wird, da zwischen diesen in Ethernet unter Nutzung von Bridges eine 1:1-Beziehung besteht.

Verkehrsklasse 7) für alle zeitgesteuerten Streams nutzen, sodass die Abbildung für jeden Link die Gate-Events dieser einen Queue zeigt. Die Gate-Close-Events fallen dabei absichtlich nicht präzise auf die Endzeitpunkte der Frameübertragungen, da hierbei zu berücksichtigen ist, dass die GCL in der Bridge nur mit begrenzter zeitlicher Auflösung verarbeitet werden kann (hier: $1\,\mu s$), wohingegen die Dauer einer Übertragung durch Framegröße und Linkbandbreite bestimmt wird und daher nicht mit diesem Zeitraster übereinstimmt. Laut [61, §8.6.8.4, §Q.2] müssen die Gate-Close-Events dabei nach dem Ende der Frameübertragungen liegen, da eine Bridge-Implementierung vor einer Frameübertragung sicherstellen muss, dass diese vor dem nächsten Gate-Close-Event der zugehörigen Queue vollständig abgeschlossen werden kann. In der Praxis werden immer dann, wenn gerade keine zeitgesteuerten Übertragungen geplant sind, die Gates anderer Verkehrsklassen geöffnet.

Ein weiteres relevantes Merkmal der dargelegten Funktionsweise ist, dass bei der zeitgesteuerten Frameübertragung einfach nach dem FIFO-Prinzip der erste Frame der höchstpriorisierten, geöffneten und sendebereiten Queue zur Übertragung ausgewählt wird. Dabei erfolgt keine weitere Prüfung der Headerfelder eines gesendeten Frames, um beispielsweise dessen Zugehörigkeit zu einem laut Schedule zu diesem Zeitpunkt vorgesehenen Echtzeitstream zu bestätigen. Ein Scheduler muss also bereits bei der Planung des zeitgesteuerten Verkehrs sicherstellen, dass stets der richtige Frame an der Spitze einer Queue steht, wenn diese geöffnet wird. Soll zwischen zwei Frames eine Sendereihenfolge erzielt werden, die von der Empfangsreihenfolge abweicht, so kann dies nur ermöglicht werden, indem mehrere Queues für den zeitgesteuerten Verkehr genutzt werden und der Scheduler außerdem entsprechende Queuezuweisungen plant. Das Beispiel in Abbildung 1.1 bzw. 2.5 wurde absichtlich so gewählt, dass es sich unter Nutzung von nur einer Queue für zeitgesteuerten Verkehr in einer GCL abbilden lässt. In komplexeren Szenarien liegt dieser Fall für gewöhnlich nicht vor, was in Kapitel 7.2 ausführlich diskutiert wird.

2.2.3.3 Interferenz zwischen Verkehrsklassen (IEEE 802.1Qbv, IEEE 802.1Qbu, IEEE 802.3br)

Nachdem die Abbildung von Schedules in GCLs diskutiert wurde, muss außerdem das Zusammenspiel mit weiteren Verkehrsklassen (d.h. nicht zeitgesteuertem Verkehr) betrachtet werden. Beim Scheduling wird an jeder Bridge und jedem Ausgangsport eine bestimmte Anzahl der Queues für den zeitgesteuerten Verkehr genutzt. Damit der geplante Schedule tatsächlich eingehalten werden kann, dürfen Streams anderer Verkehrsklassen grundsätzlich nicht den gleichen Queues zugewiesen werden, die bereits für zeitgesteuerten Verkehr genutzt werden, da sonst nicht sichergestellt ist, dass zum Zeitpunkt eines Gate-Open-Events der ursprünglich geplante Frame an der Spitze der jeweiligen Queue steht. Um neben dem zeitgesteuerten Verkehr weitere Verkehrsklassen im gleichen Netzwerk zu betreiben, ist es also zwingend erforderlich beim Scheduling einige (mindestens eine) der zur Verfügung stehenden Queues eines Ausgangsports ungenutzt zu lassen, damit diese den anderen Verkehrsklassen zur Verfügung stehen. Das Traffic Shaping dieser Verkehrsklassen kann wiederum prioritätsbasiert und mit Hilfe der verschiedenen TSAs erfolgen, während bei der Transmission Selection weiterhin der jeweilige Gate-Zustand berücksichtigt wird.

Im Folgenden soll ein weiteres potentielles Problem beschrieben werden, das auch bei Nutzung separater Queues für zeitgesteuerten Verkehr und andere Verkehrsklassen zu unerwarteten Verzöge-

rungen des zeitgesteuerten Verkehrs führen könnte. Angenommen es befindet sich zum Zeitpunkt eines Gate-Open-Events für einen geplanten zeitgesteuerten Frame bereits ein Frame einer anderen Queue in der Übertragung, so muss (sofern keine weiteren Gegenmaßnahmen getroffen werden) das Ende dieses Sendevorgangs abgewartet werden, bevor der ursprünglich geplante Frame übertragen werden kann. Für zeitgesteuerte Frames könnten somit Verzögerungen aufgrund der Frames anderer Verkehrsklassen entstehen. In TSN kann dieses Problem ausgeschlossen werden, indem die Queues anderer Verkehrsklassen geschlossen werden, wenn ein Gate-Open-Event für einen zeitgesteuerten Frame erfolgt (d.h. exklusives Öffnen des Gates). Da Bridges wie bereits erläutert nur dann eine Frameübertragung starten dürfen, wenn diese vor dem nächsten Gate-Close-Event der jeweiligen Queue abgeschlossen werden kann, wäre das Starten einer Frameübertragung somit unzulässig, wenn sich diese bis in den geplanten Zeitschlitz des zeitgesteuerten Frames erstreckt. Unerwartete Verzögerungen zeitgesteuerter Frames werden durch dieses Prinzip also zuverlässig vermieden.

Aus der beschriebenen Funktionsweise ergibt sich vor jedem geplanten Zeitschlitz des zeitgesteuerten Verkehrs ein Zeitraum, in dem keine neue Frameübertragung begonnen werden kann, da sie nicht rechtzeitig abgeschlossen werden kann. Dieser Zeitraum wird auch als *Guard Band* bezeichnet [61, § Q.1]. Die genaue Länge dieses Guard Bands hängt von der Bridge-Implementierung ab, wobei es im Wesentlichen drei Möglichkeiten gibt [61, § Q.1]:

(a) Feste Guard-Band-Länge entsprechend der Übertragungsdauer eines *Maximum Transmission Unit* (MTU)-Frames.

(b) Die Bridge entscheidet dynamisch anhand der Framelänge, ob ein Frame noch vor dem nächsten Gate-Close-Event der zugehörigen Queue vollständig übertragen werden kann.

(c) Die Bridge unterstützt *Frame Preemption* nach IEEE 802.3br und sendet ein Framefragment, bevor die Übertragung zum Zeitpunkt des Gate-Close-Events unterbrochen wird [61, § S.4].

Da es sich beim Guard Band um nicht nutzbare Sendezeit handelt und dieses somit einen Verlust von Bandbreite darstellt, sind die beiden letzteren Verfahren zu bevorzugen. Bei unterstützter Frame Preemption entspricht das Guard Band schlimmstenfalls einem $123\,\mathrm{B}$-Frame, da dieser nicht ohne Unterschreitung der Mindestframegröße fragmentiert werden kann [61, § S.2]. Die Details der Guard-Band-Ermittlung sind im Rahmen dieser Arbeit nicht von Interesse.

Die wichtigste Schlussfolgerung aus den erläuterten Mechanismen zur Unterstützung weiterer Verkehrsklassen ist, dass diese bei der Planung des zeitgesteuerten Verkehrs durch einen Scheduler nicht berücksichtigt werden müssen, da eine korrekte Implementierung und Konfiguration der Bridges sicherstellt, dass andere Verkehrsklassen keinen Einfluss auf den zeitgesteuerten Verkehr haben.

2.2.3.4 Streamidentifikation und Streamtransformation (IEEE 802.1CB, IEEE 802.1Qcc)

Die bisherigen Erläuterungen zum Queuing-Modell von TSN und der Abbildung von Sendezeitschlitzen in GCLs haben sich mit dem zeitgesteuerten Senden von Frames befasst, nachdem diese bereits in eine Queue eines Ausgangsports eingereiht wurden. Um dieses Funktionsprinzip ausnutzen zu können, muss eine Bridge die Zugehörigkeit eines eintreffenden Frames zu einem bestimmten Stream erkennen und diesen daraufhin in die laut Schedule vorgesehenen Ausgangsports und Queues

einordnen. Für die Streamerkennung sind die in IEEE 802.1CB [63] definierte *Streamidentifikation* und die in [61, § 46.1.4] (ursprünglich IEEE 802.1Qcc) beschriebene *Streamtransformation* relevant.

Die Aufgabe der Streamidentifikation (*Stream Identification Function* in [63]) ist es, die Zugehörigkeit eintreffender Frames zu vorab bekannten Streams anhand von Headerfeldern zu erkennen und den für den jeweiligen Stream in der Gerätekonfiguration hinterlegten `stream_handle` zuzuweisen. Dabei handelt es um eine einfache ganzzahlige *Identifikation* (ID) des Streams, welche lokal auf dem jeweiligen Gerät genutzt wird, um den Stream in verschiedenen Verarbeitungsschritten eindeutig identifizieren zu können. Nach [63, § 6] kann die Streamidentifikation in Bridges und Endgeräten direkt oberhalb des MAC-Layers positioniert werden und stellt allen darüber liegenden Schichten den Frame mitsamt `stream_handle` zur Verfügung. Bei Bridges bedeutet dies, dass der `stream_handle` auch im Rahmen von Weiterleitungs- und Queuing-Entscheidungen (vgl. [61, § 8.6]) berücksichtigt werden kann. Neben der *passiven* Streamidentifikation beim Empfang wird in [63, § 6] auch der Begriff der *aktiven* Streamidentifikation eingeführt. Diese kann beim Senden oder Empfangen eines Frames, welcher *durch die Streamidentifikation hindurch* zwischen MAC-Layer und höheren Schichten weitergeleitet wird, bei Bedarf Headerfelder des Frames austauschen. Konkret werden in [63, § 6] vier verschiedene Funktionen zur Streamidentifikation definiert, welche separat für jeden Port einer Bridge bzw. eines Endgerätes konfiguriert werden können [63, § 9]:

Null Stream Identification (N-SI)
: Passive Streamidentifikation mittels MAC-Zieladresse und VLAN-ID

Source MAC and VLAN Stream Identification (SV-SI)
: Passive Streamidentifikation mittels MAC-Quelladresse und VLAN-ID

Active Destination MAC and VLAN Stream Identification (ADV-SI)
: Aktive Streamidentifikation mittels MAC-Zieladresse und VLAN-ID mit potentiellem Austausch von MAC-Zieladresse, VLAN-ID und Priorität

IP Stream Identification (IP-SI)
: Passive Streamidentifikation mittels MAC-Zieladresse, VLAN-ID, *Internet Protocol* (IP)-Zieladresse, IP-Quelladresse, DSCP, IP Next Protocol, Quellport, Zielport

Die Unterstützung der verschiedenen Funktionen ist implementierungsabhängig, wobei die Null Stream Identification verpflichtend ist, sofern ein Gerät die Unterstützung von Streamidentifikation nach IEEE 802.1CB ausweist. Bei der Verarbeitung eines Frames ist außerdem eine Verkettung mehrerer Funktionen zur Streamidentifikation möglich: So kann beispielsweise ein Stream zunächst mittels IP-SI erkannt werden und daraufhin ein Austausch von Headerfeldern mit Hilfe der ADV-SI ausgelöst werden [61, § 46.1.4]. Die Funktionen der Streamidentifikation können in Bridges sowohl an Eingangsports (d.h. Empfangsrichtung) als auch Ausgangsports (d.h. Senderichtung) zum Einsatz kommen und haben eine Vielzahl verschiedener Anwendungsfälle, wie z.B. die im Folgenden erläuterte Streamtransformation.

Innerhalb eines TSN ist es möglich und laut der hier diskutierten Standards auch vorgesehen, dass Endgeräte (d.h. Sender/Empfänger) einen zeitkritischen Stream anhand anderer MAC-Adressen,

IP-Adressen und weiterer Headereinträge identifizieren als das Netzwerk (d.h. Bridges). Damit Bridges in diesem Fall die laut Schedule vorgesehenen Routen und Zeitschlitze korrekt anwenden können, kann die in [61, § 46.1.4] beschriebene Streamtransformation genutzt werden. Diese wird am Anfang einer Route (z.B. im Netzwerkinterface des Senders oder in der ersten Bridge) eingesetzt, um die entsprechenden Identifikationsmerkmale im Header auszutauschen, sodass bei der Frameweiterleitung im Netzwerk die von den Bridges erwarteten Merkmale vorliegen. Da es sich bei Bridges um vorherrschend auf Layer 2/MAC arbeitende Geräte handelt, werden hierbei vorwiegend MAC-Zieladressen und VLAN-IDs verwendet. Die verwendeten MAC-Zieladressen zur Erkennung von Echtzeitstreams sind dabei in der Regel keine Adressen physikalischer Geräte, sondern Multicastadressen (*Group Adresses* in [61]), welche den Streams bei Planung von der CNC zugewiesen und den Bridges und Endgeräten über die jeweiligen Managementprotokolle mitgeteilt werden. Die Streamtransformation kann basierend auf den zuvor definierten Funktionen zur Streamidentifikation realisiert werden: Durch die Verkettung von IP-SI und ADV-SI können Streams zunächst feingranular (d.h. bis hin zum Layer 4/Transportprotokoll) unterschieden und dann die von der CNC festlegte MAC-Zieladresse (und ggf. VLAN-ID) gesetzt werden.

Insgesamt ermöglichen die hier vorgestellten Funktionen zur Streamidentifikation und Streamtransformation das folgende Vorgehen bei der Behandlung von Echtzeitstreams:

1. Die CNC weist jedem Echtzeitstream im Rahmen der Offline-Planung eine MAC-Zieladresse (und ggf. VLAN-ID) zur Wiedererkennung des Streams im Netzwerk zu.
2. Bei der Konfiguration eines erstellten Schedules in den Bridges werden Streams anhand dieser zugewiesenen Adresse referenziert.
3. Endgeräte werden über das UNI über die zugewiesenen Adressen der für sie jeweils relevanten Streams informiert.
4. Sender setzen die zugewiesene Adresse eines Streams mittels Streamtransformation (IP-SI und ADV-SI), um die Wiedererkennung des Streams im Netzwerk zu ermöglichen.
5. Empfänger führen die umgekehrte Streamtransformation (nur ADV-SI) durch, um die intern verwendeten Erkennungsmerkmale wiederherzustellen.

Es sei angemerkt, dass weitere Details der Netzwerkkonfiguration von der genutzten Architektur (mit oder ohne CUC) und den unterstützten Funktionen und Protokollen der eingesetzten Geräte abhängen. Abweichende Vorgehensweisen sind grundsätzlich denkbar, wie beispielsweise die Nutzung einheitlicher Erkennungsmerkmale in Bridges und Endgeräten und somit der Wegfall der Streamtransformation.

2.2.3.5 Routing und Queuezuweisung (IEEE 802.1Q, IEEE 802.1Qci)

Die vorigen Erläuterungen haben gezeigt, dass TSN eine zentralisierte Konfiguration ermöglicht, Funktionen zur Wiedererkennung von Echtzeitstreams im Netzwerk definiert und an den Ausgangsports einer Bridge außerdem das zeitgesteuerte Senden erlaubt. Abschließend soll beschrieben werden, wie die in einem Schedule gegebenen Routingentscheidungen und Queuezuweisungen in

Abbildung 2.6: Weiterleitungs- und Queuing-Prozess einer Bridge nach [61, § 8.6].

einer Bridgekonfiguration abgebildet werden können. Hierfür muss der für jeden Frame innerhalb einer Bridge durchlaufene Weiterleitungs- und Queuing-Prozess betrachtet werden. Dieser wird in [61, § 8.6] definiert und ist in Abbildung 2.6 dargestellt. Im Kontext dieser Arbeit sind insbesondere das *Frame Filtering*, *Flow Metering*, *Frame Queuing*, und die *Transmission Selection* relevant.

Das Frame Filtering entscheidet über die zur Frameweiterleitung zu nutzenden Ausgangsports der Bridge und bestimmt somit die Route des Streams. Dabei wird auf die in [61, § 8.8] beschriebene *Filtering Database* (FDB) zurückgegriffen, in der sich statische und dynamische Einträge befinden können, welche die Frameverarbeitung anhand von MAC-Zieladressen und VLAN-IDs festlegen. Dynamische Einträge werden dabei auf Basis der unterstützten Mechanismen und Protokolle der Bridge automatisch erstellt. Bezüglich herkömmlicher BE-Streams ist hierbei vor allem das Lernen von MAC-Quelladressen nach [61, § 8.7] relevant. Im Kontext vorab geplanter Echtzeitstreams müssen dagegen die von einer Managementinstanz (z.B. CNC) konfigurierbaren statischen Einträge verwendet werden. Nach [61, § 8.8.1] besteht ein statischer Eintrag aus einer MAC-Adresse, VLAN-ID und *Port Map*, wobei in der Adressangabe Einzel- oder Gruppenaddressen zulässig sind (d.h. Unicast und Multicast). Ein solcher Eintrag wird auf jeden im Frame Filtering verarbeiteten Frame mit der entsprechenden MAC-Zieladresse und VLAN-ID angewendet. Die Port Map legt dabei für jeden Ausgangsport der Bridge fest, ob eine Weiterleitung über diesen Port erfolgen soll, sodass eine Weiterleitung auf mehreren Ports zwecks Multicast problemlos abgebildet werden kann. In statischen Filtereinträgen für Echtzeitstreams ist die MAC-Zieladresse zu verwenden, welche zuvor bei der

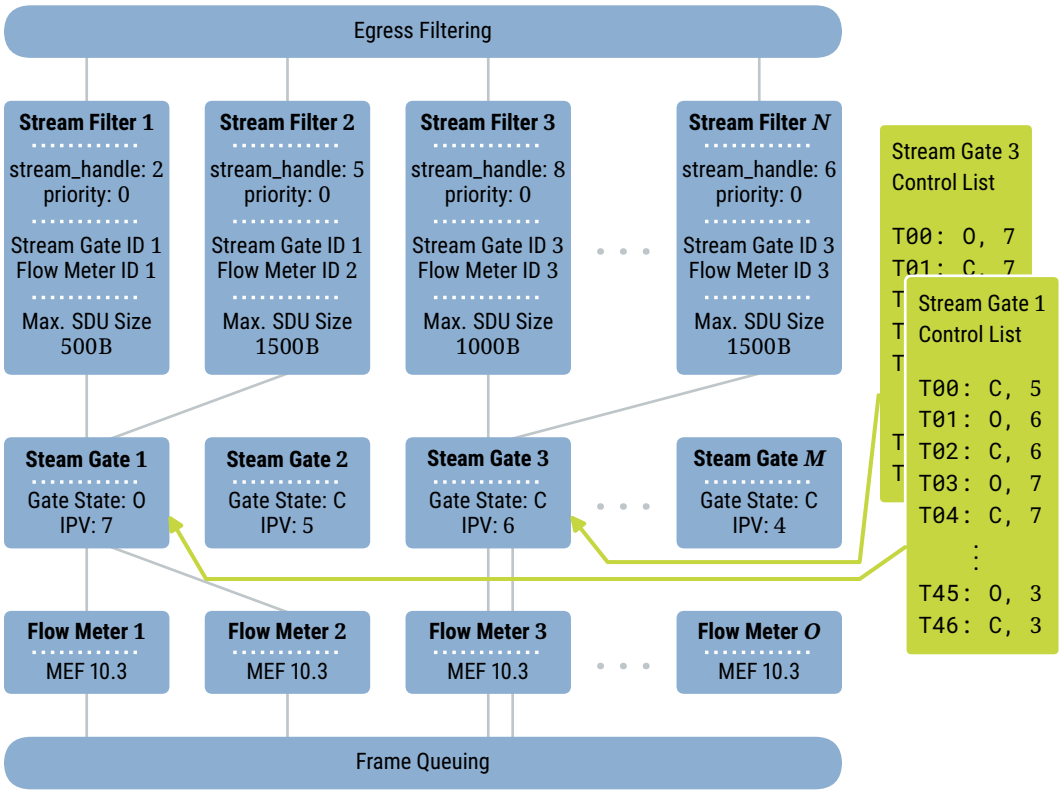


Abbildung 2.7: Per-Stream Filtering and Policing nach [61, § 8.6.5.2].

Planung von der CNC für den jeweiligen Stream vorgesehen wurde und vom Sender ggf. mittels Streamtransformation im Frameheader eingetragen wird.

Für die Queuezuweisung zu einer der maximal acht Queues auf dem jeweiligen Ausgangsport (vgl. Abb. 2.4) muss das Flow Metering betrachtet werden, in dem auch PSFP nach [61, § 8.6.5.2] angesiedelt ist. Bei PSFP handelt es sich um eine Frameverarbeitung, die sequentiell durch *Stream Filter*, *Stream Gates* und *Flow Meter* durchgeführt wird. Wie in Abbildung 2.7 gezeigt, existieren von jeder dieser drei Komponenten mehrere Instanzen, welche jeweils durch ganzzahlige IDs identifiziert werden. Ein Stream kann wiederum über das `stream_handle` (welches beim Empfang eines Frames mittels Streamidentifikation festgestellt wird) und seine Priorität mit einem Stream Filter assoziiert sein, während dieser festlegt, welches Stream Gate und Flow Meter durchlaufen werden müssen. Die drei Komponenten haben folgenden Funktionsumfang:

Stream Filter ermöglichen die Angabe einer maximalen *Service Data Unit* (SDU), welche zum Verwerfen von Frames mit größerer Payload als dem angegebenen Maximalwert führt. Außerdem wird das zu nutzende Stream Gate mittels ID festgelegt. Optional kann ein zu nutzendes Flow Meter mittels ID angegeben werden (Flow Metering inaktiv sofern nicht angegeben).

Stream Gates können zeitgesteuert geöffnet und geschlossen werden. Frames, die bei geschlossenem Gate eintreffen, werden verworfen. Für weitergeleitete Frames (d.h. geöffnetes Gate) kann ein *Internal Priority Value* (IPV) konfiguriert werden, welcher in der folgenden Frameverarbeitung

(z.B. Queuing am Ausgangsport) anstelle der ursprünglichen Priorität der Frames genutzt wird. Falls kein IPV definiert wird, wird im Folgenden die ursprüngliche Framepriorität genutzt.

Flow Meter ermöglichen die Angabe von Burst- und Bandbreitenprofilen entsprechend der Mechanismen der technischen Spezifikation MEF 10.3 [71, §12.1]. Überschreiten die durch das Flow Meter geleiteten Frames das konfigurierte Profil, so können Frames verworfen werden. Die entsprechenden Funktionen sind im Rahmen dieser Arbeit nicht relevant.

Wie in der Abbildung zu sehen, besitzt jedes Stream Gate eine eigene Stream-GCL, welche zyklisch wiederholt wird und aus einer Reihe von Zeitpunkten und dem jeweils zu setzenden Gatezustand und IPV besteht. Bridges ohne Unterstützung für PSFP können eine einfachere Stream-Klassifizierung und Metering nach [61, §8.6.5.1] implementieren. Die Unterscheidung von Streams kann in diesem Fall anhand MAC-Quelladresse, MAC-Zieladresse, VLAN-ID und Priorität (d.h. ohne `stream_handle` der Streamidentifikation) erfolgen, woraufhin auch hier ein Flow Metering nach MEF 10.03 folgen kann. Diesem einfacheren Mechanismus fehlt also die zuvor beschriebene Funktionalität der Stream Gates: Prioritätsanpassungen und zeitgesteuertes Verwerfen von Frames.

Entsprechend dem in Abbildung 2.6 gezeigten Weiterleitungsprozess folgt nach dem Flow Metering das Queuing weitergeleiteter Frames auf allen Ausgangsports, die laut der vorangegangenen Schritte (insbesondere Frame Filtering) für den jeweiligen Frame verwendet werden sollen. Dabei liegt an jedem dieser Ausgangsports das in Abbildung 2.4 eingeführte Queuing-Modell vor, wobei sich die Anzahl der Verkehrsklassen bzw. Queues implementierungsabhängig und pro Ausgangsport unterscheiden kann (bei maximal acht Queues). Die Zuordnung von Frames zu den vorhandenen Queues erfolgt anhand dessen Priorität und einer portspezifischen Mapping-Tabelle von Priorität zu Queuenummer. Die genutzte Framepriorität ist dabei entweder die ursprünglich im Frameheader angegebene, oder der zugewiesene IPV, sofern ein Stream Gate mit definiertem IPV durchlaufen wurde. Nach dem Frame-Queuing folgt im Weiterleitungsprozess ein Queue-Management, welches weitere Regeln zum Verwerfen von Frames definiert und hier nicht weiter diskutiert werden soll. Im letzten Schritt folgt die Transmission Selection einschließlich TSAs und Transmission Gates, welche bereits ausführlich erläutert wurde.

Insgesamt ist das in diesem Unterkapitel vorgestellte Frame Filtering also geeignet, das durch einen Schedule vorgegebene Routing abzubilden, während das Flow Metering die ebenfalls gegebenen Queuezuweisungen von Streams an den jeweiligen Ausgangsports ermöglicht. Bezüglich dieser Queuezuweisungen ergeben sich unterschiedliche Einschränkungen je nach Unterstützung von PSFP:

- Sofern Streamidentifikation nach [63] und PSFP nach [61, §8.6.5.2] von einer Bridge unterstützt wird, erlaubt eine entsprechende Konfiguration des `stream_handle`s und des damit assoziierten Stream Gates (einschließlich IPV) eine explizite Queuezuweisung auf der jeweiligen Bridge.
- Ohne Streamidentifikation und PSFP kann eine Queuezuweisung durch passendes Setzen der Framepriorität (z.B. am Sender) erfolgen. Eine wesentliche Einschränkung besteht in diesem Fall darin, dass die Queuezuweisung in mehreren Bridges aneinander gekoppelt ist. Die explizite Queuezuweisung pro Bridge ist aufgrund fehlendem IPV nicht möglich.

Anhand dieser Routing- und Queuingmechanismen ist außerdem ersichtlich, dass es in der nachfolgenden Transmission Selection ausreichend ist, das zeitgesteuerte Senden zu realisieren (siehe Kapitel 2.2.3.2), ohne dabei eine nochmalige Streamidentifikation mit Hilfe des Frameheaders durchzuführen: Bei korrekter Konfiguration der Bridges und korrektem Sendeverhalten der Endgeräte sind die Zustände der für zeitgesteuerten Verkehr vorgesehenen Queues für jeden Zeitpunkt vorhersagbar, sodass auch bekannt ist, welcher Frame sich zum Zeitpunkt eines geplanten Gate-Open-Events an der Spitze der jeweiligen Queue befindet.

Kapitel 3

Das Planungsproblem zeitgesteuerter Übertragungen (TT-RSP)

Das in Kapitel 2.1 vorgestellte Konzept eines Echtzeitnetzwerks hat verdeutlicht, dass die Planung von Streams mit harten Echtzeitanforderungen offline erfolgen muss, also vor Beginn der Kommunikation. Dabei liegen dem zentralen Controller, der die Planungsalgorithmen ausführt, vollständige Topologie- und Streaminformationen vor. Bei der Planung der hier betrachteten zeitgesteuerten Übertragungen muss der Controller die Routen und Sendezeitschlitze aller benötigten Echtzeitstreams festlegen, sodass deren QoS-Anforderungen eingehalten werden.

Neben dem zeitgesteuerten Verkehr von TSN unterliegen auch die zeitgesteuerten Verkehrsklassen weiterer IE-Systeme wie z.B. Profinet und TTEthernet einem ähnlichen Planungsproblem für Routen und Zeitschlitze. Zur Beschreibung der grundlegenden Anforderungen bei der Planung zeitgesteuerter Übertragungen wird in diesem Kapitel daher zunächst ein allgemeines Time-Triggered Routing and Scheduling Problem (TT-RSP) formuliert, ähnlich zur Problembeschreibung in den eigenen Arbeiten [E14, E15]. In diesem werden nur Anforderungen erfasst, die gleichermaßen für TSN, Profinet und TTEthernet gelten, während spezielle Anforderungen (z.B. durch das Queueing-Modell von TSN) später als Erweiterung formuliert werden. Wie in Kapitel 1.2 eingeführt, werden Planungsalgorithmen zur Lösung von TT-RSP als Scheduler bezeichnet. Die im Folgenden gezeigte formale Beschreibung von TT-RSP wird unterteilt in die vom Scheduler benötigten Eingangsdaten sowie das von diesem zu berechnende Ergebnis.

3.1 Eingangsdaten von TT-RSP

Die Eingangsdaten des Schedulers werden von Mechanismen der Netzwerkerkennung zur Verfügung gestellt und umfassen die *Netzwerktopologie* und die benötigten zeitkritischen Datenstreams. Die benötigten Datenstreams werden zusammengefasst auch als *Verkehrsmuster* bezeichnet. In TSN werden diese Informationen beispielsweise mit Hilfe einer zentralen Konfigurationsarchitektur erfasst (siehe Kapitel 2.2.3.1) und zur Planung in der CNC zusammengeführt.

Netzwerktopologie Die Netzwerktopologie wird durch einen gerichteten Graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, zugehörige Bridgeeigenschaften $(v_i)_{i \in \mathcal{V}^{br}}$ und Kanteneigenschaften $(e_m)_{m \in \mathcal{E}}$ beschrieben.[1] Die Netzwerkknoten $\mathcal{V} := \mathcal{V}^{br} \cup \mathcal{V}^{es}$ sind unterteilt in Bridges $\mathcal{V}^{br}$, welche der Weiterleitung von Frames dienen, sowie Endgeräte $\mathcal{V}^{es}$, die als Sender oder Empfänger von Datenstreams agieren. Mit jeder Bridge $i \in \mathcal{V}^{br}$ ist ein Tupel $v_i := (v_i.hb, v_i.prc)$ assoziiert, das die Bridgeeigenschaften beschreibt. Hierbei gibt $v_i.hb$ an, wie viele Headerbytes eines Frames von der Bridge empfangen werden müssen, bevor anhand der in der Bridge gespeicherten Regeln eine Entscheidung über die Frameweiterleitung getroffen werden kann (z.B. Auswahl des korrekten Ausgangsports). Eine typische Bridge mit *Cut-Through* (CT)-Technologie kann beispielsweise durch $v_i.hb := 24$ modelliert werden, was dem Empfang von Präambel und *Start-of-Frame Delimiter* (SFD) sowie der Frame-Headerfelder *MAC Source*, *MAC Destination* und *VLAN Tag* entspricht. Für eine Bridge mit *Store-and-Forward* (SF)-Technologie kann durch $v_i.hb := \infty$ angezeigt werden, dass der komplette Frame empfangen werden muss, bevor eine Weiterleitungsentscheidung getroffen werden kann. Die aus diesem Empfang der benötigten Headerbytes entstehende Verzögerung hängt von der Geschwindigkeit der Netzwerkverbindung ab, von der der Frame empfangen wird (siehe Kapitel 3.2). Die Verzögerung, die zusätzlich durch die Frameverarbeitung in der Bridge entsteht, wird durch $v_i.prc$ angegeben. Die Netzwerkverbindungen zwischen den Knoten werden als *gerichtete* Kanten $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ abgebildet, da grundsätzlich von der Nutzung von *full-duplex* Ethernet ausgegangen wird. Dementsprechend ist für die beiden Kanten $m := (i, j)$ und $n := (j, i)$ einer Netzwerkverbindung eine voneinander unabhängige Ressourcenreservierung möglich. Jede Kante $m \in \mathcal{E}$ besitzt ebenfalls ein zugehöriges Eigenschaftstupel $e_m := (e_m.ls, e_m.prp)$. Hierbei gibt $e_m.ls$ die Geschwindigkeit einer Netzwerkverbindung an, während $e_m.prp$ die im Wesentlichen von der Leitungslänge abhängige Signallaufzeit spezifiziert.

Verkehrsmuster Das Verkehrsmuster umfasst die von den Endgeräten geforderten zeitkritischen Datenstreams. Es wird durch eine Menge von Streams F und deren zugehörige Eigenschaften $(f_k)_{k \in F}$ modelliert. Hierbei wird davon ausgegangen, dass Datenstreams, die mittels zeitgesteuerten Bridgemechanismen weitergeleitet werden sollen, ein periodisches Sendeverhalten aufweisen und dabei stets eine vordefinierte Datenmenge verschicken. Dementsprechend wird jeder Stream $k \in F$ durch ein Eigenschaftstupel $f_k := (f_k.src, f_k.dsts, f_k.ct, f_k.fs, f_k.ml)$ beschrieben. Ein Stream besitzt also einen einzigen Sender $f_k.src$, während in $f_k.dsts$ ein oder mehrere Empfänger angegeben und somit auch Multicaststreams modelliert werden können. Die Zykluszeit $f_k.ct$ gibt dabei das periodische Sendeintervall an, mit dem der Sender Frames der Größe $f_k.fs$ verschickt. Bei $f_k.fs$ soll es sich hier per Definition um die Layer-2-Framegröße handeln, also die Framegröße einschließlich MAC-Header und CRC-Checksumme, aber ohne die üblicherweise dem Layer 1 zugeordnete Präambel, SFD und *Inter-Frame Gap* (IFG). Die maximal zulässige Ende-zu-Ende (E2E)-Latenz des Streams ist in $f_k.ml$ erfasst und bezieht sich auf den maximal erlaubten Zeitabstand zwischen dem Beginn der Übertragung am Sender und dem Eintreffen des vollständigen Frames bei allen Empfängern.

Szenario Die Kombination aus Netzwerktopologie und Verkehrsmuster wird im Folgenden auch als *Szenario* bezeichnet.

[1] Die Schreibweise $(e_m)_{m \in \mathcal{E}}$ definiert eine *Familie*, die jedem Element m der *Indexmenge* $\mathcal{E}$ ein Objekt e_m zuweist.

$$
\begin{aligned}
&\mathcal{V}^{br} := \{\mathrm{n0}, \mathrm{n1}, \mathrm{n2}, \mathrm{n3}\} \\
&\mathcal{V}^{es} := \{\mathrm{n4}, \mathrm{n5}, \mathrm{n6}, \mathrm{n7}\} \\
&\mathcal{E} := \big\{(\mathrm{n0}, \mathrm{n1}), (\mathrm{n1}, \mathrm{n0}), (\mathrm{n1}, \mathrm{n3}), (\mathrm{n3}, \mathrm{n1}), \\
&\qquad (\mathrm{n3}, \mathrm{n2}), (\mathrm{n2}, \mathrm{n3}), (\mathrm{n2}, \mathrm{n0}), (\mathrm{n0}, \mathrm{n2}), \\
&\qquad (\mathrm{n0}, \mathrm{n4}), (\mathrm{n4}, \mathrm{n0}), (\mathrm{n1}, \mathrm{n5}), (\mathrm{n5}, \mathrm{n1}), \\
&\qquad (\mathrm{n3}, \mathrm{n7}), (\mathrm{n7}, \mathrm{n3}), (\mathrm{n2}, \mathrm{n6}), (\mathrm{n6}, \mathrm{n2})\big\} \\
\forall i \in \mathcal{V}^{br} \quad & v_i := (\infty, 1\,\mu\mathrm{s}) \\
\forall m \in \mathcal{E} \quad & e_m := (1000\,\mathrm{Mbit/s}, 1\,\mu\mathrm{s}) \\
&F := \{0, 1, 2\} \\
&f_0 := (\mathrm{n6}, \{\mathrm{n5}\}, 50\,\mu\mathrm{s}, 1000\,\mathrm{B}, 50\,\mu\mathrm{s}) \\
&f_1 := (\mathrm{n7}, \{\mathrm{n4}, \mathrm{n6}\}, 50\,\mu\mathrm{s}, 1000\,\mathrm{B}, 50\,\mu\mathrm{s}) \\
&f_2 := (\mathrm{n7}, \{\mathrm{n4}, \mathrm{n5}\}, 25\,\mu\mathrm{s}, 200\,\mathrm{B}, 50\,\mu\mathrm{s})
\end{aligned}
$$

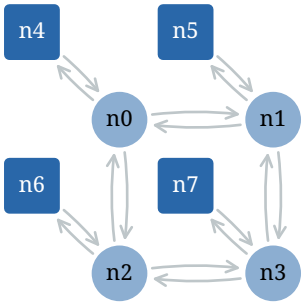


Abbildung 3.1: Formale Spezifikation des Beispiels aus Abbildung 2.5.

Als Beispiel für die formale Beschreibung der Eingangsdaten zeigt Abbildung 3.1 die Spezifikation des bisher genutzten Beispielszenarios (vgl. Abb. 1.1 und 2.5). In diesem wird eine Topologie aus SF-Bridges mit einer Verarbeitungszeit von $v_i.prc := 1\,\mu\mathrm{s}$ und Links mit einer Bandbreite von $e_m.ls := 1000\,\mathrm{Mbit/s}$ sowie einer Verzögerung $e_m.prp := 1\,\mu\mathrm{s}$ verwendet.

3.2 Spezifikation gültiger Ergebnisse für TT-RSP

Das von einem Scheduler zu berechnende Ergebnis muss jedem Stream $k \in F$ eine Route vom Sender $f_k.src$ zu allen Empfängern $f_k.dsts$ sowie zugehörige Übertragungszeitschlitze entlang dieser Route zuweisen. Weitere Details der gesuchten Lösung werden im Folgenden spezifiziert und in Abbildung 3.2 dargestellt.

Routingbedingungen Für die Route eines Streams (auch als Pfad bezeichnet) gilt die offensichtliche Anforderung, dass sie den Sender durch zusammenhängende, in Senderichtung ausgerichtete Netzwerklinks mit allen Empfängern verbinden muss. Für Multicaststreams kann an beliebigen Knoten der Topologie eine Abzweigung erfolgen, indem der Frame an einer Bridge über eine Kante empfangen und über zwei Kanten weitergeleitet wird.

Ressourcenbedingungen Da auf einem Netzwerklink zu jedem Zeitpunkt nur eine Frameübertragung stattfinden kann, müssen die auf demselben Link geplanten Sendezeitschlitze der Streams überlappungsfrei (d.h. konfliktfrei) sein. Dabei sind alle periodischen Wiederholungen bis zur Hyperperiode des Schedules zu beachten. Die Länge sl_{km} eines Zeitschlitzes von Stream k auf Kante m ist dabei an die Framegröße $f_k.fs$ und die Geschwindigkeit $e_m.ls$ der jeweiligen Kante anzupassen und muss außerdem notwendige Layer-1-Elemente wie Präambel, SFD und IFG berücksichtigen:

$$sl_{km} := \frac{f_k.fs + pre + sfd + ifg}{e_m.ls} .$$

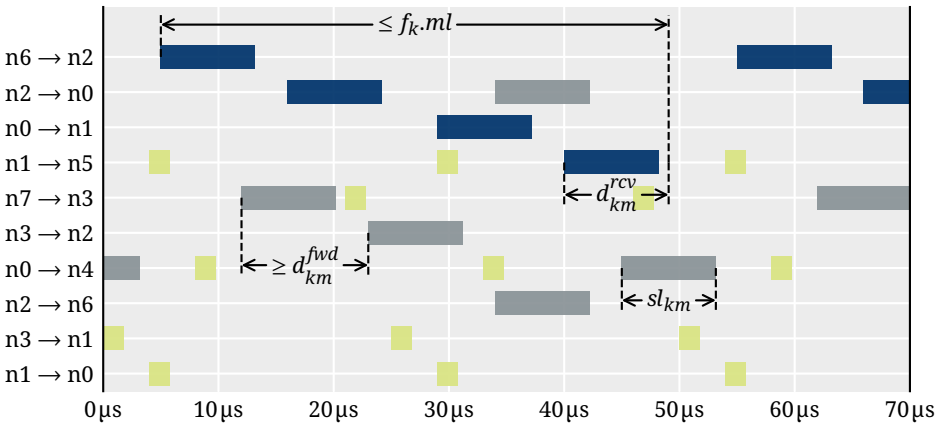


Abbildung 3.2: Grundlegende Anforderungen an Schedulingergebnisse. Beispielszenario nach Abb. 3.1.

Pfadscheduling Entlang des Pfades eines Streams sind die zeitlichen Abhängigkeiten einzuhalten. Die festgelegten Startzeitpunkte der Sendezeitschlitze des Streams k für zwei aufeinanderfolgende Links müssen also *mindestens* eine zeitliche Verschiebung d^{fwd}_{km} aufweisen, die den Verzögerungen durch den ersten Link $m := (i, j)$ und die durchlaufene Bridge $j := m_1$ entspricht:[2]

$$d^{fwd}_{km} := \frac{\min(v_j.hb, f_k.fs + pre + sfd)}{e_m.ls} + e_m.prp + v_j.prc \ .$$

Hierbei gibt $\min(v_j.hb, f_k.fs + pre + sfd)/e_m.ls$ die Übertragungszeit der Headerbytes an, die von Bridge j benötigt werden, um eine Weiterleitungsentscheidung zu treffen, $e_m.prp$ ist die Leitungsverzögerung bedingt durch die Leitungslänge des Links m und $v_j.prc$ ist die Verarbeitungszeit in der Bridge j. Durch das Setzen von $v_j.hb$ entsprechend der Bridgespezifikation im Falle von CT-Bridges und $v_j.hb := \infty$ für SF-Bridges deckt die Formel beide Bridgetechnologien ab. Die Bedeutung von d^{fwd}_{km} kann auch folgendermaßen aufgefasst werden: Beginnt die Übertragung eines Frames von Stream k auf Link m zum Zeitpunkt t^s, so kann die folgende Übertragung auf einem beliebigen Ausgangslink von Bridge m_1 frühestens zum Zeitpunkt $t^s + d^{fwd}_{km}$ gestartet werden. Spätere Übertragungszeitpunkte sind außerdem zulässig, da der Frame in der Bridge zwischengespeichert werden kann (z.B. in der Queue am Ausgangsport).

Anwendungsbedingungen Weiterhin müssen die festgelegten Zeitschlitze von k die E2E-Latenz $f_k.ml$ erfüllen. Wie zuvor definiert handelt es sich dabei um eine relative Latenzschranke in Bezug zum Sendebeginn am Sender, welche zudem für alle periodischen Wiederholungen innerhalb der Hyperperiode einzuhalten ist.

Granularität Alle von einem Scheduler geplanten Zeitpunkte, an denen eine Aktion durch ein Gerät auszuführen ist (z.B. Sendebeginn an Bridges und Endgeräten), unterliegen einer begrenzten zeitlichen Auflösung gr und benötigen daher eine ganzzahlige Repräsentation. In Abbildung 3.2 liegt die zeitliche Auflösung beispielsweise bei $gr := 1\,\mu s$.

[2] Wie in der Mathematik für Tupel üblich, kennzeichnen m_0 und m_1 hier das erste bzw. zweite Element des Tupels m

Eine präzisere Ausformulierung der beschriebenen Bedingungen soll an dieser Stelle nicht erfolgen, da diese zu großen Teilen redundant zum ILP in Kapitel 6 wäre. Als Hilfsausdruck bei der Modellierung wird in dieser Arbeit außerdem die Verzögerungszeit

$$d_{km}^{rcv} := \frac{f_k.fs + pre + sfd}{e_m.ls} + e_m.prp$$

definiert. Diese gibt an, wie lange nach dem Übertragungsbeginn eines Frames von Stream k auf Link m gewartet werden muss, bis der *vollständige* Frame am Knoten m_1 empfangen wurde und zur Weiterverarbeitung (z.B. durch eine Anwendung) zur Verfügung steht. Innerhalb dieses Ausdrucks ist $e_m.prp$ dafür verantwortlich, dass sich d_{km}^{rcv} in der Darstellung in Abbildung 3.2 sichtbar über das Ende des markierten Zeitschlitzes hinaus erstreckt: Da die Darstellung die Sendeaktivität auf dem entsprechenden Link kennzeichnet, muss nach Ende des Sendevorgangs noch die Verzögerungszeit des Links abgewartet werden, bevor die Verarbeitung im Empfänger erfolgen kann. Im Beispiel war diese mit $e_m.prp := 1\,\mu s$ angegeben.

Die bisher genannten Bedingungen gelten grundsätzlich für zeitgesteuerte Übertragungen. Da es sich um eine Offline-Planung handelt, unterliegt der Scheduler bei der Berechnung passender Schedules trotz des Anwendungsbereichs für zeitkritische Kommunikation in der Regel keinen strengen Anforderungen bezüglich der Rechenzeit. In Abhängigkeit von Zieltechnologie und Anwendungsbereich des IE-Systems können sich allerdings weitere Einschränkungen bezüglich des gesuchten Schedules ergeben:

Jitter
: Sofern Jitter in der Kommunikationslatenz unzulässig ist, müssen alle zyklischen Wiederholungen von Sendezeitschlitzen exakt äquidistant (entsprechend $f_k.ct$) positioniert werden. Andernfalls kann eine individuelle Positionierung von Wiederholungen innerhalb der Hyperperiode weitere Schedulingmöglichkeiten bieten.

Queueingmodelle
: Je nach Bridgetechnologie ist *Reordering* (d.h. unterschiedliche Sende- und Empfangsreihenfolge von Frames) bei der Frameweiterleitung durch Bridges unmöglich (z.B. TSN mit einer Queue für zeitgesteuerten Verkehr), eingeschränkt möglich (z.B. TSN mit mehreren Queues für zeitgesteuerten Verkehr) oder unbegrenzt möglich (z.B. TTEthernet).

Frameisolation
: Insbesondere in TSN kann der Schutz vor Fehlerfällen durch Interferenz von Frames unterschiedlicher Streams eine zusätzliche zeitliche Isolation von Frames oder passende Queuezuweisungen erfordern (siehe Kapitel 7.2).

Die jeweils zutreffenden zusätzlichen Einschränkungen werden im Laufe der Arbeit bei der Formulierung von Lösungsverfahren für TT-RSP konkret angegeben.

Kapitel 4

Benchmarking von Schedulern für zeitgesteuerte Übertragungen

Das in Kapitel 3 vorgestellte TT-RSP führt in der Praxis häufig zu problematischen Rechenzeiten. Da ein entsprechender Scheduler nach jeder Änderung der Netzwerktopologie oder der kommunizierenden Anwendungen erneut ausgeführt werden muss, kommt es während des Entwurfsprozesses realer Echtzeitnetzwerke (z.B. in der Fabrikautomation) regelmäßig zu Wartezeiten. Hierdurch wird nicht nur der ursprüngliche Entwurfsprozess komplexer Netzwerke signifikant verlangsamt, sondern auch die dynamische Anpassung von Netzwerken an sich ändernde Anforderungen, beispielsweise durch das Hinzufügen neuer Geräte und Anwendungen, wird erschwert. Bei der Entwicklung von effizienteren Schedulern, die zur Beschleunigung dieses Entwurfsprozesses beitragen, ist daher ein objektiver und detaillierter Performancevergleich verschiedener Scheduler unerlässlich.

In diesem Kapitel sollen daher die in der Literatur vorhandenen Bewertungsverfahren für Scheduler vorgestellt, deren Schwachstellen analysiert und darauf basierend ein eigenes, umfangreiches Benchmarkingverfahren etabliert werden. Außerdem wird der Einfluss der Testplattform (Hardware und Software) und deren Konfiguration analysiert und daraus eine sinnvolle Konfiguration für den Vergleich verschiedener Planungsalgorithmen abgeleitet. Bei den hier vorgestellten Verfahren handelt es sich um eine Weiterentwicklung der eigenen Arbeit [E15]. Die konkreten Beiträge umfassen

- einen *Szenariogenerator* zur Erstellung generischer Eingangsdaten zum Testen von Schedulern für TT-RSP,
- die Strukturierung von Szenarien in *Testcases*, die der Hervorhebung von Stärken und Schwächen der Scheduler dienen,
- die *Parametrisierung* der Testcases, um das Benchmarking aktueller Scheduler in angemessener Zeit zu ermöglichen,
- und eine Analyse des *Einflusses der Benchmarkingplattform* und deren Konfiguration auf die Benchmarkingergebnisse.

Die Präsentation des Benchmarkingverfahrens ist der Vorstellung eigener Scheduler für TT-RSP vorangestellt, da deren detaillierte Analyse in Kapitel 6 kontinuierlich von dem hier eingeführten Benchmarking Gebrauch macht.

4.1 Analyse bestehender Bewertungsverfahren

Bereits vor dem Aufkommen von TSN wurden Scheduler für ähnliche Problemstellungen wie TT-RSP in Forschungsarbeiten veröffentlicht, beispielsweise mit dem Fokus auf Profinet [2] oder TTEthernet [3]. Seit Annahme des TSN-Drafts IEEE 802.1Qbv für das zeitgesteuerte Weiterleiten von Frames in Bridges hat diese Forschungsrichtung ein massives Interesse erfahren, was sich an einer Vielzahl von Publikationen in diesem Bereich zeigt [4]. In diesen Veröffentlichungen wird dabei in der Regel auch ein Bewertungsverfahren für die entwickelten Scheduler vorgestellt, das die Leistungsfähigkeit der Scheduler und ggf. die Güte der gefundenen Lösungen aufzeigen soll. Diese Bewertung erfolgt jedoch in der Literatur anhand unterschiedlicher Performanceindikatoren, Messmethoden und Scheduler-Eingangsdaten. Im Folgenden wird daher ein Überblick über diese Aspekte gegeben, bevor Schwachstellen aufgezeigt und Ziele für ein systematisches Benchmarking von Schedulern formuliert werden.

4.1.1 Performanceindikatoren und Messmethodik

Zur Bewertung der Leistungsfähigkeit von Schedulern werden in der Literatur verschiedene Metriken eingeführt. In dieser Arbeit werden diese unterteilt in *grundlegende Performanceindikatoren* (BPIs, von *Basic Performance Indicator*) und *Qualitätsindikatoren* (QIs). Folgende Metriken werden hier als BPIs aufgefasst:

Rechenzeit (rt*)* beschreibt die Zeit, die ein Scheduler zum Lösen einer Probleminstanz benötigt. Je nach Scheduler bzw. dessen konkreter Konfiguration handelt es sich dabei um die Rechenzeit bis zum Finden der ersten gültigen Lösung oder einer optimierten Lösung. Im Falle optimierter Lösungen schließt dies somit auch die Rechenzeit eines Optimierungsvorgangs ein, zum Teil weit über das Finden der ersten Lösung hinaus.

Schedulability (sab*)* gibt an, wie viele Probleminstanzen einer gegebenen Menge von Probleminstanzen erfolgreich gelöst werden konnten. Da die Schedulability in der Regel unter Nutzung eines Zeitlimits pro Probleminstanz gemessen wird, ist sie eng mit der durchschnittlichen Rechenzeit eines Schedulers verknüpft. Ein weiterer wichtiger Einflussfaktor auf die Schedulability ist die Lösungsraumabdeckung eines Scheduler. Viele heuristische Verfahren aus der Literatur schließen im Zuge von Modellvereinfachungen gültige Lösungen aus dem von ihnen durchsuchten Lösungsraum aus, was zur einer verringerten Schedulability führen kann.

Stream Schedulability ($\mathrm{sab}^{\mathrm{st}}$*)* bezieht sich im Gegensatz zur Schedulability auf eine einzige Probleminstanz und beschreibt, wie viele der Streams eines einzigen Verkehrsmusters geplant werden konnten. Es handelt sich somit um eine Beurteilung von Teillösungen. Ob Teillösungen überhaupt betrachtet und als Ergebnis zurückgegeben werden, unterscheidet sich je nach Scheduler.

Während sich die BPIs mit Lösbarkeit und Rechenzeit befassen, beschreiben die QIs die Güte von gefundenen Lösungen. Aus der Literatur bekannt sind folgende QIs:

Streamlatenzen (lt*):* In zeitgesteuerten Netzwerken werden die E2E-Latenzen von Streams durch das Routing sowie möglicherweise unvermeidbares Buffering in Bridges bestimmt. Während eine Mindestanforderung an die Latenz in der Regel durch Streamparameter ($f_k.ml$) gegeben ist, kann eine Optimierung der Latenzen positive Auswirkungen auf die Anwendung haben, wie z.B. eine verbesserte Regelgüte einer Regelungsanwendung [20]. Daher können die erzielten Streamlatenzen zur Beurteilung der Güte eines Schedules herangezogen werden.

Pfadlängen (pl*):* Anstelle der Streamlatenzen werden in einigen Arbeiten die Pfadlängen in Form der Hopanzahl bewertet. In vielen Fällen ist diese Metrik eng mit den Streamlatenzen verknüpft, im Falle von langen Buffering-Zeiten innerhalb von Bridges können sich jedoch auch bei kurzen Pfaden unerwartet hohe Latenzen ergeben.

Lastverteilung (lb*):* In einigen Arbeiten werden Schedules dahingehend optimiert, keine besonders stark ausgelasteten Links zu erzeugen, wenn ein alternatives Routing eine bessere Lastverteilung im Netzwerk ermöglicht. Dies ist beispielsweise sinnvoll, um später dynamisch weitere Echtzeitstreams zu integrieren oder auch ausreichend Ressourcen für andere Verkehrsklassen zur Verfügung zu stellen.

GCL-Länge ($|\mathrm{gcl}|$*):* Wie aus Kapitel 2.2.3.2 bekannt, werden Zeitschlitzbeginn und -ende in TSN-Bridges durch GCL-Einträge abgebildet, welche nur in begrenzter Anzahl pro Bridgeport zur Verfügung stehen. Unter bestimmten Umständen können direkt aufeinanderfolgende Zeitschlitze ohne Gate-Event realisiert werden, sodass eine Optimierung der GCL-Länge möglich ist und eine Realisierung von mehr zeitgesteuerten Streams unter der gegebenen Begrenzung erlaubt.

Queueanzahl für zeitgesteuerten Verkehr ($|\mathrm{ttq}|$*):* Sowohl der zeitgesteuerte Verkehr innerhalb eines TSN macht von mehreren Queues Gebrauch (beispielsweise zwecks Frame-Reordering), als auch die darunterliegenden Verkehrsklassen (zur weiteren Traffic-Priorisierung). Da die Queueanzahl pro Port nach IEEE 802.1Q auf acht begrenzt ist, kann es sinnvoll sein, die Planung des zeitgesteuerten Verkehrs hinsichtlich einer möglichst geringen Queueanzahl zu optimieren.

Jitter (jit*):* Scheduler, die im Rahmen der periodischen Sendevorgänge zulassen, dass diese entweder nicht exakt äquidistant nach dem vorgegebenen Sendeintervall $f_k.ct$ erfolgen bzw. dass die Streamlatenz sich von Periode zu Periode unterscheidet, führen zu Jitter in der E2E-Latenz. Im Falle eines solchen Vorgehens kann es sinnvoll sein, erzeugte Schedules nach dem Ausmaß des Jitters zu beurteilen.

Makespan (span*):* Bekannte IE-Systeme wie Profinet organisieren die Kommunikation in dedizierten Phasen für zeitgesteuerten Verkehr und andere Verkehrsklassen [2]. Hierbei kann es erforderlich sein, den zeitgesteuerten Verkehr auf einen konkreten Abschnitt der Periode zu begrenzen. Eine Optimierung des Schedules hinsichtlich dieser sogenannten Makespan kann wiederum geringere Sendeintervalle für den zeitgesteuerten Verkehr ermöglichen oder aber zu einer verbesserten QoS für niedriger priorisierte Verkehrsklassen beitragen.

Interferenz für niedriger priorisierte Verkehrsklassen (intf*):* Falls einzelne Netzwerklinks aufgrund des Routings zeitgesteuerter Streams stark ausgelastet sind oder falls eine Vielzahl von Zeitschlitzen für den zeitgesteuerten Verkehr für einen Port direkt aufeinanderfolgend geplant wurde, so kann dies zu erhöhten Latenzen für niedriger priorisierten Verkehr führen. Dementsprechend können diese Nachteile bereits bei der Planung des zeitgesteuerten Verkehrs berücksichtigt und erstellte Schedules dahingehend optimiert werden.

Porösität (por*):* Als eine konkrete Metrik zur Erfassung der Interferenz führen die Autoren von [21] die Porösität ein. Diese beschreibt, ob zwischen Zeitschlitzen ausreichend „Lücken" zur Übertragung anderer Verkehrsklassen vorhanden sind, sodass diese keine exzessiven Bufferingzeiten in Bridges erfahren, da ein längerer Block zeitgesteuerten Verkehrs übertragen wird.

Im Gegensatz zu den BPIs sind die QIs teilweise technologieabhängig (z.B. Makespan span für Profinet relevant, GCL-Länge $|\mathrm{gcl}|$ für TSN). Die Qualitätsindikatoren werden hier bewusst nicht präziser definiert, da sich die exakte Definition oftmals von Publikation zu Publikation unterscheidet.

Eine Übersicht über die in der Literatur genutzten Metriken ist in Tabelle 4.1 gegeben. In den meisten Fällen werden die BPIs und QIs bezüglich eines Parametersweeps von einem der Szenarioparameter bewertet. In der Tabelle zeigt $\mathrm{rt}/_{|F|,|\mathcal{V}^{es}|}$ dementsprechend an, dass rt sowohl über einen Parametersweep der Verkehrsmustergröße $|F|$ als auch über einen Parametersweep der Endgeräteanzahl $|\mathcal{V}^{es}|$ gemessen wurde. Einzeln aufgelistete Performanceindikatoren (z.B. lt für [22]) wurden dagegen anhand einiger konkreter Testszenarien (d.h. ohne Parametersweep) bewertet. Szenarien werden überwiegend zufällig generiert, wobei im Zuge von Parametersweeps in vielen Fällen mehrere Szenarien mit identischen Parametern erzeugt werden, um einen einzigen Datenpunkt zu untersuchen. Hiermit soll dem Einfluss der Zufallsgenerierung begegnet werden. Um Zufallsentscheidungen der Scheduler zu berücksichtigen, werden teilweise außerdem Rechenwiederholungen auf demselben Szenario ausgeführt. In der Tabellenspalte *Sce./Rep.* kennzeichnet die Angabe $20/1$ daher beispielsweise 20 Zufallsszenarien pro Datenpunkt bei 1 Rechenwiederholung pro Szenario. Im Falle einzelner Testszenarien (CS, von *Case Studies*) ohne Parametersweep ist dagegen die Anzahl untersuchter Szenarien angegeben.

Werden im Zuge der Tests mehrere Szenarien mit gleicher Parameterkombination untersucht oder auch Rechenwiederholungen auf einem Szenario durchgeführt, so erfolgt die Datenaggregation in der Literatur in der Regel durch Mittelwertbildung. In einigen Fällen werden auch weitere statische Eigenschaften wie *Standardabweichung* (STD), *Median* (MED), *Konfidenzintervall* (CI) oder Verteilungen in Form von *Boxplots* (BOX) angegeben, was ebenfalls in der Tabelle erfasst wird. Mit <>$^?$ markierte Werte sind unsicher, da keine explizite Angabe in der Publikation erfolgt und daher Werte aus dem Kontext abgeleitet werden mussten. Für viele der Arbeiten wurden die Angaben außerdem auf mehrere Einträge aufgespalten, da die Autoren verschiedene Messreihen mit unterschiedlichen Eingangsdaten durchführen, welche im folgenden Unterkapitel betrachtet werden. Die genutzten Eingangsdaten für einen Eintrag in Tabelle 4.1 können anhand der ersten Spalte (z.B. [35]-3) in Tabelle 4.3/4.4 nachgeschlagen werden. Abschließend sei angemerkt, dass in Tabelle 4.1 Abkürzungen für *window count* (wc), *iteration size* (is) und *slot count* (sc) verwendet werden, für deren genaue Bedeutung hier nur auf die ursprünglichen Publikationen verwiesen werden soll.

	BPIs	*QIs*	*Sce./Rep.*	*Statistics*
[22]	–	lt	2CS	–
[23]	$\mathrm{rt}/_{\lvert F\rvert}$ $\mathrm{sab}^{\mathrm{st}}/_{f_k.ct}$	lt	$1/1^{?}$	–
[24]	$\mathrm{rt}/_{\lvert F\rvert}$	$\mathrm{lt}/_{\lvert F\rvert}$	20/1	–
[25]	$\mathrm{rt}/_{\lvert F\rvert}$	–	20/1	–
[26]-1	$\mathrm{sab}/_{\lvert F\rvert}$ $\mathrm{rt}/_{\lvert F\rvert}$	$\lvert\mathrm{gcl}\rvert/_{\lvert F\rvert}$	10000/1	BOX
[26]-2	$\mathrm{sab}/_{\lvert F\rvert,\lvert\mathcal{V}^{es}\rvert}$ $\mathrm{rt}/_{\lvert F\rvert,\lvert\mathcal{V}^{es}\rvert}$	$\lvert\mathrm{gcl}\rvert/_{\lvert F\rvert}$ $\lvert\mathrm{gcl}\rvert/_{\lvert\mathcal{V}^{es}\rvert}$	10000/1	–
[27]-1	$\mathrm{rt}/_{\lvert F\rvert}$	–	$1/1^{?}$	–
[27]-2	rt	$\lvert\mathrm{ttq}\rvert$	20CS	–
[28]	$\mathrm{sab}^{\mathrm{st}}, \mathrm{rt}/_{\lvert F\rvert}$	$\lvert\mathrm{gb}\rvert/_{\lvert F\rvert}$	$1/1^{?}$	–
[29, 30]	$\mathrm{rt}/_{\lvert F\rvert}$, $\mathrm{sab}^{\mathrm{st}}$	–	$1/1^{?}$ 10000/1	–
[31]	$\mathrm{rt}/_{\lvert F\rvert}$	$\mathrm{jit}/_{\lvert F\rvert}$	$1/1^{?}$	–
[32]	$\mathrm{rt}, \mathrm{sab}^{\mathrm{st}}$	lb, lt	7CS	–
[33]-1	$\mathrm{rt}/_{\lvert F\rvert, wc}$	–	$1/1^{?}$	–
[33]-2	$\mathrm{rt}/_{wc}$	$\mathrm{jit}/_{wc}$	$1/1^{?}$	–
[33]-3	$\mathrm{rt}/_{\lvert F\rvert}$	–	$1/1^{?}$	–
[8]	$\mathrm{sab}^{\mathrm{st}}$	lt (only RC)	32CS	–
[34]	rt	$\lvert\mathrm{ttq}\rvert$	8CS	–
[35]-1	$\mathrm{sab}/_{is}$	–	$20/1^{?}$	–
[35]-2	$\mathrm{rt}, \mathrm{sab}/_{\lvert F\rvert}$	–	$20/1^{?}$	–
[35]-3	$\mathrm{rt}, \mathrm{sab}/_{\lvert\mathcal{V}^{br}\rvert}$	–	$20/1^{?}$	–
[9, 36]-1	$\mathrm{sab}^{\mathrm{st}}$	–	160CS	–
[9]-2	$\mathrm{rt}/_{\lvert F\rvert}$	–	$1/1^{?}$	–
[9, 36]-3	$\mathrm{rt}/_{\lvert\mathcal{E}\rvert, sc}$	–	$1/1^{?}$	–
[36]	$\mathrm{rt}/_{f_k.ct}$	–	1CS	STD

	BPIs	*QIs*	*Sce./Rep.*	*Statistics*
[37]-1	–	span	30CS	–
[37]-2	$\mathrm{rt}/_{\lvert F\rvert}$	–	$1/1^{?}$	–
[38]-1	$\mathrm{rt}, \mathrm{sab}/_{\lvert\mathcal{V}\rvert}$	–	10/1–20/1	BOX STD
[38]-2	$\mathrm{rt}, \mathrm{sab}/_{\lvert F\rvert}$	–	10/1–20/1	BOX STD
[38]-3	$\mathrm{rt}/_{\lvert\mathcal{V}\rvert,\lvert F\rvert}$	–	$10/1–20/1^{?}$	STD
[39]	$\mathrm{rt}, \mathrm{sab}/_{\lvert F\rvert}$ $\mathrm{rt}, \mathrm{sab}/_{\lvert paths\rvert}$	custom	8/1, 10/1	$\mathrm{CI}^{?}$
[40]-1	$\mathrm{rt}/_{\lvert F\rvert}$	–	20/1	STD
[40]-2	$\mathrm{rt}/_{cvertices}$	–	40/1	–
[40]-3	$\mathrm{rt}/_{\lvert F\rvert = \lvert\mathcal{V}\rvert}$	–	80/1	BOX
[10]-1	$\mathrm{rt}/_{\lvert F\rvert,\lvert\mathcal{V}\rvert, gr}$	–	$1/100^{?}$	STD
[10]-2	–	$\lvert\mathrm{gb}\rvert$ intf	1CS	–
[13]-1	rt, sab	–	200CS w/ 5 rep.	MED, CI
[13]-2	$\mathrm{rt}, \mathrm{sab}/_{\lvert F\rvert}$	–	5/5	MED, CI
[41]-1	$\mathrm{sab}^{\mathrm{st}}, \mathrm{rt}/_{\lvert F\rvert}$	–	5/1	–
[41]-2	$\mathrm{sab}^{\mathrm{st}}, \mathrm{rt}/_{\lvert\mathcal{E}\rvert}$	–	$5/1^{?}$	–
[41]-3	$\mathrm{sab}^{\mathrm{st}}, \mathrm{rt}/_{\lvert F\rvert}$	–	$5/1^{?}$	–
[42]-1	$\mathrm{rt}/_{\lvert F\rvert}$	–	$100/1^{?}$	–
[42]-2	–	$\mathrm{span}/_{\lvert F\rvert}$	$100/1^{?}$	–
[42]-3	$\mathrm{rt}/_{\lvert F\rvert}$	$\mathrm{span}/_{\lvert F\rvert}$	$100/1^{?}$	–
[42]-4	$\mathrm{sab}/_{\lvert F\rvert}$	–	$100/1^{?}$	–
[21]	$\mathrm{rt}/_{\lvert F\rvert}$	$\mathrm{por}/_{\mathrm{rt}}$	10/10	$\mathrm{CI}^{?}$
[43]	$\mathrm{rt}/_{\lvert F\rvert}$	lb	$1/1^{?}$	–
[14]	$\mathrm{sab}, \mathrm{rt}/_{\lvert F\rvert}$	$\mathrm{lb}/_{\lvert F\rvert}$	?	–

Tabelle 4.1: Performanceindikatoren und Messmethodik in der Literatur.

- Sofern Streamparameter ($f_k.ct$, $f_k.fs$) bei der Erzeugung aus vorab definierten, diskreten Mengen zufällig gezogen wurden, so erfolgt die Darstellung hier ebenfalls als Menge.
- Genutzte Schrittweiten für Mengen sind angegeben, außer wenn diese nicht spezifiziert wurden oder nicht gleichbleibend für eine Menge waren.
- Wertebereiche der Form <>–<> stehen in der Regel für Mengen mit Schrittweite 1.
- Die Zufallsziehung von Streamparametern folgt in der Regel gleichverteilten Wahrscheinlichkeiten, wobei dies in vielen Veröffentlichungen nicht explizit angegeben und entsprechend unsicher ist.
- Kennzeichnung von Streamparametern mit <>^{U} bedeutet, dass bei der Erstellung des Verkehrsmusters ein einziger Wert für alle Streams gezogen wurde (*uniforme* Streamparameter), während <>^{H} die Nutzung heterogener Streamparameter anzeigt, also die Ziehung eines Zufallswertes für jeden einzelnen Stream.
- Angabe mehrerer heterogener Wertebereiche oder Mengen (z.B. für $f_k.ct$ in [24]), bedeutet, dass diese verschiedenen Wertebereiche im Rahmen unterschiedlicher Messreihen zum Einsatz kamen.
- <>^{GU} zeigt an, dass die Publikation den Scheduler unter Nutzung generischer Zeiteinheiten testet, ohne diese in reale Zeiteinheiten oder Bitraten zu übertragen.
- Nicht genannte bzw. nicht nachvollziehbare Werte sind mit ? vermerkt, während aus dem Kontext abgeleitete, unsichere Werte weiterhin mit <>$^{?}$ gekennzeichnet sind.

Tabelle 4.2: Anmerkungen zur Interpretation von Tabelle 4.3/4.4.

4.1.2 Eingangsdaten der Scheduler

Wie bereits im Kontext der Performanceindikatoren genannt, werden Scheduler im Zuge von Parametersweeps über eine Vielzahl zufallsgenerierter Eingangsdaten (d.h. Testszenarien) hinweg bewertet. In den Tabellen 4.3 und 4.4 werden daher die Eckdaten der in der Literatur genutzten Testszenarien zusammengefasst. Hierfür wurde angestrebt, die unterschiedlichen Notationen aus der Literatur bestmöglich in die eingeführten Parameter aus Kapitel 3 zu überführen. Zur Interpretation der Angaben sind außerdem die Anmerkungen aus Tabelle 4.2 zu beachten. Des Weiteren sei angemerkt, dass die konkreten Parameterkombinationen (aus Topologie, $|F|$, $f_k.ct$, usw.) aufgrund des Umfangs nicht einzeln abgebildet werden können. Wurden im Rahmen einer Publikation jedoch Tests mit stark unterschiedlichen Parameterkombinationen durchgeführt, so wurden diese zur besseren Übersicht auf mehrere Tabelleneinträge aufgeteilt (z.B. für [27]).

Besondere Aufmerksamkeit erfordern außerdem die Größe $|F|$ des Verkehrsmusters sowie die Latenzanforderung $f_k.ml$. Während ein Stream in den meisten Arbeiten nur genau einen Frame pro Zykluszeit sendet, besteht ein Stream in einigen Arbeiten (z.B. [27]) aus mehreren Frames, was in der Tabellenspalte zur Verkehrsmustergröße $|F|$ ggf. unter Angabe der *Frames per Stream* (FPS) vermerkt ist. Der Planungsaufwand eines Verkehrsmusters kann in diesen Fällen entsprechend höher ausfallen, als es die Angabe der Größe $|F|$ des Verkehrsmusters vermuten lässt. Bezüglich der Latenzanforderungen von Streams ist zu beachten, dass einige Arbeiten die Latenzanforderung von Streams auf den Beginn eines Sendeintervalls beziehen, sodass die Deadline für die Übertragung zum Empfänger unabhängig vom Sendezeitpunkt am Sender ist. Bei späterem Absenden des Frames steht somit weniger Zeit für die Übertragung durch das Netzwerk zur Verfügung. Latenzanforderungen

	Network Structure *Type:* $\lvert\mathcal{V}^{br}\rvert/\lvert\mathcal{V}^{es}\rvert$	*Stream Set* *Size* $\lvert F\rvert$	*Cycle Time* $f_k.ct$ [µs]	*Frame Size* $f_k.fs$ [B]	*Max. Latency* $f_k.ml$ [µs]
[22]	mesh:6/37 line:4/6	85 55	$\{166,\dots,100000\}^{H}$ $\{304,609,1000\}^{H}$	$\{64,\dots,1500\}^{H}$ $\{80,719,1480\}^{H}$	$\{166,\dots,16667\}^{R}$ $\{300,600,1000\}^{R}$
[23]	line:$7^{?}$/?	$\{4,\dots,30000\}$ (step ?)	$\{100,\dots,2000\}^{U}$ step 100	$\{64,256,1518\}^{U}$	$= f_k.ct^{R}$
[24]	line:7/? ring:6/? tree:8/?	$\{200,\dots,1200\}$ (step 200)	$1000-10\,000^{H}$ $10\,000-20\,000^{H}$ $1000-20\,000^{H}$	$64-1500^{H}$	$PL-$ $PL+f_k.ct^{A}$
[25]	line:? ring:? tree:?	$\{5,\dots,50,\dots,400\}$ (step 5 then 50)	$10-10\,000^{H}$ $10-1000^{H}$	$64-1500^{H}$	$PL(CQF)-$ $PL(CQF)+f_k.ct^{A}$
[26]-1	randomized pm:$6/6^{?}$	$\{10,\dots,50\}$ (step 10)	$32-64^{HGU}$	?	$\leq f_k.ct^{A}$
[26]-2	randomized pm: $\{8/8^{?},\dots,20/20^{?}\}$ (step 3)	$\{1000,\dots,10000\}$ (step 1000/2000)	$4096-32\,768^{HGU}$ $8192-16\,384^{HGU}$	?	$\leq f_k.ct^{A}$
[27]-1	?:$\{1/3,2/5,5/7\}$	$\{5,15,25\}$ $\{5,25,50\}$ $\{10,50,100\}$ (2–8 FPS)	$\{10000,20000\}^{H}$ $\{10000,25000,50000,100000\}^{H}$ $\{5000,10000,200000,500000\}^{H}$	1542^{U}	?
[27]-2	?:$\{1/3,2/5,5/7\}$	$\{3,4,5\}$ ($\{1,\dots,100\}$ FPS)	$\{1000,2000,3000,5000,10000,$ $100,150,200,500\}^{H}$	?	?
[28]	random: $\{20/?,10/30,$ $25/75,50/150\}$	$\{100,\dots,1500\}$ (step 100)	$?^{H}$	?	?
[29, 30]	mesh:9/–	$\{100,\dots,500\}$ (step 100, high FPS)	$\{5000,10000,25000,50000\}^{H}$	$?^{H}$	?
[31]	?:10/50	$\{10,\dots,50\}$ (step 10)	5000^{U}	1542^{U}	?
[32]	reuses parameters from [44]				$10\,f_k.ct^{R?}$
[33]-1	line:5/15 line:10/50	$\{10,\dots,50\}$ (step 10)	$\{10000,20000\}^{H}$	1542^{U}	?
[33]-2	line:5/15	25	20000^{U}	1542^{U}	?
[33]-3	line:5/15	$\{25,\dots,750\}$	20000^{U}	1542^{U}	?
[8]	?:$\{3/5,\dots,8/37\}$	$\{43,\dots,220\}$	many sets w/ range $1000-375000^{H}$	many sets w/ range $1-1471^{H}$ (payload)	?
[34]	reuses parameters from [27]				–
[35]-1	?:8/8	$\{240,480\}$	$5000-100\,000^{H}$	$50-1500^{H}$	?
[35]-2	?:4/10	$\{20,30,40,60\}$	$5000-100\,000^{H}$	$50-1500^{H}$	?
[35]-3	?:$\{3,9,15,21\}$/?	?	$5000-100\,000^{H}$	$50-1500^{H}$	?
[9, 36]-1	random:6/24	$20-110$	$1000^{U?}$	?	?
[9]-2	random:6/24	$\{20,\dots,110\}$ (step 10)	$1000^{U?}$	?	?
[9, 36]-3	random: $\lvert\mathcal{E}\rvert \in \{30,\dots,256\}$	up to 300	$1000^{U?}$	?	?
[36]	random: $\lvert\mathcal{E}\rvert := 150$	100	$\{1000,\dots,6000\}^{H}$ (step 1000)	?	?

Tabelle 4.3: Szenarioparameter in der Literatur (Teil 1/2).

	Network Structure *Type:* $\lvert\mathcal{V}^{br}\rvert/\lvert\mathcal{V}^{es}\rvert$	*Stream Set* *Size* $\lvert F\rvert$	*Cycle Time* $f_k.ct$ [µs]	*Frame Size* $f_k.fs$ [B]	*Max. Latency* $f_k.ml$ [µs]
[37]-1	random:$\{5/24,\ldots,20/100\}$	$\{30,\ldots,1500\}$	$?^{U}$	?	?
[37]-2	?	$\{10,\ldots,1500\}$	$?^{U}$	?	?
[38]-1	line,ring, scalefree,random: $\lvert\mathcal{V}\rvert \in \{n \in \mathbb{N}^0 \mid 5 \leq n \leq 36\}$	7	$\{1000,2000,4000,8000\}^{H}$	$\{375,750,1125\}^{H}$	based on $\lvert\mathcal{V}\rvert^{R}$
[38]-2	line,ring, scalefree,random: $\lvert\mathcal{V}\rvert := 8$	2−30	$\{1000,2000,4000,8000\}^{H}$	$\{375,750,1125\}^{H}$	based on $\lvert\mathcal{V}\rvert^{R}$
[38]-3	line,ring, scalefree,random: $\lvert\mathcal{V}\rvert \in \{n \in \mathbb{N}^0 \mid 5 \leq n \leq 20\}$	2−15	$\{1000,2000,4000,8000\}^{H}$ $\{50,100,200,400,800\}^{H}$	$\{375,750,1125\}^{H}$	based on $\lvert\mathcal{V}\rvert^{R}$
[39]	mesh: $\lvert\mathcal{V}\rvert := 16$	$\{10,\ldots,18\}$ (step 2)	$?^{HGU}$	$?^{UGU}$	based on $\lvert\mathcal{V}\rvert^{R}$
[40]-1	extended ring: $\lvert\mathcal{V}\rvert := 50$	$\{50,\ldots,150\}$ (step 10)	300^{U}	$5\,\mu s^{U}$	?
[40]-2	extended ring: $\lvert\mathcal{V}\rvert := 50$	$\{25,\ldots,200\}$ (step 25)	1000^{U}	$5\,\mu s^{U}$	?
[40]-3	extended ring: $\lvert\mathcal{V}\rvert \in \{50,100,200,300,400\}$	$\{50,100,200,300,400\}$	1000^{U}	$5\,\mu s^{U}$	?
[10]-1	? (max 4 hops)	$\{5,\ldots,75\}$ (step 5)	10000^{U}	1500^{U}	?
[10]-2	?:4/13	27	$\{500,100,800,30,1000\}\cdot 10^{3H}$	?	?
[13]-1	hier. ring: $\lvert\mathcal{E}\rvert \in \{200,\ldots,800\}$	random from: 50−200	1000^{U}	$64-1500^{H}$	$= f_k.ct^{A}$
[13]-2	hier. ring:?	$\{250,500\}$	1000^{U}	$64-1500^{H}$	$= f_k.ct^{A}$
[41]-1	Orion CEV:13/31	$\{500,1000,1500,2000\}$	$\{1000,2000,4000,8000\}^{H}$	?	?
[41]-2	random: $\lvert\mathcal{E}\rvert \in \{15,\ldots,30\}$ (step 3)	1500	$\{1000,2000,4000,8000\}^{H}$	?	?
[41]-3	ring: $\lvert\mathcal{V}\rvert := 6$	$\{900,1200,1500,1800\}$	$\{1000,2000,4000,8000\}^{H}$ $\{4000,8000\}^{H}$	?	?
[42]-1	?:16/39	10−20	$\{1000,2000,4000,8000\}^{H}$ $\{1000,2000\}^{H}$ $\{1000,4000\}^{H}$ $\{4000,8000\}^{H}$	$\{1,\ldots,12\}\,\mu s^{H}$ $\{1,2,3\}\,\mu s^{H}$ $\{5,6,7\}\,\mu s^{H}$ $\{10,11,12\}\,\mu s^{H}$	$500\,\mu s - f_k.ct^{R}$
[42]-2	?:16/39	$\{20,\ldots,40\}$ (step 2)	$\{1000,2000,4000,8000\}^{H}$	$\{1,\ldots,12\}\,\mu s^{H}$	$500\,\mu s - f_k.ct^{R}$
[42]-3	?:16/39	$\{10,\ldots,40\}$ (step 2)	$\{1000,2000,4000,8000\}^{H}$	$\{1,\ldots,12\}\,\mu s^{H}$	$500\,\mu s - f_k.ct^{R}$
[42]-4	?:16/39	$\{10,\ldots,160\}$ (step 10)	$\{1000,2000,4000,8000\}^{H}$	$\{1,\ldots,12\}\,\mu s^{H}$	$500\,\mu s - f_k.ct^{R}$
[21]	mesh:3/6,8/8	$\{10,20,30,40,50\}$	$\{10000,20000,40000\}^{H}$	1500^{U}	$= f_k.ct^{A}$ $= 0.25 f_k.ct^{R}$
[43]	?:$\lvert\mathcal{E}\rvert := 24$	$\{20,30,40,60,90\}$	$\{5000,10000,25000,50000\}^{H}$	$50000^{U?}$	?
[14]	?:24/72	$\{50,\ldots,500\}$ (step 50) $\{10,\ldots,100\}$ (step 10)	$\{2,4,8,16,32\}\cdot 10^{3H}$	$64-1518^{H}$	8000^{R}

Tabelle 4.4: Szenarioparameter in der Literatur (Teil 2/2).

dieser Form werden hier als *absolute* Latenzschranken bezeichnet und mit <>^{A} gekennzeichnet. Relative Latenzschranken beziehen sich dagegen (entsprechend der Definition nach Kapitel 3.2) auf den Sendebeginn am Sender und werden hier mit <>^{R} gekennzeichnet. Außerdem orientieren sich die für einen Stream im Rahmen der Zufallsgenerierung gewählten Latenzschranken in einigen Fällen an der bestmöglichen E2E-Latenz von Sender zu Empfänger, wobei hierfür in der Regel unveränderliche Größen wie Bridge- und Leitungsverzögerungen berücksichtigt werden, nicht jedoch Queueing-Verzögerungen aufgrund von Interferenz mit anderen Streams. Diese bestmöglichen Latenzen werden in Tabelle 4.3/4.4 mit PL abgekürzt. Des Weiteren sei angemerkt, dass viele der Arbeiten heterogene Sendeintervalle $f_k.ct$ für die Streams eines Verkehrsmusters nutzen. Während dies bei sinnvollen Vorgaben bezüglich der möglichen Sendeintervalle (z.B. als Menge, wie in [27]) ein plausibles Vorgehen ist, werden in einigen Arbeiten Wertebereiche (z.B. [24]) angegeben. Da jedoch bei beliebiger Zufallsziehung aus einem Wertebereich in der Regel keine kompatiblen Sendeintervalle entstehen, bleibt in diesen Fällen unklar, wie überhaupt planbare Szenarien erzeugt wurden.

4.1.3 Ziele eines systematischen Benchmarkings

Die vielen Veröffentlichungen haben gemeinsam, dass der Fokus auf der Entwicklung neuer Planungsverfahren für die verschiedenen Verkehrsklassen von TSN und vor allem den zeitgesteuerten Übertragungen liegt, während die *Evaluation* der Planungsverfahren nur als Notwendigkeit betrachtet wird, um die grundsätzliche Funktionsfähigkeit der entwickelten Methoden nachzuweisen. Das Bewertungsverfahren für Scheduler wird also nicht als Kernthema der Arbeit aufgefasst und die Auswahl von Testszenarien und Performanceindikatoren daher auch nicht umfassend diskutiert. Daraus resultieren in der Mehrheit der Arbeiten Unzulänglichkeiten in der Evaluation der Scheduler, die zu einer schwachen Aussagekraft der Ergebnisse führen. Entsprechend der eigenen Arbeit [E15] sollen daher nachfolgend die Probleme der Testmethoden aus der Literatur identifiziert und die wesentlichen Anforderungen an ein *systematisches* Benchmarkingverfahren abgeleitet werden. Das systematische Benchmarking wird außerdem in der Arbeit von Xue [45] explizit behandelt. Ein Vergleich zu dieser Arbeit wird in Kapitel 4.4 vorgenommen.

Der erste problematische Aspekt ist die Zusammenstellung der Testszenarien bzw. der Parameterkombinationen bei der Durchführung von verschiedenen Messreihen. Da die verschiedenen Autoren sich bei der Parametrisierung der Szenarien am gleichen Anwendungsbereich (Echtzeitnetzwerke in der Fabrikautomation) orientieren, gibt es zwischen genutzten Szenarioparametern grundsätzlich Überschneidungen. Allerdings zeigen Tabelle 4.3 und 4.4 eindrücklich, dass es keinen Konsens bei der Abstufung genutzter Wertebereiche und bei der Kombination von Topologien und Streamparametern gibt. Parametersweeps bei der Bewertung der Performanceindikatoren (vgl. Tabelle 4.1) fokussieren sich außerdem überwiegend auf die Topologie- und Verkehrsmustergröße (d.h. $|\mathcal{V}^{es}|$ und $|F|$), während der Streamparameter $f_k.ct$ durch nur wenige separate Messreihen abgedeckt wird und der Einfluss der Streamparameter $f_k.fs$ und $f_k.ml$ kaum explizit berücksichtigt wird. Eine detaillierte Betrachtung der Erläuterungen zur Evaluationsmethodik in den Publikationen offenbart außerdem, dass die Parameterwahl abgesehen von der grundlegenden Dimensionierung anhand erwarteter, realer Anwendungen nur schwach begründet wird. So wird die Variation von

Streamparametern wie $f_k.ct$ beispielsweise in der Regel vorgenommen ohne eine konkrete Erwartung bezüglich der Schedulingergebnisse zu formulieren. Die Interaktion von Parametern wird außerdem kaum explizit berücksichtigt. Diesbezüglich ist besonders die Netzwerkauslastung hervorzuheben, welche sich erst aus der Kombination aller Parameter ergibt und beispielsweise nur in [14, 43, 22] durch Angabe der Linkauslastung für Bottleneck-Links charakterisiert wird. Alle anderen hier untersuchten Arbeiten machen diesbezüglich überhaupt keine Angabe. Dies ist vor allem deshalb höchst problematisch, weil somit in vielen Fällen offen bleibt, inwiefern die genutzten Testszenarien die Scheduler überhaupt einer stark ressourcenbeschränkten Planung aussetzen, oder ob es sich um reine Skalierbarkeitstests bei vernachlässigbarer Ressourcenbeschränkung handelt.

Testabdeckung/Testintention Die erste Zielsetzung eines systematischen Benchmarkings ist daher eine umfassende Testabdeckung. Es sollten dementsprechend Analysen bezüglich aller Szenarioparameter enthalten sein. Die Interaktion von Parametern sollte ebenfalls berücksichtigt werden. Konkret bedeutet das, dass Sweeps eines Parameters gegebenenfalls für unterschiedliche Kombinationen der anderen Parameter wiederholt werden sollten, sofern diese zu unterschiedlichen Effekten beim Scheduling führen. Insbesondere sollten Probleme der Ressourcenbeschränkung und Skalierbarkeit durch Angabe der Netzwerkauslastung von Testszenarien voneinander unterscheidbar sein. Für jede festgelegte Messreihe sollte darüber hinaus eine Testintention formuliert werden, die den Zweck der Messreihe und ggf. erwartbare Effekte erfasst.

Das zweite grundlegende Problem der Messreihen aus der Literatur ist die statistische Auswertung bzw. Berücksichtigung von Zufallsentscheidungen. Tabelle 4.1 zeigt, dass es weder bezüglich der Anzahl zufallsgenerierter Szenarien für einen Datenpunkt noch bezüglich der Rechenwiederholungen für ein Szenario ein einheitliches Vorgehen gibt. Während Rechenwiederholungen im Falle deterministisch arbeitender Scheduler berechtigterweise weggelassen werden können, werden in der Literatur in vielen Fällen scheinbar keine Rechenwiederholungen durchgeführt, obwohl offensichtlich Varianz innerhalb der Ergebnisse zu erwarten ist. Der Einfluss der zufälligen Erstellung von Szenarien wird darüber hinaus in keiner der betrachteten Arbeiten untersucht. Sofern Datenpunkte aggregierte Ergebnisse aus mehreren Szenarien und Rechenwiederholungen enthalten, werden in der Literatur außerdem nur von den Autoren Hellmanns [13] und Falk [38, 39, 40] statistische Eigenschaften für nahezu alle Ergebnisse angegeben. Da Ergebnisdaten ebenfalls nicht veröffentlicht werden und somit auch nicht nachträglich untersucht werden können, bleibt die tatsächliche Relevanz von Ergebnissen teilweise unklar.

Statistische Eigenschaften Im Rahmen eines systematischen Benchmarkings sollten statistische Eigenschaften von Zufallsprozessen bei Szenarioerstellung und Scheduling berücksichtigt und gegebenenfalls charakterisiert werden. Bei zufälliger Szenarioerstellung sollten somit stets mehrere Szenarien unter Nutzung gleicher Parameter erstellt werden, während Rechenwiederholungen für nicht deterministisch arbeitende Scheduler zum Einsatz kommen sollten. Ergebnisse sollten durch angemessene statistische Angaben repräsentiert werden und Ergebnisdatensätze ggf. zur weiteren Analyse zur Verfügung gestellt werden.

Ein weiteres zentrales Problem, das sich an den genutzten Szenarioparametern nach Tabelle 4.3/4.4 zeigt, ist die fehlende Vergleichbarkeit von Schedulerevaluationen aus der Literatur.

Aufgrund der unterschiedlichen Parameterwahl der Testszenarien sind Performancevergleiche über mehrere Publikationen hinweg nahezu unmöglich. Die nichtlinearen Einflüsse der Szenarioparameter auf die Schedulerperformance führen dazu, dass selbst kleine Abweichungen zwischen einzelnen Parametern keinen sinnvollen Vergleich zulassen. Eine sinnvolle Performanceabschätzung der zahlreichen veröffentlichten Scheduler der vergangenen Jahre ist somit bisher weitgehend unmöglich. Herbeigeführt wurde dieses Problem durch in der Regel unveröffentlichte Testszenarien, sodass Autoren gezwungenermaßen immer wieder neue Testszenarien erzeugt haben. Darüber hinaus werden die Parameter dieser neu zu erzeugenden Szenarien unter anderem an den Fähigkeiten der eigenen Scheduler ausgerichtet und ggf. auch durch spezielle Fragestellungen beeinflusst. So dienen die unterschiedlichen Größenordnungen der heterogenen Zykluszeit in [27]-2 beispielsweise dem Vergleich zweier Bedingungen zur Frameisolation, die von dieser speziellen Parameterwahl besonders beeinflusst werden. Weiter verstärkt wird die Problematik durch unzureichende Beschreibungen verwendeter Szenarien (siehe fehlende oder unsichere Angaben in Tabelle 4.1–4.4), sodass ein Nachbau von Testszenarien, sofern angestrebt, in vielen Fällen ebenfalls nicht realisierbar wäre.

Wiederverwendbarkeit Die Szenarien eines systematischen Benchmarks sollten daher zukünftig unter Berücksichtigung der vorigen Zielsetzung bzgl. Testabdeckung und statistischer Eigenschaften erzeugt und als Datensatz in Form eines wiederverwendbaren und dokumentierten Formats zur Verfügung gestellt werden. Begleitend sollte eine reproduzierbare Beschreibung der Szenarioerstellung erfolgen. Durch präzise Hardware- und Softwareangaben im Rahmen von Messreihen soll somit eine publikationsübergreifende Performanceabschätzung bzw. bei gleicher Hard- und Software direkte Vergleiche ermöglicht werden.

Es sei angemerkt, dass durch die Festlegung umfassender publikationsübergreifender Benchmarkingszenarien nicht die Relevanz spezialisierter, zusätzlicher Analysen bestimmter Schedulereigenschaften in Frage gestellt werden soll.

4.2 Systematisches Benchmarking

Das hier vorgeschlagene *systematische Benchmarking* ist unter Berücksichtigung der in Kapitel 4.1.3 formulierten Ziele entworfen worden. Somit sollen die Probleme bisheriger Bewertungsverfahren vermieden und ein detaillierter Vergleich von Schedulern ermöglicht werden. Um die hierfür notwendige Testabdeckung und statistische Signifikanz zu erzielen, ist das Testen mit einer Vielzahl von Szenarien unerlässlich. Dementsprechend wurde für das systematische Benchmarking zunächst ein *Szenariogenerator* für die zufällige Erzeugung von Szenarien entworfen. Dieser arbeitet mit beliebigen Netzwerktopologien und erzeugt zufällige Verkehrsmuster nach Angabe bestimmter Eckdaten. Diese Erzeugung zufälliger Szenarien wurde dann zur Komposition größerer *Testcases* (TCs) eingesetzt. Jeder Testcase beinhaltet Szenarien, die sich mit einer bestimmten Fragestellung bezüglich der Schedulerperformance befassen, wie z.B. dem Sweep eines Szenarioparameters. Die Strukturierung der Szenarien in Testcases dient der besseren Nachvollziehbarkeit von Testergebnissen. Insbesondere die unterschiedliche Reaktion verschiedener Scheduler auf die Veränderung bestimmter Szenarioparameter soll erkennbar werden, sodass sich konkrete Stärken und Schwächen einzelner

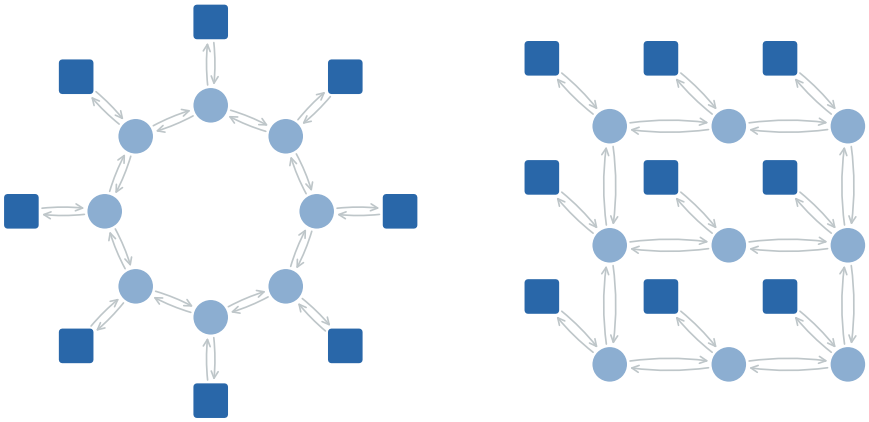


Abbildung 4.1: Ringtopologie mit $|\mathcal{V}^{es}| := 8$ und Meshtopologie mit $|\mathcal{V}^{es}| := 9$.

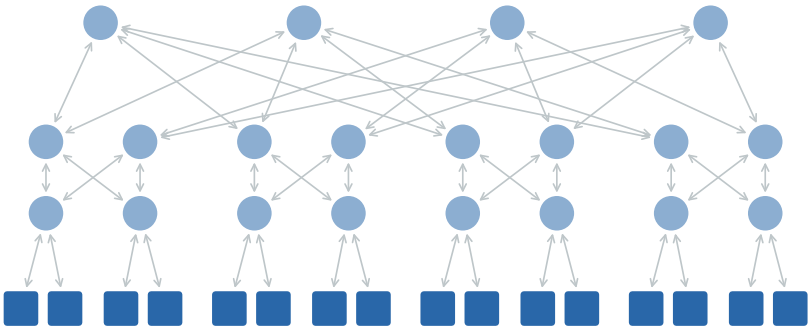


Abbildung 4.2: Fat-Tree-Topologie mit $|\mathcal{V}^{es}| := 16$.

Scheduler identifizieren lassen. Diese Erkenntnisse lassen sich nicht nur in der Weiterentwicklung von Schedulern nutzen, sondern können auch in der Praxis hilfreich sein, um die bestmöglichen Einsatzbereiche verschiedener Scheduler einzuschätzen.

4.2.1 Erzeugung synthetischer Szenarien (Szenariogenerator)

Die Zielsetzung des Szenariogenerators war es, durch entsprechende Parameter ausreichend Kontrolle über die verwendeten Topologien und erzeugten Verkehrsmuster zu bieten, um passende Szenarien für umfassende Testcases zu generieren. Die zu verwendende Topologie wird bei Erstellung eines Szenarios durch die Angabe der Netzwerkstruktur ns (Linie, Ring, usw.) und der Anzahl der verbundenen Endgeräte $|\mathcal{V}^{es}|$ festgelegt. Eine entsprechende Topologie wird dann mittels einer Helferfunktion für die konkrete Netzwerkstruktur erstellt. Im ersten im Rahmen der Arbeit erstellten Benchmarkingdatensatz kommen als Netzwerkstrukturen ausschließlich $ring$ und $mesh$ zum Einsatz, welche in Abbildung 4.1 in der kleinsten genutzten Größe gezeigt sind. Die grundlegende Struktur wird dabei durch Bridges gebildet, wobei mit jeder Bridge genau ein Endgerät verbunden ist. In einem zweiten, umfangreicheren Benchmarkingdatensatz wird außerdem der in Abbildung 4.2 gezeigte $fattree$ genutzt, dessen Nutzung in Ethernetnetzwerken durch [46] Bekanntheit erlangt hat. Abbildungen aller weiteren genutzten Topologiegrößen finden sich online in [E18]. Bridge- und

(a)

Class	*probability*	$f_k.ct$	$f_k.fs$
TT1	0.3	$1ss^{ct}$	$\max(0.67ss^{fs}, 100\mathrm{B})$
TT2	0.3	$2ss^{ct}$	$\max(0.67ss^{fs}, 100\mathrm{B})$
TT3	0.4	$4ss^{ct}$	$\max(1.00ss^{fs}, 100\mathrm{B})$

(b)

$\lvert f_k.dsts\rvert$	*probability*
1	0.50
2	0.35
3	0.10
4	0.05

Tabelle 4.5: Verkehrsklassen und Empfängeranzahl (bei Multicast-Verkehrsmustern).

Linkeigenschaften wurden statisch festgelegt, wobei alle hier genutzten Topologien aus CT-Bridges mit $4\,\mu\mathrm{s}$ Verzögerung sowie Netzwerklinks mit einer Bandbreite von $1\,\mathrm{Gbit/s}$ und einer idealisierten Signallaufzeit von Null bestehen.

Für die darauf folgende Generierung des Verkehrsmusters werden zunächst die gewünschten Eckdaten $|F|$, ss^{ct}, ss^{fs} und ss^{lf} festgelegt. Dabei gibt $|F|$ an, wie viele Streams für das Verkehrsmuster $(F, (f_k)_{k\in F})$ generiert werden sollen. Ein einzelner Stream k wird erstellt, indem Sender und Empfänger zufällig aus $\mathcal{V}^{es}$ gezogen werden, wobei die Anzahl der Empfänger $|f_k.dsts|$ zufällig anhand einer gegebenen diskreten Wahrscheinlichkeitsverteilung entschieden wird (siehe z.B. Tab. 4.5b). Die Streameigenschaften $f_k.ct$ und $f_k.fs$ werden durch die zufällige Zuweisung einer Verkehrsklasse festgelegt, wobei für die möglichen Verkehrsklassen ebenfalls eine diskrete Wahrscheinlichkeitsverteilung gegeben ist (siehe z.B. Tab. 4.5a). Jede Klasse definiert die Werte für $f_k.ct$ und $f_k.fs$ in Abhängigkeit von ss^{ct} und ss^{fs}. Die Verkehrsklassen sollen abbilden, dass die Streams verschiedener Anwendungen unterschiedliche Anforderungen bezüglich Zykluszeit und Framegröße besitzen. Für $ss^{ct} := 100\,\mu\mathrm{s}$ und $ss^{fs} := 1500\mathrm{B}$ bedeuten die gezeigten Daten beispielsweise, dass einem Stream k mit einer Wahrscheinlichkeit von 30 % die Klasse TT2 zugewiesen wird und dieser somit die Streameigenschaften $f_k.ct := 200\,\mu\mathrm{s}$ und $f_k.fs := 1000\mathrm{B}$ erhält. Abschließend wird dem Stream die maximal zulässige Latenz $f_k.ml := ss^{lf} \cdot \max_{j\in f_k.dsts}(lt_{kj}^{i,sp})$ zugewiesen. Hierbei gibt $lt_{kj}^{i,sp}$ die *ideale Latenz des kürzesten Pfades* zum Empfänger j an, welche sich unter Berücksichtigung der in der Topologie angegeben Bridge- und Leitungsverzögerungen, jedoch ohne Wartezeiten durch Konflikte mit anderen Streams (Queueing) ergibt.

Insgesamt bietet der Szenariogenerator somit das 6-Tupel $sg := (ns, |\mathcal{V}^{es}|, |F|, ss^{ct}, ss^{fs}, ss^{lf})$ als flexible Parameter bei der Szenarioerstellung an, wobei bezüglich ns hier nur $ring$, $mesh$ und $fattree$ für einige konkrete Größen $|\mathcal{V}^{es}|$ definiert wurden. Die gezeigten Wahrscheinlichkeitsverteilungen werden im Verlauf der Arbeit nicht variiert, obwohl weitere Untersuchungen hier durchaus interessant wären (beispielsweise eine größere Anzahl von Verkehrsklassen mit sich stärker unterscheidenden Zykluszeiten). Diese Variation der Wahrscheinlichkeitsverteilungen sowie weitere Netzwerkstrukturen werden im hier vorgestellten Benchmarking nicht genutzt, da zum einen aufgrund der langen Rechenzeiten der Scheduler und zum anderen aufgrund der manuellen Interpretation der Ergebnisse gewisse Einschränkungen des Testumfangs notwendig waren. Der wesentliche Unterschied des hier erläuterten Verfahrens gegenüber der Szenarioerstellung in [E15] besteht in der Einführung von Verkehrsklassen und Multicast. Alle in [E15] genutzten Szenarien wurden als Unicast mit einheitlichen Streamparametern innerhalb eines Verkehrsmusters erzeugt.

4.2.2 Komposition von Szenarien zu Testcases

Die zufällige Erzeugung von Szenarien wurde genutzt, um Testcases zusammenzustellen. Jeder Testcase (TC) dient dabei einer bestimmten Fragestellung bezüglich der Schedulerperformance bei Veränderung von bestimmten Szenarioparametern. Hier soll zunächst die Generierung und Auswertung eines Testcases anhand *TC-LL* (**L**ast- und **L**atenzvariation) erläutert werden, bevor alle für das systematische Benchmarking definierten Testcases präsentiert werden und abschließend eine Diskussion der Parameter erfolgt.

4.2.2.1 Erstellung eines Testcases

Bei TC-LL handelt es sich um einen Testcase, bei dem die Netzwerkauslastung der Szenarien mittels ss^{ct} (Sendeintervall erzeugter Streams) variiert wird und diese Parameterveränderung zudem für verschiedene ss^{lf} (Latenzschranke erzeugter Streams) wiederholt wird. Die Parameterliste für die Erzeugung der Szenarien von TC-LL kann durch folgende Tupel ausgedrückt werden:

$$\big((ring), (8), (82), (100\,\mu s, 124\,\mu s, 156\,\mu s, 196\,\mu s), (1500\,\mathrm{B}), (1.5, 3.0, 6.0)\big)$$
$$\big((mesh), (9), (79), (84\,\mu s, 100\,\mu s, 124\,\mu s, 156\,\mu s), (1500\,\mathrm{B}), (1.5, 3.0, 6.0)\big)$$

Die Parameter werden also in der Regel separat für jede Netzwerkstruktur definiert. Bei der Szenarioerstellung wird zu Beginn das kartesische Produkt für jedes dieser Tupel-von-Tupeln gebildet. Für $ring$ entstehen hier somit beispielsweise folgende Tupel:

$$\begin{gathered}(ring, 8, 82, 100\,\mu s, 1500\,\mathrm{B}, 1.5)\\ (ring, 8, 82, 100\,\mu s, 1500\,\mathrm{B}, 3.0)\\ (ring, 8, 82, 100\,\mu s, 1500\,\mathrm{B}, 6.0)\\ (ring, 8, 82, 124\,\mu s, 1500\,\mathrm{B}, 1.5)\\ \vdots\\ (ring, 8, 82, 196\,\mu s, 1500\,\mathrm{B}, 6.0)\end{gathered}$$

Jedes dieser Tupel dient dann wiederum direkt als Parametertupel $sg := (ns, |\mathcal{V}^{es}|, |F|, ss^{ct}, ss^{fs}, ss^{lf})$ für die Erzeugung entsprechender Szenarien mit Hilfe des Szenariogenerators. Konkret wurden bei der Erstellung der Benchmarkingszenarien für jedes sg immer genau vier Szenarien erzeugt, mit der Intention, mehrere Szenarien mit ähnlichem Schwierigkeitsgrad zu erhalten. Für jedes vom Szenariogenerator entworfene Szenario wurde außerdem geprüft, ob ein gültiges Routing (d.h. Auslastung aller Links unter 100 %) existiert, was im Vergleich zum Schedulingproblem TT-RSP schnell entscheidbar ist. War dies nicht der Fall, so wurde ein neues Szenario unter Nutzung der gleichen Parameter erzeugt. Dies wurde solange wiederholt, bis die gewünschten vier Szenarien vorlagen. Dieses Verwerfen offensichtlich unlösbarer Szenarien wurde eingesetzt, um bei Parameterkombinationen, die zu einer hohen Netzwerkauslastung führen, schwierige, aber potentiell noch lösbare Testszenarien zu erhalten. Ob tatsächlich eine Lösung unter Einhaltung aller Schedulingbedingungen existiert, kann jedoch nicht vorab beantwortet werden.

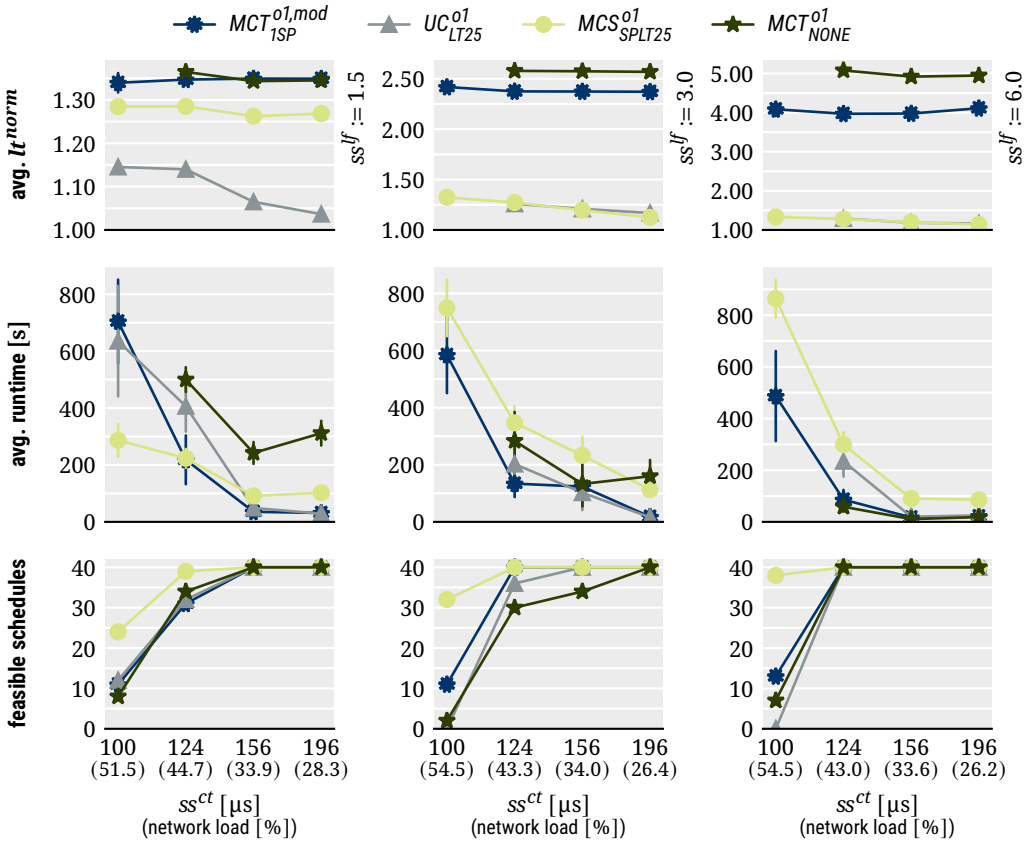


Abbildung 4.3: Ergebnisse für TC-LL auf Basis der Ringtopologie für vier Scheduler. Wiederholter Parametersweep von ss^{ct} für $ss^{lf} := 1.5$, $ss^{lf} := 3.0$ und $ss^{lf} := 6.0$ (v.l.n.r.). Konstante Szenarioparameter $ns := ring$, $|\mathcal{V}^{es}| := 8$, $|F| := 82$ und $ss^{fs} := 1500\text{B}$.

4.2.2.2 Auswertung eines Testcases

Beim Vergleich von Schedulern in dieser Arbeit musste jeder Scheduler jedes Szenario genau zehn Mal lösen. Dies ist nötig, da die Zufallsentscheidungen der Scheduler auch bei gleichen Eingabedaten zu unterschiedlichen Ergebnissen und Rechenzeiten führen. Abbildung 4.3 zeigt die Ergebnisse von TC-LL auf Basis der Ringtopologie und für vier Scheduler. Hierbei sind die Szenarioparameter ns, $|\mathcal{V}^{es}|$, $|F|$ und ss^{fs} konstant, während die Daten über einem veränderlichen Parameter des Testcases (hier ss^{ct}) aufgetragen werden und die gleiche Darstellung ggf. für weitere veränderliche Parameter (hier ss^{lf}) wiederholt wird. Jeder Eintrag auf der x-Achse entspricht somit einer konkreten Parameterkombination sg, für die genau 40 Ergebnisse (vier Szenarien, zehn Rechenwiederholungen) je Scheduler vorliegen. Die Fähigkeiten eines Schedulers werden in dieser Arbeit nach Schedulability, Rechenzeit und der Lösungsgüte in Form von Streamlatenzen beurteilt.

Schedulability Die Schedulability gibt an, wie viele der gegebenen Probleminstanzen gelöst werden konnten. Die in dieser Arbeit betrachteten Lösungsverfahren berücksichtigen keine Teillösungen. Eine Probleminstanz gilt somit nur als gelöst, wenn alle im Verkehrsmuster enthaltenen Streams konfliktfrei geplant wurden.

Rechenzeit Die gezeigten Rechenzeiten repräsentieren jeweils den Mittelwert der Rechenzeiten aller erfolgreich geplanten Probleminstanzen. In Abbildung 4.3 bedeutet dies für MCS^{o1}_{SPLT25} beispielsweise, dass für $ss^{lf} := 3.0$ (mittlere Spalte) und $ss^{ct} := 100\,\mu s$ der Mittelwert über 32 gelöste Probleminstanzen gezeigt ist. Zudem werden die Rechenzeiten erst ab 10 Ergebnissen dargestellt und Konfidenzintervalle (95 %) geben die Zuverlässigkeit der jeweiligen Datenpunkte an.

Normalisierte Latenz Die Güte eines Schedulingergebnisses wird charakterisiert, indem die E2E-Latenzen aller Streams und zu allen Empfängern aufsummiert und auf den bestmöglichen Wert dieser Latenzen normalisiert werden. Formal ist die normalisierte Latenz einer gelösten Probleminstanz durch

$$lt^{norm} := \frac{\sum_{k \in F} \sum_{j \in f_k.dsts} \boldsymbol{lt}_{kj}}{\sum_{k \in F} \sum_{j \in f_k.dsts} lt^{i,sp}_{kj}}$$

definiert, wobei $\boldsymbol{lt}_{kj}$ und $lt^{i,sp}_{kj}$ die Latenz von Stream k zu Empfänger j laut der Lösung bzw. als Idealwert bei Nutzung kürzester Pfade und ohne Ressourcenkonflikte angeben. Dargestellt wird die normalisierte Latenz entsprechend der Rechenzeit als Mittelwert für gelöste Probleminstanzen (ab 10 Ergebnissen) unter Angabe des Konfidenzintervalls (95 %).

Die Szenarien selbst werden hier neben den genutzten Parametern bei der Erstellung außerdem durch die verursachte Netzwerkauslastung charakterisiert. In Ergebnisplots wie Abbildung 4.3 wird dabei für einen Datenpunkt stets der Mittelwert der Netzwerkauslastung aller vier enthaltenen Szenarien angegeben. Die Netzwerkauslastung eines Szenarios wird dabei folgendermaßen ermittelt.

Netzwerkauslastung Zur Bestimmung der Netzwerkauslastung eines Szenarios wird davon ausgegangen, dass alle Streams zu all ihren Empfängern die kürzeste Route nutzen und ein Stream k auf einem genutzten Link m die Auslastung $sl_{km}/f_k.ct$ verursacht. Es sei angemerkt, dass Multicaststreams dabei zu Beginn der Route (vor dem Aufsplitten zu verschiedenen Empfängern) nur einfach gewichtet werden, da diese auch in der Praxis bis zum Punkt des Aufsplittens ohne duplizierte Frames übertragen werden. Die Netzwerkauslastung eines Szenarios ergibt sich dann als Mittelwert der Auslastung von allen Links der Topologie. Das bei dieser Berechnung angenommene Routing wird dabei nicht auf das Vorhandensein gültiger Schedules geprüft. Der Vorteil dieses Vorgehens liegt darin, dass die Netzwerkauslastung somit effizient und unabhängig vom Schedulingverfahren berechnet werden kann.

Bei den in Abbildung 4.3 gezeigten Schedulern handelt es sich um vier ILP-basierte Scheduler:

- MCT^{o1}_{NONE}: Multicastfähiges Modell ohne Optimierungsziel
- $MCT^{o1,mod}_{1SP}$: Alternatives multicastfähiges Modell mit Optimierungsziel bezüglich Pfadlänge, aber Abbruchbedingung bei erster gefundener Lösung
- MCS^{o1}_{SPLT25}: Weiteres multicastfähiges Modell mit zweistufiger Optimierung von Pfadlängen und Streamlatenzen
- UC^{o1}_{LT25}: Unicastfähiges Modell mit Optimierungsziel bezüglich Streamlatenzen

Eine genaue Beschreibung dieser Scheduler folgt in Kapitel 6, während sie hier sowie in den folgenden Abbildungen dieses Kapitels nur zur Verdeutlichung der Funktionsweise der Testcases dienen sollen. Dabei wurden zu Demonstrationszwecken bewusst Modelle gewählt, die sich aufgrund maßgeblicher Unterschiede bei den Randbedingungen und Optimierungszielen deutlich im Verhalten unterscheiden. Berechnet wurden die hier gezeigten Ergebnisse auf einem *Intel i7 9700* (deaktivierter Turbo Boost) mit *32GB DDR4-2666* unter Nutzung von *Ubuntu 18.04 LTS*, *Python 3.8.6* und *Gurobi 9.1.1* [72] (limitiert auf sechs Threads). Eine Diskussion der Testplattform und deren Konfiguration folgt in Kapitel 4.3.

Abschließend sei auf eine Besonderheit bei der Interpretation der Ergebnisse verwiesen. Da bei der Szenarioerstellung für jedes sg neue Zufallsszenarien erzeugt werden, kann der Schwierigkeitsgrad der Szenarien neben ggf. veränderten Parametern auch aufgrund der zufällig zugewiesenen Sender, Empfänger und Verkehrsklassen variieren. In Abbildung 4.3 kann somit beispielsweise für $ss^{ct} := 156\,\mu s$ nicht zweifelsfrei behauptet werden, dass Szenarien mit $ss^{lf} := 3.0$ (mittlere Spalte) grundsätzlich schwieriger planbar sind als mit $ss^{lf} := 6.0$ (rechte Spalte), obwohl die Rechenzeiten aller vier Scheduler für $ss^{lf} := 3.0$ höher ausfallen. Ob der Zufallseinfluss bei der Szenarioerstellung hier relevant ist, ließe sich beispielsweise mit einer Varianzanalyse bezüglich der jeweils vier mit gleichem sg erstellten Szenarien beantworten. Dies wird im Folgenden bewusst nicht näher analysiert. Einerseits dominiert in vielen Fällen offensichtlich die Auswirkung einer Parameterveränderung gegenüber dem Zufallseinfluss bei der Szenarioerstellung (siehe Parametersweep von ss^{ct} in Abb. 4.3). Andererseits liegt die Zielsetzung der hier vorgestellten Benchmarkingszenarien sowie der Analysen in dieser Arbeit beim Vergleich von Schedulern, wobei diese Problematik nicht relevant ist: Da alle Scheduler exakt dieselben Szenarien lösen müssen, ohne dass diese etwa für jeden Scheduler zufällig neu generiert werden, ist der Vergleich von Schedulern zuverlässig möglich. Aus Abbildung 4.3 lässt sich somit zweifelsfrei aussagen, dass MCT^{01}_{NONE} im Vergleich zu den anderen Schedulern schlechter auf eine Absenkung der Latenzschranken (ss^{lf}) reagiert.

4.2.3 Testcases des systematischen Benchmarkings

Entsprechend dem am Beispiel beschriebenen Schema wurden für das in [E15] entworfene und im Rahmen dieser Arbeit weiterentwickelte Benchmarking sechs Testcases definiert, wovon sich je drei davon mit Szenarien mit niedriger bzw. hoher Netzwerkauslastung befassen. Eine niedrige Netzwerkauslastung wurde hier vor allem dadurch erzielt, dass Szenarien mit hohem ss^{ct} und niedrigem ss^{fs} erzeugt wurden. Mit diesen Testcases soll vor allem die Skalierbarkeit (z.B. bezüglich der Größe von Topologie und Verkehrsmuster) der Scheduler untersucht werden, wobei großzügige Ressourcen zur Verfügung stehen und Schedulingkonflikte leicht zu lösen sein sollten. Die Testcases mit hoher Netzwerkauslastung wurden dagegen mit kleinem ss^{ct} und großem ss^{fs} erzeugt und sollen zeigen, inwiefern die getesteten Scheduler in der Lage sind, Schedulingkonflikte bei begrenzten Ressourcen aufzulösen. Insbesondere bezüglich dieser beiden Grundprobleme der Skalierbarkeit und der Schedulingkonflikte können sich die Fähigkeiten verschiedener Scheduler aufgrund der eingesetzten Methoden signifikant unterscheiden. Im Rahmen dieser Arbeit wurden alle sechs Testcases gegenüber [E15] bezüglich der Parameter überarbeitet und zudem mit dem überarbeiteten Szenariogenerator (d.h. Nutzung von mehreren Verkehrsklassen nach Tab. 4.5a) erstellt. Außerdem wurde

TC-L Load	$((ring), (8), (57, 82, 107), (100\,\mu s, 124\,\mu s, 156\,\mu s, 196\,\mu s), (1500\,B), (6)),$ $((mesh), (9), (55, 79, 103), (84\,\mu s, 100\,\mu s, 124\,\mu s, 156\,\mu s), (1500\,B), (6))$
TC-LL Load-Latency	$((ring), (8), (82), (100\,\mu s, 124\,\mu s, 156\,\mu s, 196\,\mu s), (1500\,B), (1.5, 3, 6)),$ $((mesh), (9), (79), (84\,\mu s, 100\,\mu s, 124\,\mu s, 156\,\mu s), (1500\,B), (1.5, 3, 6))$
TC-G Granularity	$((ring), (8), (45, 57, 70), (100\,\mu s), (1500\,B), (6)),$ $((mesh), (9), (43, 55, 67), (84\,\mu s), (1500\,B), (6)),$ $((ring), (8), (57, 72, 88), (100\,\mu s), (1200\,B), (6)),$ $((mesh), (9), (55, 70, 85), (84\,\mu s), (1200\,B), (6))$
TC-TS Topology Size	$((ring), (12, 24, 48, 96), (44), (400\,\mu s), (100\,B), (6)),$ $((mesh), (12, 25, 47, 95), (43), (400\,\mu s), (100\,B), (6))$
TC-SSS Stream Set Size	$((ring), (24), (44, 66, 89, 111), (400\,\mu s), (100\,B), (6)),$ $((mesh), (25), (43, 64, 85, 107), (400\,\mu s), (100\,B), (6))$
TC-DS Design Space	$((ring), (24), (66), (160\,\mu s, 200\,\mu s, 280\,\mu s, 400\,\mu s, 640\,\mu s), (100\,B), (6)),$ $((mesh), (25), (64), (160\,\mu s, 200\,\mu s, 280\,\mu s, 400\,\mu s, 640\,\mu s), (100\,B), (6)),$ $((ring), (24), (66), (400\,\mu s), (100\,B), (1.5, 3, 6)),$ $((mesh), (25), (64), (400\,\mu s), (100\,B), (1.5, 3, 6))$

Tabelle 4.6: Parameterliste der Unicast-Testcases. Bei der Erzeugung der Testcases wird aus jedem Tupel das kartesische Produkt gebildet, in dem jedes Einzelelement eine Parameterkombination $sg := (ns, |\mathcal{V}^{es}|, |F|, ss^{ct}, ss^{fs}, ss^{lf})$ für den Szenariogenerator darstellt (vgl. Kapitel 4.2.2.1).

hier eine Unicast- und eine Multicast-Variante aller Testcases entworfen. In den Unicast-Testcases besitzt jeder Streams stets nur einen Empfänger, während die Multicast-Testcases für jeden Stream die Empfängeranzahl nach der Verteilung in Tabelle 4.5b zufällig ermitteln. Um einen Eindruck über die abgedeckten Parameterkombinationen zu vermitteln, ist die vollständige Parameterliste der Unicast-Testcases in Tabelle 4.6 gegeben. Die Multicast-Testcases nutzen abweichende Parameter (siehe Anhang A.3), da sie zum einen aufgrund der Multicasts an die gesteigerte Linkauslastung nahe der Endgeräte in Empfangsrichtung angepasst werden müssen und zum anderen in dieser Arbeit umfangreicher gestaltet wurden (Nutzung von *fattree* als dritte Topologie), da sie nur für Tests von besonders performanten Schedulern genutzt wurden. Im Zuge der Evaluation sind die jeweiligen Szenarioparameter auch stets in den entsprechenden Abbildungen ablesbar. Die Szenarien beider Varianten finden sich online in [E18]. An dieser Stelle soll der jeweilige Grundgedanke aller Testcases erläutert und mithilfe eines Ausschnitts der Ergebnisse für die Unicast-Testcases illustriert werden. Eine Präsentation aller Ergebnisse ist aufgrund des Umfangs nicht möglich, die vollständigen Ergebnisdaten und -plots stehen jedoch online in [E19] zur Verfügung. Alle Abbildungen dieser Arbeit (Ergebnisdiagramme sowie konzeptionelle Darstellungen) sind mit Hilfe von `Matplotlib` [73] erstellt.

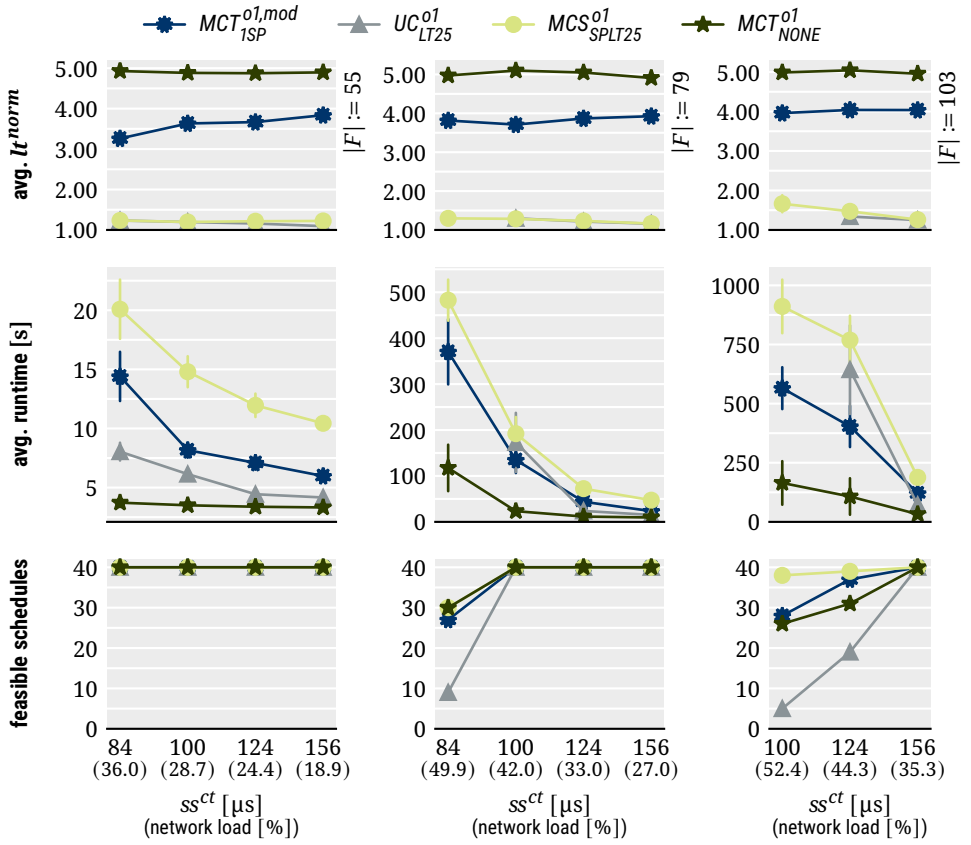


Abbildung 4.4: Ergebnisse für TC-L auf Basis der Meshtopologie für vier Scheduler. Wiederholter Parametersweep von ss^{ct} für $|F| := 55$, $|F| := 79$ und $|F| := 103$ (v.l.n.r.). Konstante Szenarioparameter $ns := mesh$, $|\mathcal{V}^{es}| := 9$, $ss^{fs} := 1500\,\mathrm{B}$ und $ss^{lf} := 6.0$.

TC-L (Load) *Hohe Netzwerkauslastung; testet graduellen Anstieg der Netzwerkauslastung; modelliert durch schrittweise Verkleinerung von ss^{ct} für drei verschiede $|F|$.* Bei diesem Testcase sollen die Fähigkeiten der Scheduler bezüglich zunehmender Ressourcenkonflikte verglichen werden. Dies wird hier durch Verkleinerung von ss^{ct} realisiert, da ein Anstieg der Streamanzahl $|F|$ zugleich die Größe der Probleminstanzen verändern würde, was sich unabhängig von Ressourcenkonflikten bereits stark auf die Performance auswirken würde. Bei der Parameterwahl war das Ziel, dass abnehmende Schedulability und zunehmende Rechenzeiten in den Ergebnissen beobachtbar sind (vgl. Abb. 4.4). Bei zu kleinem $|F|$ oder zu großem ss^{ct} zeigen sich diese Effekte beispielsweise nicht, was wiederum bedeuten würde, dass die Netzwerkauslastung zu gering ist, um die Scheduler bezüglich des Auflösens von Ressourcenkonflikten an ihre Grenzen zu bringen. Da dieser Grenzbereich vom jeweiligen Scheduler abhängig ist, beinhaltet TC-L den Parametersweep von ss^{ct} für drei verschiedene Größen des Verkehrsmusters $|F|$.

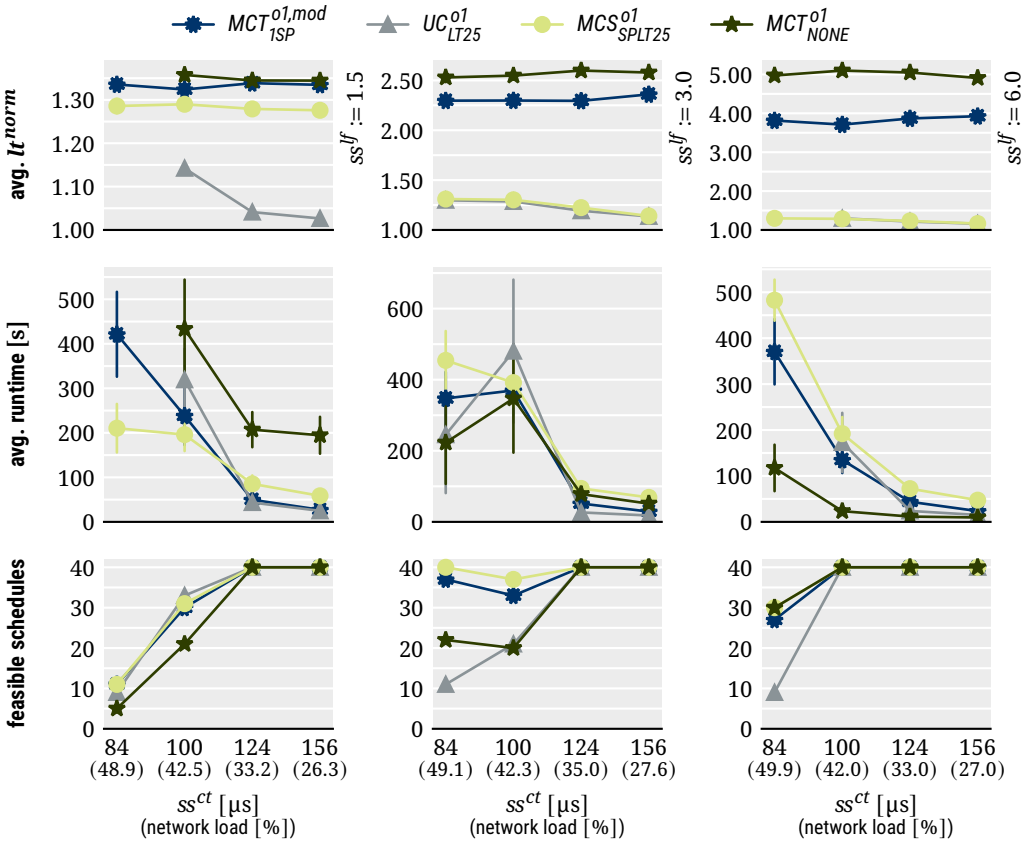


Abbildung 4.5: Ergebnisse für TC-LL auf Basis der Meshtopologie für vier Scheduler. Wiederholter Parametersweep von ss^{ct} für $ss^{lf} := 1.5$, $ss^{lf} := 3.0$ und $ss^{lf} := 6.0$ (v.l.n.r.). Konstante Szenarioparameter $ns := mesh$, $|\mathcal{V}^{es}| := 9$, $|F| := 79$ und $ss^{fs} := 1500\text{B}$.

TC-LL (Load-Latency) *Hohe Netzwerkauslastung; testet Interaktion zwischen Latenzschranken der Streams und Netzwerkauslastung; modelliert durch wiederholte Sweeps von ss^{ct} bei unterschiedlichen ss^{lf}.* Wie schon in TC-L soll hier die Fähigkeit der Scheduler, Ressourcenkonflikte aufzulösen, erprobt werden. Während dieser Test in TC-L jedoch unter großzügigen Latenzschranken der Streams (d.h. Szenarioerstellung mit großem ss^{lf}) erfolgte, wird dieser Test hier für zwei weitere, strengere ss^{lf} durchgeführt. Konkret erlaubt dies einem Scheduler weniger Bufferingzeiten in Bridges und beschränkt die Nutzung längerer Routen, was beides die Möglichkeiten zur Auflösung von Ressourcenkonflikten reduziert. In Abbildung 4.5 zeigt sich diese Einschränkung insbesondere durch eine schlechtere Schedulability für $ss^{lf} := 1.5$ bei kleinem ss^{ct}.

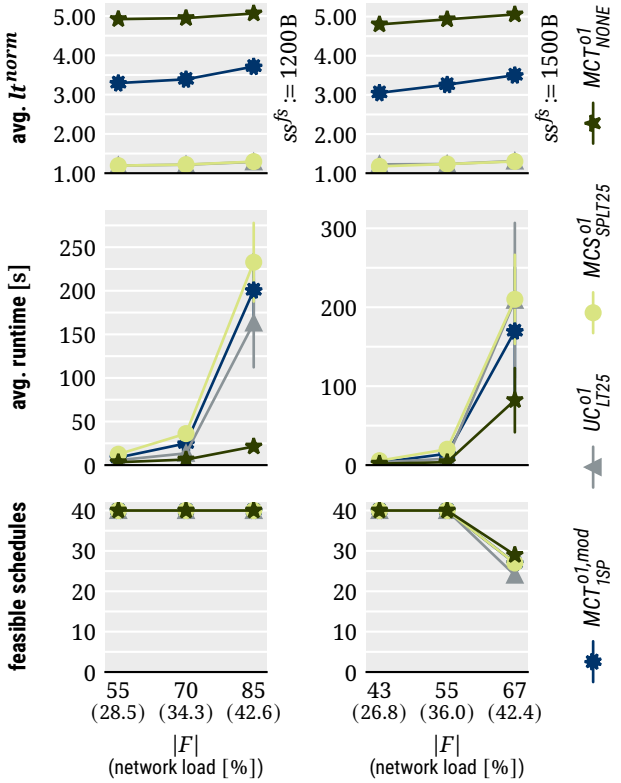


Abbildung 4.6: Ergebnisse für TC-G auf Basis der Meshtopologie für vier Scheduler. Wiederholter Parametersweep von $|F|$ für $ss^{fs} := 1200\text{B}$ und $ss^{fs} := 1500\text{B}$. Konstante Szenarioparameter $ns := mesh$, $|\mathcal{V}^{es}| := 9$, $ss^{ct} := 84\mu\text{s}$ und $ss^{lf} := 6.0$.

TC-G (Granularity) *Hohe Netzwerkauslastung; testet gleiche Last bei unterschiedlichen Kombinationen von Streamanzahl $|F|$ und Framegröße ss^{fs}.* Während die Szenarien von TC-L und TC-LL ausschließlich mit $ss^{fs} := 1500\text{B}$ und somit großen Frames (die exakte Größe hängt von der zufällig gewählten Verkehrsklasse eines Streams ab) generiert wurden, werden in diesem Testcase als Vergleich Szenarien mit $ss^{fs} := 1200\text{B}$ und um den Faktor 1.25 größerem $|F|$ erzeugt. Beim Vergleich von Schedulern wird somit sichtbar, ob sich bestimmte Scheduler besser für die Planung vieler kleinerer Streams oder für die Planung weniger größerer Streams eignen. Dabei kommen zwei gegensätzliche Effekte zum Tragen: Einerseits die gesteigerte Größe der Probleminstanzen bei größerem $|F|$ und andererseits die bessere Planbarkeit der Streams bei kleinerem ss^{fs}. In Abbildung 4.6 sind beide Effekte beim Vergleich der Ergebnisse für $ss^{fs} := 1200\text{B}$ und $ss^{fs} := 1500\text{B}$ sichtbar. Zum einen zeigen sich für wenige, große Streams ($ss^{fs} := 1500\text{B}$) bessere Rechenzeiten bei den beiden kleineren Größen des Verkehrsmusters (ca. 27 % bzw. 35 % Netzwerkauslastung), während andererseits bei der höchsten getesteten Netzwerkauslastung die Schedulability schlechter ist.

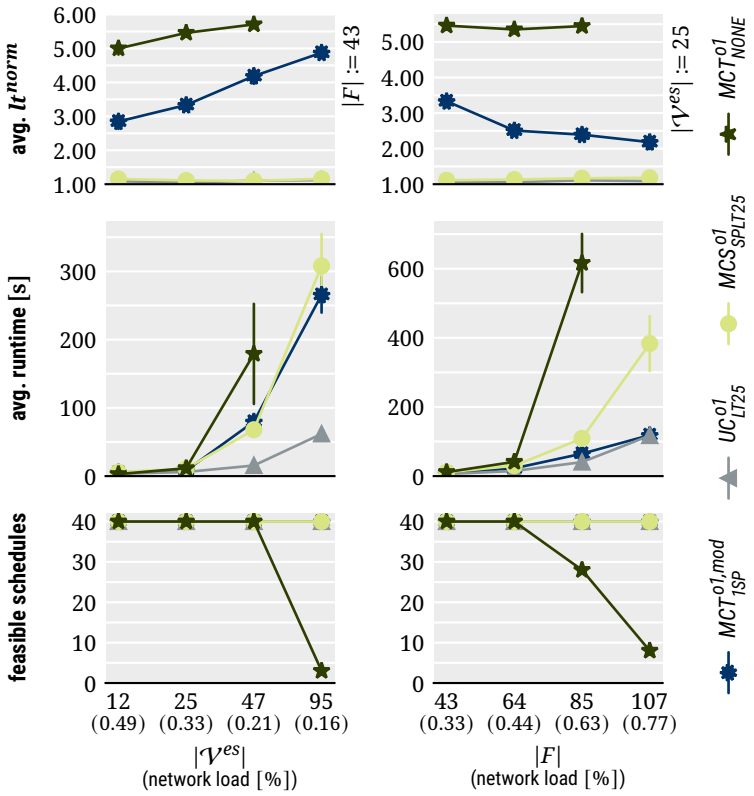


Abbildung 4.7: Ergebnisse für TC-TS (links) und TC-SSS (rechts) auf Basis der Meshtopologie für vier Scheduler. Parametersweep von $|\mathcal{V}^{es}|$ bzw. $|F|$. Konstante Szenarioparameter $ns := mesh$, $ss^{ct} := 400\,\mu s$, $ss^{fs} := 100\,B$ und $ss^{lf} := 6.0$.

TC-TS (Topology Size) *Niedrige Netzwerkauslastung; erprobt Skalierbarkeit von Schedulern bezüglich der Topologiegröße $|\mathcal{V}^{es}|$.* Ein kleines ss^{fs} und großes ss^{ct} werden hier zur Erstellung von Szenarien mit geringer Netzwerkauslastung eingesetzt. Im Vergleich zu den Testcases mit hoher Netzwerkauslastung kommen hier deutlich größere Topologien zum Einsatz. Darüber hinaus wird eine geringe Streamanzahl $|F|$ genutzt, um auch für große Netzwerktopologien Szenarien zu erhalten, die durch aktuelle Scheduler in angemessener Zeit lösbar sind. Wie in Abbildung 4.7 zu sehen, dominiert bei diesem Testcase die Größe der Probleminstanzen in der Regel gegenüber der kleiner werdenden Netzwerkauslastung in den größeren Topologien, sodass es zu einem entsprechenden Anstieg der Rechenzeiten kommt.

TC-SSS (Stream Set Size) *Niedrige Netzwerkauslastung; erprobt Skalierbarkeit von Schedulern bezüglich der Streamanzahl $|F|$.* Dieser Testcase nutzt die gleichen Werte für ss^{fs} und ss^{ct} wie schon TC-TS. Es kommen größere Topologien sowie ein größerer Parametersweep bezüglich $|F|$ zum Einsatz als bei den Testcases mit hoher Netzwerkauslastung. Erwartungsgemäß zeigen die Ergebnisse (Abb. 4.7) einen Anstieg der Rechenzeiten. Unter Berücksichtigung der Ergebnisse von TC-TS und TC-DS ist davon auszugehen, dass hierbei die Größe der Probleminstanzen der dominierende Faktor ist, während die steigende Netzwerkauslastung aufgrund der allgemein sehr niedrigen Auslastung ($< 1\%$) nur eine geringe Rolle spielt.

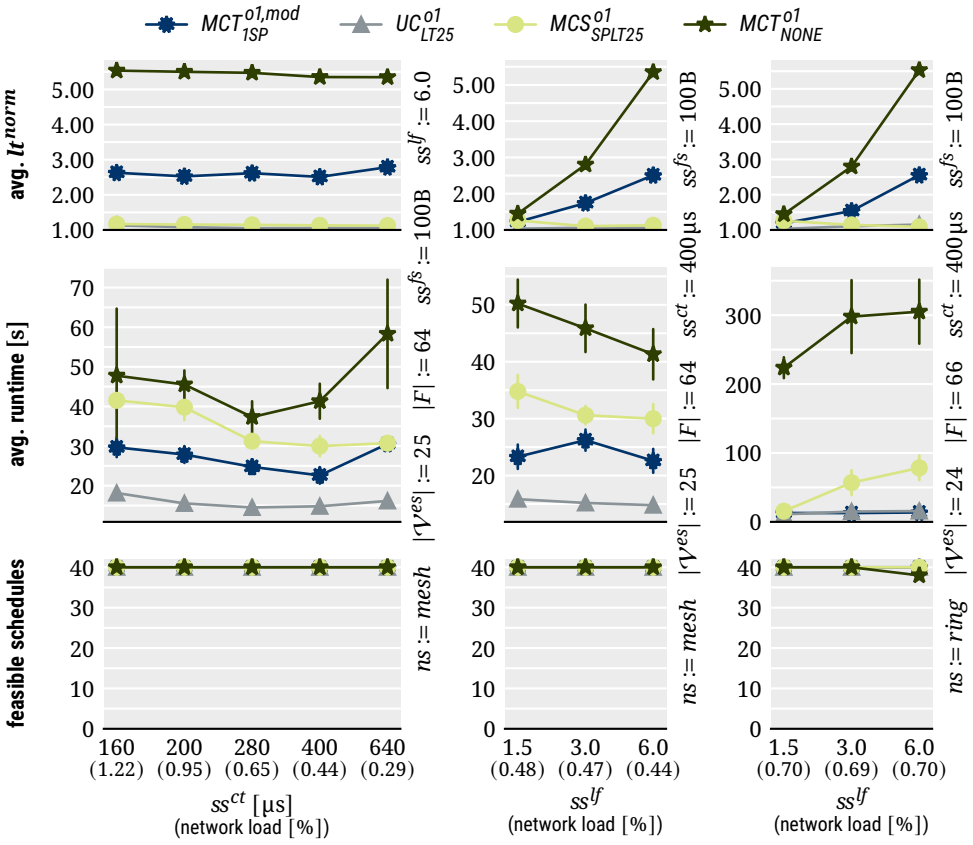


Abbildung 4.8: Ergebnisse für TC-DS. Parametersweep von ss^{ct} bzw. ss^{lf} bei niedriger Netzwerkauslastung.

TC-DS (Design Space) *Niedrige Netzwerkauslastung; erprobt Reaktion von Schedulern bezüglich des Design Space (d.h. der Anzahl von Lösungsmöglichkeiten), welcher durch Variation von ss^{lf} und ss^{ct} verändert wird.* Während TC-TS und TC-SSS ausschließlich mit Sendeintervall $ss^{ct} := 400\,\mu s$ und schwachen Latenzanforderungen ($ss^{lf} := 6.0$) arbeiten, wird hier ähnlich zu TC-LL auch das Verhalten bei verschiedenen Kombinationen dieser beiden Parameter erprobt. Da hier allerdings im Gegensatz zu TC-LL eine niedrige Netzwerkauslastung vorliegt, sollte es eine vergleichsweise geringe Rolle spielen, dass durch strengere Latenzanforderungen Routingoptionen oder längere Bufferingzeiten in Bridges aus den Lösungsmöglichkeiten ausgeschlossen werden. Auch die Veränderung der Sendeintervalle sollte keinen bedeutenden Einfluss auf die Lösbarkeit der Szenarien haben, da sich Ressourcenkonflikte aufgrund der allgemein sehr geringen Netzwerkauslastung leicht vermeiden lassen. In diesem Testcase soll dagegen beantwortet werden, ob die Veränderung dieser beiden Parameter aufgrund ihres großen Einflusses auf die Anzahl der vorhandenen Lösungsmöglichkeiten dennoch einen signifikanten Einfluss auf die Schedulerperformance hat. Im Gegensatz zu den bisher gezeigten Testcases kann hier keine konkrete Erwartung formuliert werden, da die Ergebnisse vom spezifischen Verhalten eines Schedulers abhängen. Abbildung 4.8 zeigt hier im Wesentlichen, dass beide Parameter einen signifikanten Einfluss haben, die vier verschiedenen Scheduler unterschiedlich reagieren, und sich die Ergebnisse maßgeblich gegenüber der Veränderung von ss^{ct} und ss^{lf} bei hoher Netzwerkauslastung unterscheiden (vgl. Abb. 4.5).

4.2.4 Zusammenfassung von Benchmarkingergebnissen

Die bisher gezeigten Ergebnisse der einzelnen Testcases ermöglichen es durch die Separierung nach bestimmten Szenarioparametern, konkrete Stärken und Schwächen der getesteten Scheduler zu identifizieren. Diese äußern sich beispielsweise darin, dass das Ranking (z.B. nach Rechenzeit) der Scheduler sich je nach Testcase bzw. konkreter Parameterkombination unterscheidet. Derartige Erkenntnisse unterstützen die Weiterentwicklung eines Schedulers und untermauern die Bedeutung der sehr vielseitigen Benchmarkingszenarien. Die große Anzahl an Einzelergebnissen erschwert es jedoch, einen Gesamteindruck über die Performance eines Schedulers zu gewinnen.

Zu diesem Zweck können die in Abbildung 4.9 gezeigten, zusammengefassten Ergebnisse bezüglich Schedulability, Rechenzeit und normalisierter Latenz betrachtet werden. In den entsprechenden Darstellungen sind alle Probleminstanzen des jeweiligen Benchmarkings enthalten (hier 2960 Probleminstanzen des Unicast-Datensatzes), wobei die Ergebnisse nur noch nach Instanzen mit hoher Netzwerkauslastung (gekennzeichnet als $ss^{fs} > 100\,\mathrm{B}$) oder niedriger Netzwerkauslastung ($ss^{fs} \leq 100\,\mathrm{B}$) gruppiert sind. Diese Gruppierung wurde gewählt, da sich in einer Vielzahl von Tests gezeigt hat, dass Performanceunterschiede zwischen Schedulern in diesen beiden Bereichen oftmals sehr unterschiedlich ausfallen. Als Beispiel können hier MCT^{o1}_{NONE} und UC^{o1}_{LT25} betrachtet werden, die hier im Falle hoher Netzwerkauslastung ähnliche Rechenzeiten und Schedulability zeigen, während UC^{o1}_{LT25} bei niedriger Netzwerkauslastung nur einen Bruchteil der Rechenzeit von MCT^{o1}_{NONE} benötigt.

In der Darstellung der Schedulability wird dabei im Falle ungelöster Probleminstanzen unterschieden zwischen solchen Ergebnissen, für die Unlösbarkeit vom Scheduler nachgewiesen wurde (*infeasible*), und Suchvorgängen die aufgrund des Zeitlimits (hier $20\,\mathrm{min}$) abgebrochen wurden, bevor eine Lösung gefunden wurde (*tl w/o sol*). Für gelöste Szenarien ist dagegen im Falle optimierender Scheduler (hier MCS^{o1}_{SPLT25} und UC^{o1}_{LT25}) gekennzeichnet, ob der Optimierungsvorgang durch Erreichen des Optimierungsziels (*optimal*) oder des Zeitlimits (*tl w/ sol*) beendet wurde. Für die nicht-optimierenden Scheduler (hier MCT^{o1}_{NONE} und $MCT^{o1,mod}_{1SP}$) wird jede gelöste Instanz als optimal gelöst aufgeführt.

Die Rechenzeit wird in der Zusammenfassung in drei Diagrammen präsentiert:

- Summe der Rechenzeiten über alle Instanzen der jeweiligen Gruppe (zweite Darstellung von unten in Abb. 4.9). Die Gruppe mit hoher Last enthält für den gezeigten Unicast-Datensatz beispielsweise 1920 Instanzen pro Scheduler, die Gruppe mit niedriger Last 1040 Instanzen.
- Summe der Rechenzeiten über Instanzen, die von allen im Vergleich gezeigten Schedulern erfolgreich gelöst wurden (mittlere Darstellung in Abb. 4.9). Die Anzahl solcher *commonly solved instances* (c.s.i.) ist stets im Gruppenlabel angegeben. Im Beispiel wurden 1474 der 1920 Instanzen mit hoher Last und 873 der 1040 Instanzen mit niedriger Last von allen vier Schedulern im Vergleich gelöst und sind in der jeweiligen Darstellung der c.s.i.-Rechenzeit enthalten. Die c.s.i.-Rechenzeit ermöglicht einen fairen Vergleich für gelöste Instanzen, da für alle Scheduler die Rechenzeiten aus exakt denselben Instanzen enthalten sind, wohingegen die zuvor diskutierte Rechenzeit aller Instanzen in vielen Fällen von ungelösten, zeitlimitierten Instanzen (*tl w/o sol*) dominiert wird und daher kaum Mehrwert gegenüber dem Schedulability-Vergleich bietet.

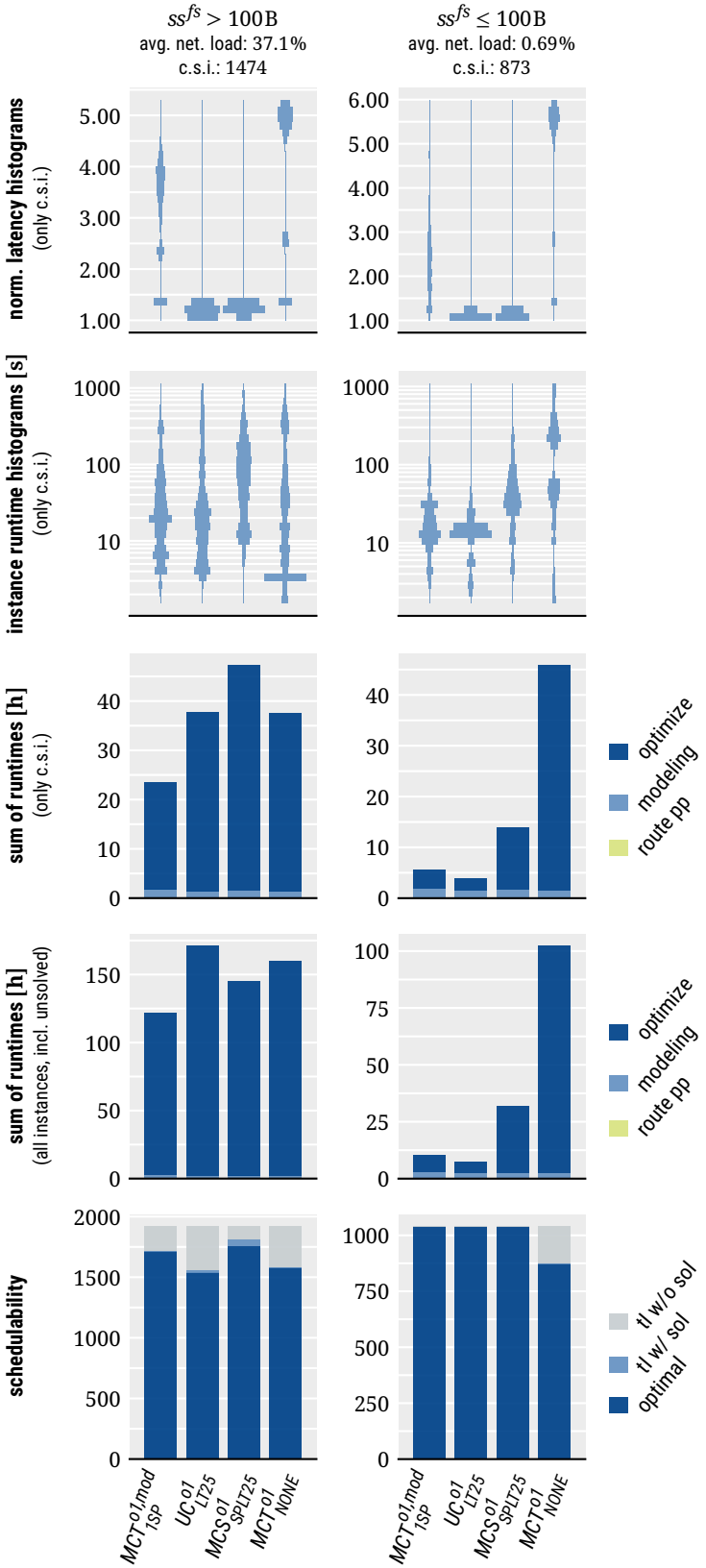


Abbildung 4.9: Zusammenfassung von Benchmarkingergebnissen (hier: vier Scheduler, Unicast-Szenarien).

- Verteilung (Histogramm) der c.s.i.-Rechenzeiten (zweite Darstellung von oben in Abb. 4.9). Diese gibt Aufschluss über die auch bei ähnlicher Summe der Rechenzeiten teilweise sehr unterschiedliche Rechenzeitverteilung einzelner Scheduler.

Die Rechenzeit ist außerdem in unterschiedliche Lösungsschritte der Scheduler unterteilt, welche in Kapitel 6 erläutert werden.

Abschließend wird die Lösungsgüte durch die Verteilung der normalisierten Latenz lt^{norm} für gemeinsam gelöste Szenarien (c.s.i.) dargestellt. Diese Darstellung gibt ggf. Aufschluss darüber, inwiefern Rechenzeitunterschiede mit unterschiedlich guten Lösungen einhergehen.

Im Rahmen der in dieser Arbeit durchgeführten Schedulervergleiche werden in vielen Fällen primär die hier vorgestellten zusammengefassten Ergebnisse analysiert. Zusätzlich wurde anhand der Ergebnisse der einzelnen Testcases stets (auch wenn dies nicht explizit erwähnt wird) geprüft, dass durch die Verwendung der zusammengefassten Daten keine wesentlichen Eigenschaften der getesteten Scheduler übersehen wurden. In der Arbeit nicht diskutierte Diagramme von einzelnen Testcases (einsehbar in [E19]) können somit möglicherweise weitere Details des Schedulervergleichs liefern, sollten jedoch grundsätzlich nicht den hier getroffenen Aussagen widersprechen.

4.2.5 Diskussion der Testcaseparameter

Trotz der für jeden Testcase klar definierten Zielsetzung bestand die Herausforderung bei der Erstellung der Testcases vor allem in der genauen Parameterwahl. Dies liegt insbesondere an Einschränkungen, die bezüglich der Rechenzeit einzelner Szenarien bzw. des gesamten Benchmarkings bestehen:

- Die Rechenzeit eines Schedulers für eine einzelne Probleminstanz (d.h. einmaliges Lösen eines Szenarios ohne Rechenwiederholungen) sollte nicht nur wenige Sekunden betragen, da die Unterschiede zwischen Schedulern in diesem Bereich in der Regel nicht praxisrelevant sind. Die Szenarioparameter sollten also so gewählt sein, dass mehrheitlich Rechenzeiten von mehr als $10\mathrm{s}$ erreicht werden.
- Andererseits wurde in dieser Arbeit durchgängig ein Zeitlimit von $1200\mathrm{s}$ für das Lösen einer Probleminstanz angesetzt, um die Gesamtrechenzeit des Benchmarkings in einem Bereich zu halten, der das Testen einer Vielzahl von Schedulern ermöglicht. Dementsprechend mussten die Szenarioparameter so gewählt werden, dass die Benchmarkingszenarien auch mehrheitlich innerhalb dieses Zeitlimits lösbar sind.
- Mit Rücksicht auf die Gesamtrechenzeit des Benchmarkings wurde bei der Parameterwahl darauf geachtet, dass es möglichst viele Überlappungen bei den Parametern der verschiedenen Testcases gibt (siehe z.B. Abb. 4.4 und Abb. 4.5 für $ss^{lf} := 6$). Im Falle solcher Überlappungen wurden dann keine dedizierten Szenarien für jeden Testcase erzeugt (also mehrfach mit gleichen Parametern) und von den Schedulern gelöst, sondern dieselben Szenarien und Ergebnisse in der Auswertung mehrerer Testcases genutzt.
- Des Weiteren ist es aufgrund des Rechenzeitproblems nicht möglich, die gewünschte Parametervariation (z.B. von ss^{ct} in TC-L) innerhalb der Testcases in kleinen Schritten und

über einen großen Bereich durchzuführen. Die Szenarioparameter müssen also trotz dieser Einschränkung bezüglich Abtastung und Wertebereich präzise so gewählt werden, dass der vorgesehene Effekt eines Testcases sichtbar ist. Der in TC-L durchgeführte Parametersweep von ss^{ct} hätte beispielsweise bei noch größeren Werten von ss^{ct} (als in Abbildung 4.4 gezeigt) einen wesentlich geringeren Einfluss auf die Rechenzeiten und keinen messbaren Einfluss auf die Schedulability, während ein noch kleineres ss^{ct} zu überwiegend nicht planbaren Szenarien führen würde und somit ebenfalls keine verwertbaren Ergebnisse liefert.

Aufgrund der genannten Einschränkungen sind die Parameter der hier vorgestellten Testcases das Ergebnis von wiederholten, zeitaufwändigen Testdurchläufen mit anschließender Parameteranpassung, bis sowohl die gewünschten Effekte der Testcases sichtbar waren als auch die aufgeführten Anforderungen an die Rechenzeit erfüllt wurden.

Entsprechend der Ziele aus Kapitel 4.1.3 sollen die Testcases die bestmögliche Testabdeckung unter Beachtung des Rechenzeitproblems bieten. Dabei ist auch die Interaktion zwischen Parametern berücksichtigt, indem mehrfach Parametersweeps eines Parameters in unterschiedlichen Kombinationen mit anderen Parameter durchgeführt werden. In allen Analysen in dieser Arbeit wird durch die konsequente Verwendung von 40 Datenpunkten pro Parameterkombination und Scheduler und die Darstellung aller Ergebnisse mit Konfidenzintervallen auch die Signifikanz der Ergebnisse einbezogen. Außerdem sind die verwendeten Benchmarkingszenarien öffentlich zugänglich [E18], um die in der Literatur erkennbare Problematik von immer wieder neu entworfenen, unzulänglich beschriebenen und zwischen mehreren Publikationen kaum vergleichbaren Testumgebungen zu adressieren. Die Veröffentlichung der konkreten Szenarien ist insbesondere im Hinblick auf die zuvor diskutierte, schwierige Parametrisierung und die Zufallsentscheidungen bei der Szenarioerstellung von Bedeutung. Der veröffentlichte Datensatz nutzt sowohl für Topologien als auch Szenarien ein `json`-Format, welches direkt die formale Beschreibung der Eingangsdaten nach Kapitel 3.1 widerspiegelt.

Bei der weiteren Entwicklung des Schedulerbenchmarkings ist zu berücksichtigen, dass die sinnvolle Parametrisierung der Testcases stets mit den hierbei genutzten Schedulern zusammenhängt. In [E15] wurde daher für einen ähnlichen Datensatz, wie er in dieser Arbeit verwendet wird, gezeigt, dass die erstellten Szenarien neben den in dieser Arbeit vorgestellten Lösungsverfahren auch für zwei weitere Scheduler sinnvoll zur Evaluation eingesetzt werden können. Zukünftig ist mit steigender Rechenleistung sowie Weiterentwicklung der Scheduler jedoch zu erwarten, dass die Testszenarien um schwierigere Probleme (z.B. Skalierung, Topologiekomplexität) ergänzt werden sollten. Ein Schritt in diese Richtung wurde in dieser Arbeit mit dem Multicast-Datensatz gemacht, welcher mit 3760 Probleminstanzen bereits umfangreicher als der hier bisher präsentierte Unicast-Datensatz ausfällt. Des Weiteren kann es zukünftig sinnvoll sein, weitere Testcases zu ergänzen, um beispielsweise spezielle Parameterkombinationen aus realen Problemen besser abzubilden oder besondere Schedulercharakteristika zu verdeutlichen, sofern diese in den aktuellen Szenarien nicht hinreichend enthalten sind. Bei derartigen Weiterentwicklungen handelt es sich allerdings im Wesentlichen um Parameteranpassungen oder die Ergänzung weiterer Testcases, während das hier vorgestellte Vorgehen bei der Szenarioerstellung und Komposition von Testcases beibehalten werden kann.

4.3 Konfiguration der Benchmarkingplattform

Neben zielgerichteten Testdaten, wie hier in Form der Testcases präsentiert, ist für den Vergleich von Schedulern außerdem die genutzte Hardware- und Softwareplattform sowie die konkrete Konfiguration von großer Bedeutung. Während in der Literatur diesbezüglich in der Regel nur die technischen Spezifikationen genannt werden, sollen hier die möglichen Einflussfaktoren genauer diskutiert werden. Dies umfasst zum einen allgemeingültige Maßnahmen bezüglich der zugrundeliegenden Hardware- und Softwareplattform und zum anderen die Konfigurationsmöglichkeiten beim Benchmarking eines multithreadingfähigen Schedulers auf aktuellen Mehrkernsystemen. Hierbei werden konkrete Empfehlungen für Performancemessungen gegeben und deren praktische Umsetzung in dieser Arbeit beschrieben.

4.3.1 Einfluss von Hardware- und Softwareplattform

Da das Benchmarking mittels der definierten Testcases insgesamt 2960 Probleminstanzen (Unicast-Datensatz) bzw. 3760 Probleminstanzen (Multicast-Datensatz) mit einem Zeitlimit von jeweils $20\mathrm{min}$ umfasst, dauert das Testen von einem oder mehreren Schedulern in der Regel mehrere Tage. Bei der Weiterentwicklung von Schedulern sollen allerdings regelmäßig die Auswirkungen iterativer Veränderungen und neu hinzugefügter Optionen überprüft werden. Da es aufgrund des Zeitaufwands des Benchmarkings nicht möglich ist, bei jedem solcher Tests alle zu vergleichenden Versionen eines Schedulers erneut zu testen, muss stattdessen eine stabile Testplattform geschaffen werden, die die Vergleichbarkeit zwischen neuen und früheren Benchmarkingergebnissen über einen längeren Zeitraum gewährleistet. Neben der selbstverständlich konstant zu haltenden Hardware sollten folgende Einflussfaktoren berücksichtigt werden:

- Die *Biosversion* der verwendeten PCs bzw. Mainboards sollte konstant gehalten werden. Insbesondere zu Beginn des Lebenszyklus einer Plattform[1] können entsprechende Updates zu messbaren Performanceunterschieden führen. Des Weiteren werden über neue Biosversionen auch Aktualisierungen des CPU-Microcodes ausgeliefert, welche in der Vergangenheit insbesondere im Zuge der Behebung von CPU-Sicherheitslücken wiederholt zu signifikanten Performanceunterschieden geführt haben.
- Die *Betriebssystemversion* einschließlich aller installierten Softwarepakete und -bibliotheken sollte konstant gehalten werden. Auch über Betriebssystemupdates werden u.a. Updates des CPU-Microcodes ausgeliefert. Außerdem kann durch die Veränderung der installierten Softwarebibliotheken sämtliche Anwendungssoftware, die durch dynamisches Linking auf diese zurückgreift, in der Performance beeinflusst werden.
- Die *Softwareversionen* der genutzten Anwendungssoftware sollten konstant gehalten werden. Bei den in dieser Arbeit durchgeführten Performancemessungen sind beispielsweise die Pythonversion und die Version des genutzten ILP-Solvers Gurobi relevant. Während Gurobi aufgrund seines Anwendungsgebiets grundsätzlich kontinuierlich bezüglich der Performance

[1] gemeint ist hier z.B. AMD AM4, Intel LGA1151, usw.

weiterentwickelt wird, kommt Python in dieser Arbeit bei der ILP-Modellbildung zum Einsatz, welche in den gezeigten Messungen der Rechenzeit ebenfalls enthalten ist.

- Die *CPU-Taktfrequenz* sollte konstant gehalten werden. Bei modernen Plattformen empfiehlt sich hierbei die Abschaltung von Turbomodi der CPU, obwohl die durchschnittliche Rechenleistung dadurch (teils signifikant) sinkt. Das Problem der Turbomodi liegt darin, dass die tatsächlich zur Verfügung stehende Rechenleistung unvorhersehbar ist, was für Performancemessungen ungeeignet ist. Unter anderem können Ergebnisse zwischen baugleichen Rechnersystemen nicht mehr verglichen werden, da die gleiche CPU aufgrund von Prozessvariation (im Herstellungsprozess) nicht immer die gleiche Taktfrequenz erreicht. Des Weiteren hängt der Turbotakt eines Mehrkernprozessors von der Lastverteilung auf den Kernen ab. Beim Vergleich von Schedulern würde dies bedeuten, dass Eingabedaten oder Lösungsverfahren mit besserer Parallelisierbarkeit (d.h. Last auf mehr Kernen) durch eine niedrigere Taktfrequenz „bestraft“ werden. Außerdem hängt der Turbotakt im Allgemeinen von Temperatur und Leistungsaufnahme der CPU ab und wird daher u.a. von der Umgebungstemperatur beeinflusst und kann nicht langfristig sichergestellt werden. Der Basistakt wird dagegen in der Regel zuverlässig und lastunabhängig erreicht. Gegebenenfalls sollten Temperatur und Taktfrequenz während Performancemessungen überwacht werden.
- Die *Speicherauslastung* sollte während Performancemessungen grundsätzlich überwacht werden. Während Performancemessungen muss eine hundertprozentige Speicherauslastung und infolge dessen die Verwendung von Auslagerungsdateien vermieden werden.

Bei sämtlichen in dieser Arbeit präsentierten Ergebnissen wurden die Berechnungen auf zehn baugleichen *Dell Optiplex 3070* mit *Intel i7 9700* (deaktivierter Turbomodus), *32GB DDR4-2666*, *Ubuntu 18.04 LTS*, *Python 3.8.6* und *Gurobi 9.1.1* (limitiert auf 6 Threads) durchgeführt und CPU-Taktfrequenz und Speicherauslastung in einem 10-Sekunden-Intervall überwacht. Die oben genannten Maßnahmen wurden mit Ausnahme der Betriebssystemupdates berücksichtigt, was aus praktischen Gründen nicht realisierbar war. Da es sich bei den verwendeten Systemen nicht von Beginn an um ein isoliertes Rechencluster handelte, sondern um regulär über das Netzwerk erreichbare Systeme, war eine regelmäßige Pflege mit Updates notwendig. Allerdings sei an dieser Stelle angemerkt, dass das verwendende Betriebssystem stabile und langfristig kompatible Paketversionen als Zielsetzung besitzt und ein Großteil der Messung nach der Überführung der Systeme in ein isoliertes Rechencluster ohne weitere Updates durchgeführt wurde. Eine erneute Messung von vor diesem Zeitpunkt durchgeführten Messungen war aufgrund der Rechenzeit von mehreren Monaten nicht möglich. Eine genaue *quantitative* Analyse aller oben genannten Einflussfaktoren wurde nicht durchgeführt, da dies als nicht relevant in Bezug auf die Fragestellung der Schedulerperformance erachtet wurde.

Die vorgenommene Verteilung auf mehrere baugleiche Systeme bietet sich bei dem vorliegenden Schedulerbenchmarking an, da eine Vielzahl voneinander unabhängiger Probleminstanzen zu lösen ist. Die zu lösenden Probleminstanzen werden bei diesem Vorgehen in eine *Task Queue* eingereiht, welche von den zehn Rechnersystemen gemeinsam abgearbeitet. Ein durch *Locks* geschützter Zugriff auf die Task Queue sorgt für eine Synchronisierung zwischen den Systemen und verhindert so die doppelte Berechnung einzelner Probleminstanzen. Da der Synchronisierungsaufwand gering

im Verhältnis zur Rechenzeit der Probleminstanzen ist, wird durch das verteilte Rechnen ein nahezu idealer Speedup von 10 erreicht.

4.3.2 Benchmarking auf Mehrkernsystemen

Aufgrund des hohen Zeitaufwands für das Benchmarking ist es notwendig, die zur Verfügung stehenden Rechenressourcen bestmöglich zu nutzen. Es sollte daher berücksichtigt werden, dass Scheduler nur begrenzt von Mehrkernsystemen profitieren. Im Allgemeinen werden Mehrkernsysteme durch einen Scheduler erstens nicht vollständig ausgelastet und zweitens ist der Speedup eines Schedulers durch das Nutzen von mehreren Threads nicht proportional zur Threadanzahl. Diese *Parallelisierbarkeit* hängt dabei sowohl vom Lösungsverfahren als auch von den Eingabedaten ab. Um das Benchmarking möglichst schnell durchführen zu können, kann also in Betracht gezogen werden, die Scheduler in der Threadanzahl zu begrenzen und ein Mehrkernsystem durch mehrere voneinander unabhängige Schedulerinstanzen auszulasten. Die Schedulerinstanzen arbeiten dann nach dem gleichen Prinzip wie die zehn vorhandenen Rechnersysteme an der gleichen Task Queue. Auf den hier genutzten *Intel i7 9700* mit 8 CPU-Kernen könnten beispielsweise zwei auf 4 Threads limitierte Schedulerinstanzen anstelle von einer Schedulerinstanz mit 8 Threads betrieben werden. Die Idee dabei ist, dass mit zwei unabhängig arbeitenden Schedulerinstanzen pro Rechner ein nahezu idealer Speedup von zwei erzielt werden kann (bezüglich der Bearbeitung der gesamten Task Queue) und dies möglicherweise bedeutend mehr ist als der suboptimale Speedup durch die Verdopplung der Threadanzahl einer Schedulerinstanz (z.B. von 4 auf 8 Threads). Grundsätzlich muss dabei beachtet werden, dass die Gesamtanzahl Threads nicht die Anzahl der CPU-Kerne übersteigt (Vermeidung von unfairem Task-Scheduling) und weiterhin eine hundertprozentige Speicherauslastung vermieden wird. Um einen besseren Einblick in diese Thematik zu erhalten und insbesondere für die in dieser Arbeit untersuchten ILP-basierten Scheduler unter Nutzung von *Gurobi 9.1.1* eine effiziente Konfiguration für das Benchmarking zu ermitteln, wurden folgende Untersuchungen durchgeführt:

- Scheduler-Speedup bei steigender Threadanzahl (eine Schedulerinstanz pro Rechner)
- Schedulerperformance bei mehreren Schedulerinstanzen pro Rechner

Konkret soll auf Basis dieser Analysen entschieden werden, mit welcher Threadanzahl und mit wie vielen Instanzen pro Rechner die folgenden Schedulervergleiche in dieser Arbeit durchgeführt werden sollten.

4.3.2.1 Performanceanalyse: Threadanzahl

Um den Einfluss des Threadlimits auf die Performance der ILP-basierten Scheduler zu charakterisieren, werden an dieser Stelle die c.s.i.-Rechenzeiten der vier bereits eingeführten Scheduler (siehe Kapitel 4.2.2.2) unter Nutzung verschiedener Threadlimits betrachtet. In den in Abbildung 4.10 und 4.11 gezeigten Ergebnissen wird das Threadlimit im Superskript gekennzeichnet. Die Kennzeichnung *1x5t* bedeutet, dass ein Threadlimit von 5 sowie maximal eine Schedulerinstanz pro

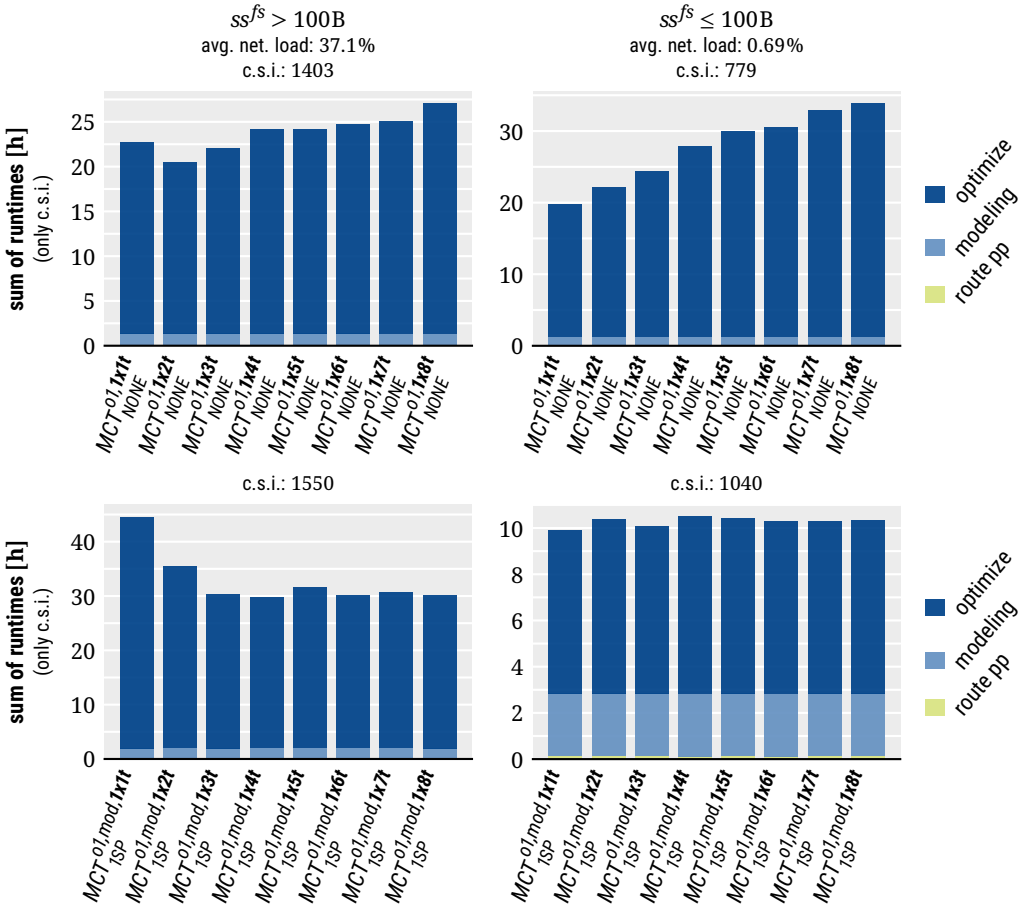


Abbildung 4.10: c.s.i.-Rechenzeiten für zwei nichtoptimierende Scheduler bei variablem Threadlimit.

Rechner genutzt wurde. Die Ergebnisse zeigen, dass die ILP-basierten Scheduler[2] grundsätzlich nicht besonders gut vom Multithreading profitieren: Die Reduktion der Rechenzeit bleibt offensichtlich weit hinter dem idealen Faktor von $1/threadcount$ zurück. Bemerkenswert ist, dass MCT^{o1}_{NONE} bei Erhöhung der Threadanzahl in der Regel sogar eine schlechtere Performance zeigt. Dieses Verhalten wurde nicht weiter untersucht. Es ist jedoch davon auszugehen, dass in diesem Fall folgende negative Effekte des Multithreadings dominieren:

- Softwareoverhead durch Erstellung und Zusammenführung von Threads
- Schlechtere Nutzung der CPU-Caches durch gegenseitige Verdrängung
- Aufteilung der Speicherbandbreite auf konkurrierende Zugriffe durch mehrere Threads

Abbildung 4.12 bestätigt außerdem, dass bei den beiden optimierenden Schedulern keine wesentliche Veränderung bezüglich der Ergebnisgüte (Streamlatenzen) vorliegt. Bei Betrachtung der Ergebnisse der einzelnen Testcases (siehe [E19]) zeigt sich außerdem, dass die Auswirkung der Threadanzahl bei allen vier Schedulern von den Eingangsdaten abhängt. Dies soll hier jedoch nicht im Detail untersucht

[2] Es sei angemerkt, dass die Ergebnisse nur für die genutzten Eingangsdaten und ILP-Modelle gelten und hieraus keine Allgemeingültigkeit abgeleitet werden kann.

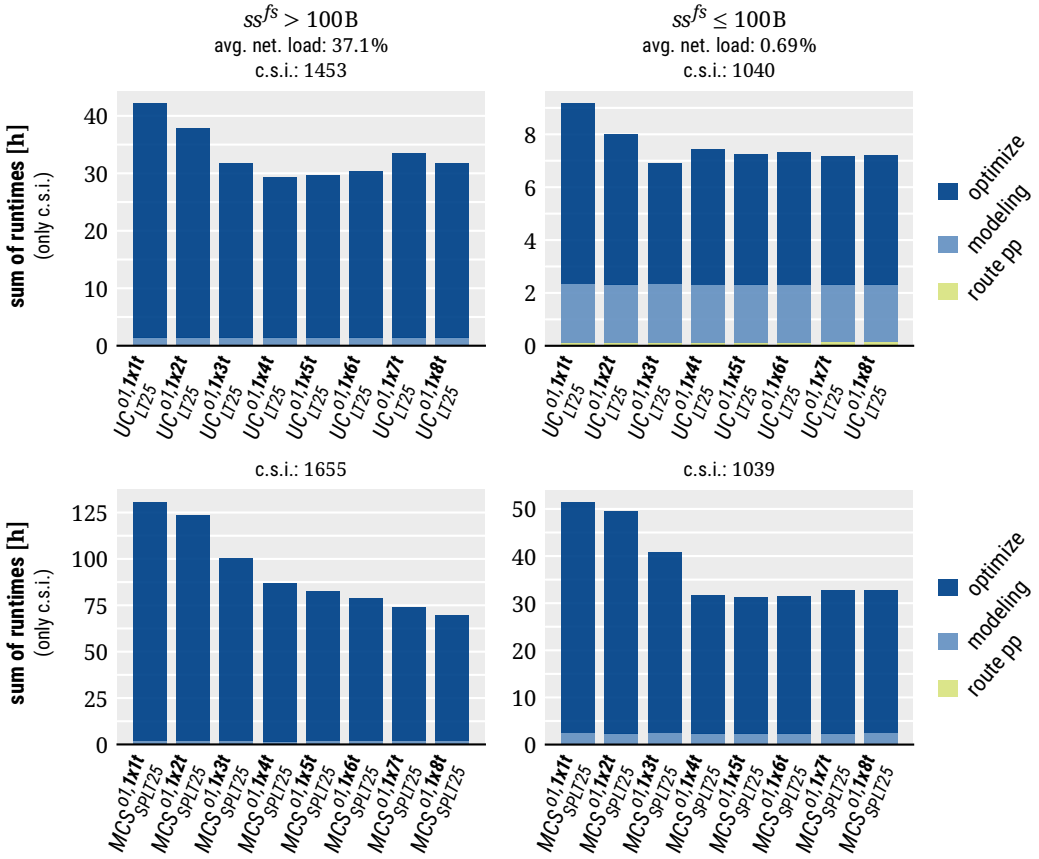


Abbildung 4.11: c.s.i.-Rechenzeiten für zwei optimierende Scheduler bei variablem Threadlimit.

werden, da hier nur die Frage beantwortet werden soll, welche Konfiguration sich am besten zum Vergleich von einer Vielzahl weiterer ILP-basierter Scheduler eignet. Zu diesem Zweck sind die zusammengefassten c.s.i.-Rechenzeiten aus Abbildung 4.10/4.11 ausreichend. Mit Ausnahme von MCT^{o1}_{NONE} zeigen die Ergebnisse, dass trotz des geringen Speedups eine höhere Threadanzahl zu bevorzugen ist, wobei nur MCS^{o1}_{SPLT25} erkennbar von mehr als 4 Threads profitiert. Aufgrund der gegensätzlichen Ergebnisse für MCT^{o1}_{NONE} und die drei anderen Scheduler wäre das ideale Vorgehen für weitere Schedulervergleiche ein Vergleich bei verschiedenen Threadlimits. Da dies allein aufgrund des Zeitaufwands nicht realisierbar ist, wurde in dieser Arbeit (wenn nicht anders angegeben) durchgängig mit einem Threadlimit von 6 gearbeitet. Diese Entscheidung wird damit begründet, dass heutzutage in der Praxis über verschiedenste Hardwareplattformen hinweg üblicherweise von einem Mehrkernprozessor ausgegangen werden. Dementsprechend sollten bei der Entwicklung und Optimierung von Schedulern in dieser Arbeit gezielt die Scheduler bevorzugt werden, die durch eine bessere Parallelisierbarkeit von einem Mehrkernprozessor profitieren. Ein höheres Threadlimit als 6 wurde dagegen vermieden, da hierbei keine oder nur geringe Speedups zu erwarten sind, während im Rahmen des hier durchgeführten Vergleichs dennoch eine höhere Speicherauslastung und Hitzeentwicklung beobachtet wurden. Das niedrigere Threadlimit von 6 soll somit das Auftreten von Throttling und Speicherlimitierungen vermeiden.

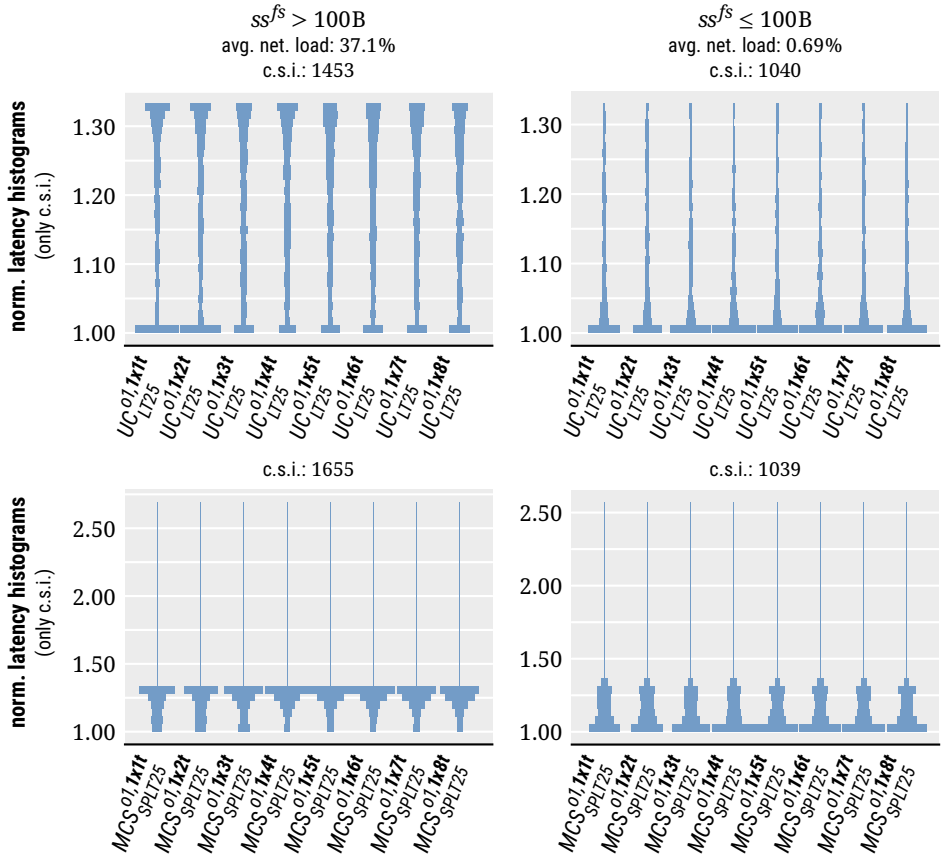


Abbildung 4.12: Histogramme der normalisierten Latenz für zwei opt. Scheduler bei variablem Threadlimit.

4.3.2.2 Performanceanalyse: Schedulerinstanzen pro Rechner

Die bisherige Analyse hat gezeigt, dass Multithreading bei den hier getesteten Schedulern in der Regel keinen überzeugenden Speedup mit sich bringt. Dementsprechend soll hier untersucht werden, ob stattdessen die parallele Ausführung von mehreren Schedulerinstanzen (SI) pro Rechner bei niedrigerem Threadlimit besser geeignet ist, den Zeitaufwand des Benchmarkings bei einem umfassenden Schedulervergleich zu reduzieren. Dabei muss außerdem sichergestellt werden, dass die Ausführung von mehreren SI auf einem Rechner keine Interferenz zwischen den Prozessen erzeugt, die beim Vergleich von Schedulern zu abweichenden Ergebnissen gegenüber dem Benchmarking mit nur einer SI pro Rechner führt. Aufgrund der verwendeten Prozessoren mit 8 Kernen wurden hier zunächst die Konfigurationen *2x4t* (2 SI pro Rechner, 4 Threads) und *1x4t* (1 SI pro Rechner, 4 Threads) verglichen, um den möglichen Speedup und ggf. relevante Prozessinterferenzen abzuschätzen. Die Rechenzeiten für die beiden Konfigurationen bei Nutzung der vier Beispielscheduler sind in Abbildung 4.13 gezeigt. Die Schedulability wird hier nicht gezeigt, da diese sich jeweils nur unwesentlich zwischen den beiden betrachteten Konfigurationen (*1x4t*/*2x4t*) unterscheidet. Grundsätzlich ist festzustellen, dass bei *2x4t* trotz des zur Kernanzahl der CPU passend gewählten Threadlimits Interferenzen auftreten, die zu höheren Rechenzeiten bzgl. der einzelnen Probleminstanzen führen. Von den in Abbildung 4.13 gezeigten Ergebnissen zu unterscheiden ist jedoch die Ausführungszeit eines Benchmarkingdurch-

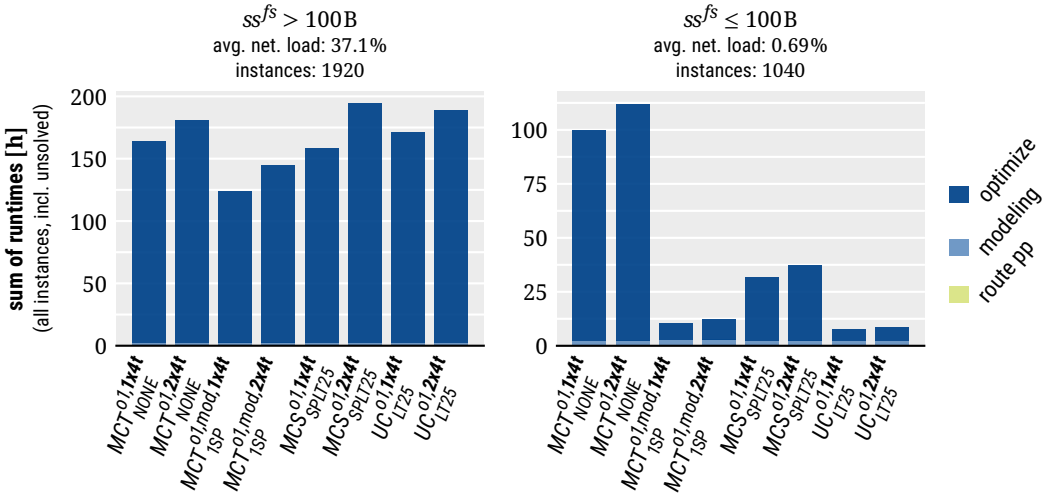


Abbildung 4.13: Summe der Rechenzeiten über alle Probleminstanzen des Benchmarkings für vier Scheduler mit den Rechnerkonfigurationen *2x4t* und *1x4t*.

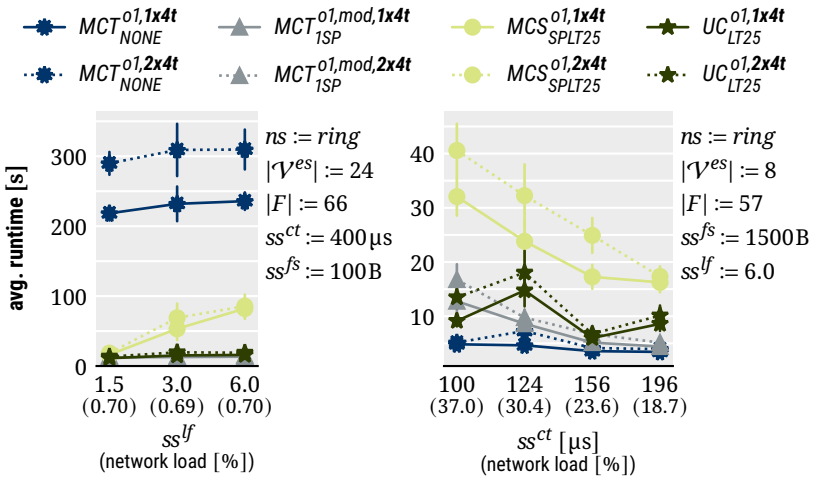


Abbildung 4.14: Auswirkung von Prozessinterferenz in TC-DS bzw. TC-L (vollständige Schedulability für die hier gezeigten Scheduler, d.h., 40 Ergebnisse liegen pro Datenpunkt vor).

laufs. Bei *2x4t* werden alle zu lösenden Probleminstanzen von 20 parallelen Schedulerinstanzen (10 Rechner, 2 SI pro Rechner) bearbeitet, während bei *1x4t* nur 10 parallele Schedulerinstanzen aktiv sind. In Anbetracht des vergleichsweise geringen Anstiegs der Rechenzeit der Probleminstanzen kann mit *2x4t* also beinahe die erhoffte Halbierung der Ausführungszeit des Benchmarkings erzielt werden. Andererseits zeigen bereits die zusammengefassten Ergebnisse in Abbildung 4.13, dass die Prozessinterferenz das Ergebnis eines Schedulervergleichs beeinflussen kann. Für Szenarien mit hoher Netzwerkauslastung ($ss^{fs} > 100\text{B}$) steigt beispielsweise die Rechenzeit von MCS^{o1}_{SPLT25} durch Prozessinterferenz stärker an als von UC^{o1}_{LT25}. Weitere Beispiele für diese Problematik lassen sich in der detaillierten Analyse der einzelnen Testcases finden. Abbildung 4.14 zeigt beispielsweise, dass die Rechenzeiten von MCT^{o1}_{NONE} in TC-DS im Vergleich zu den anderen drei Schedulern wesentlich stär-

ker durch Prozessinterferenz (vgl. *2x4t* und *1x4t*) ansteigen. Derart unterschiedliche Auswirkungen der Prozessinterferenz beeinflussen nicht nur quantitativ die Relationen zwischen den einzelnen Schedulern, sondern könnten möglicherweise auch zu einem abweichenden Ranking führen. Obwohl für die hier getesteten vier Scheduler in keinem der Testcases eine Veränderung des Rankings beobachtet wurde, kann dieses Problem jedoch beim Testen weiterer Scheduler nicht grundsätzlich ausgeschlossen werden. Zur Vermeidung dieses Problems wurde daher außerdem ein Vergleich der Konfigurationen *1x3t*/*2x3t* durchgeführt. Obwohl der Anstieg der Rechenzeiten für *2x3t* hierbei geringer ausfällt als beim Threadlimit von 4, bleibt das grundsätzliche Problem bestehen. Aufgrund dieser Probleme durch Prozessinterferenz wurde das Ausführen von mehreren Schedulerinstanzen pro Rechner beim Schedulervergleich in dieser Arbeit vermieden und stattdessen standardmäßig die im vorigen Abschnitt genannte Konfiguration *1x6t* genutzt.

4.3.2.3 Diskussion: Wahl der Hardwareplattformen

Die gezeigten Ergebnisse bezüglich Threadlimit und Prozessinterferenz sind auch bei der Wahl der Hardwareplattform für Schedulervergleiche zu berücksichtigen. In dieser Arbeit wurde in Betracht gezogen, für die Schedulervergleiche anstelle der Desktoprechner (*Dell Optiplex 3070*) diverse zur Verfügung stehende Serversysteme einzusetzen. Der wesentliche Unterschied zur genutzten Desktopplattform wäre hierbei, dass in der Regel nur wenige bzgl. Hardware und Software identische Systeme (zwecks Vergleichbarkeit) zur Verfügung stehen, diese aber eine hohe Anzahl CPU-Kerne bieten. Da die Ergebnisse des Threadvergleichs zeigen, dass die getesteten Scheduler nicht wesentlich von einer hohen Threadanzahl profitieren, wäre es für eine zeiteffiziente Bearbeitung des Benchmarkings dementsprechend unerlässlich, eine Vielzahl paralleler Schedulerinstanzen auf dem gleichen System zu nutzen. Die Analyse der Prozessinterferenz hat jedoch gezeigt, dass hierbei ein Einfluss auf die Ergebnisse des Schedulervergleichs nicht grundsätzlich ausgeschlossen werden kann. Daher wurde in dieser Arbeit ausschließlich die bereits genannte Testumgebung auf Basis identischer Desktoprechner verwendet.

4.4 Vergleich zum Benchmarking nach Xue

Kurz nach der Veröffentlichung von [E15] hat Xue in [45] ebenfalls konkrete Vorgehensweisen für einen fairen Vergleich von TSN-Schedulern ausgearbeitet. Dabei werden die gleichen wesentlichen Einflussfaktoren auf die Schedulerperformance (vgl. Parametertupel sg) berücksichtigt und im Sinne der Testabdeckung eine umfassende Variation aller Parameter vorgenommen. Auch wenn die Größenordnungen der dort getesteten Parameter große Überschneidungen zu den gewählten Parametern in dieser Arbeit aufweisen, ist ein direkter Vergleich aufgrund der unterschiedlichen, konkreten Parameterkombinationen nicht möglich. Des Weiteren werden in der Arbeit von Xue viele Ziele und Merkmale des hier formulierten systematischen Benchmarkings nicht berücksichtigt:

- Die fehlende Strukturierung der Szenarien in Testcases lässt in der Regel eine klare Testintention der durchgeführten Parametersweeps vermissen.

- Fehlende Angaben für die im Rahmen eines Tests festen (d.h. nicht durch Parametersweeps variierten) Parameter erschweren eine Charakterisierung der durchgeführten Tests. Insbesondere die Netzwerkauslastung bleibt überwiegend unklar.
- Die Anzahl der generierten Szenarien pro Parameterkombination und Rechenwiederholungen pro Szenario bleibt ebenso unklar wie die Gesamtanzahl der Ergebnisse pro Datenpunkt.
- Messungen wurden auf Serversystemen durchgeführt, wobei 64 Schedulerinstanzen pro Rechner gleichzeitig aktiv waren. Nach den Ergebnissen aus Kapitel 4.3 kann eine Interferenz zwischen den Schedulern somit nicht ausgeschlossen werden.
- Weder Szenarien noch Ergebnisdaten wurden veröffentlicht.

Andererseits werden von den dort durchgeführten Tests auch Aspekte berücksichtigt, die hier bisher aufgrund des erhöhten Aufwands nicht beachtet wurden. Dies umfasst u.a.:

- Nutzung verschiedener Mengen für die heterogenen Sendeintervalle der Streams (entspräche hier einer Variation der Verkehrsklassen nach Tabelle 4.5a)
- Nutzung mehrerer *Frames per Stream* (FPS) in einigen Parameterkombinationen
- Protokollierung des Speicherverbrauchs beim Lösen einzelner Probleminstanzen

Insgesamt weisen das Benchmarking nach [45] und das in [E15] vorgeschlagene und hier erweiterte Benchmarking große Überschneidungen in der Zielsetzung und Methodik auf, besitzen jedoch beide auch eine Vielzahl einzigartiger Aspekte. Zukünftig wird es dementsprechend notwendig sein, die beiden Vorschläge in einem einzigen systematischen Benchmarking zusammenzuführen.

4.5 Zusammenfassung Schedulerbenchmarking

In diesem Kapitel wurden zunächst die bisherigen Schwachstellen der Bewertungsverfahren für Scheduler anhand der aktuellen Literatur herausgearbeitet und darauf basierend Ziele für ein systematisches Benchmarking formuliert. Dieses soll demnach alle Parameter der Eingangsdaten des Schedulers (d.h. von Netzwerktopologie und Verkehrsmuster) berücksichtigen und somit eine umfassende *Testabdeckung* gewährleisten. Durch Verwendung mehrerer Szenarien mit der gleichen Parametrisierung sowie Rechenwiederholungen eines Schedulers für ein einziges Szenario soll der Einfluss von Zufallsprozessen bei der Szenarioerstellung sowie dem Scheduling *statistisch auswertbar* sein. Die *Wiederverwendbarkeit* soll durch Veröffentlichung der Benchmarkingszenarien sowie eine präzise Dokumentation der Szenarioerstellung und Datenformate sichergestellt werden. Durch die Veröffentlichung der Szenarien wird außerdem angestrebt, zukünftig publikationsübergreifende Performanceabschätzungen von Schedulern zu ermöglichen.

Die hier mit diesen Zielen entworfenen Benchmarkingszenarien sind in sogenannte *Testcases* eingeteilt, wobei für jeden Testcase eine bestimmte *Testintention* formuliert wurde und somit bestimmte Parametersweeps in den Vordergrund gestellt werden. Bei der Zusammenstellung der Testcases wurde insbesondere auch auf die *Parameterinteraktion* Rücksicht genommen, d.h., dass bestimmte

Parametersweeps (z.B. von $f_k.ct$) für unterschiedliche Kombinationen der anderen Parameter wiederholt werden. Zur einfacheren Auswertung des umfassenden Benchmarkings wurde außerdem eine Zusammenfassung der Ergebnisse vorgeschlagen, bei der nur noch eine Gruppierung nach Netzwerkauslastung der Szenarien erfolgt, da diese als ein besonders ausgeprägter Einflussfaktor auf die Schedulerperformance identifiziert werden konnte.

Mithilfe des erstellten Benchmarkings und vier ILP-basierten Schedulern konnte außerdem gezeigt werden, dass derartige Scheduler von Mehrkernprozessoren nur bis zu einer Threadanzahl von etwa 4 – 6 profitieren und darüber hinaus nur in Einzelfällen ein Performancegewinn verzeichnet werden kann. Da zugleich auch gezeigt werden konnte, dass der parallele Betrieb mehrerer Schedulerinstanzen pro Rechner zu Interferenzen führt (d.h., die Schedulerperformance wird beeinflusst), wurde die Nutzung von Serversystemen mit hoher Kernanzahl für ein umfassendes Benchmarking ausgeschlossen und stattdessen ein Rechencluster aus gleichartigen Desktopsystemen als sinnvollste Plattform für umfassende Schedulervergleiche identifiziert.

Bezüglich der erstellten Benchmarkingszenarien ist zu erwarten, dass diese zukünftig aufgrund steigender Schedulerperformance erweitert und im Schwierigkeitsgrad angepasst werden müssen. Dabei können die hier vorgeschlagenen Konzepte erhalten bleiben. Dies umfasst beispielsweise die Ziele eines systematischen Benchmarking, die Vorgehensweise bei der Szenarioerstellung (Szenariogenerator nach Kapitel 4.2.1) und die Strukturierung der Szenarien in Testcases. Bei einer entsprechenden Erweiterung des Benchmarkings sollten bisher nicht berücksichtigte Aspekte aus anderen Vorschlägen zum Benchmarking (z.B. [45]) aufgegriffen werden.

Kapitel 5

Integer Linear Programming (ILP)

Die in dieser Arbeit vorgestellten Lösungsverfahren für TT-RSP basieren auf ILP. Dabei werden die Bedingungen für gültige Schedules (vgl. Kapitel 3.2) als ein lineares Gleichungssystem abgebildet, in dem die zu treffenden Entscheidungen bezüglich Routing und Scheduling durch Variablen dargestellt werden. Ein derartiges *ILP-Modell* kann dann durch bekannte Algorithmen aus der Mathematik gelöst werden, um einen gültigen Sendezeitplan zu erhalten. Da effiziente Implementierungen dieser Algorithmen in Form von hochoptimierten *Solvern* mit gut dokumentierten Schnittstellen zur Verfügung stehen, wurden diese im Rahmen der Arbeit nicht selbst implementiert oder im Detail analysiert. Dennoch sollen in diesem Kapitel einige der grundlegenden Strategien skizziert werden, da diese die Möglichkeiten und Einschränkungen eines solchen Lösungsansatzes verdeutlichen. Des Weiteren ist dieses Hintergrundwissen notwendig, um den Einfluss bestimmter Modellierungsentscheidungen nachvollziehen zu können, sofern für eine Randbedingung unterschiedliche ILP-Repräsentationen möglich sind. Detaillierte Erläuterungen zu den Lösungsalgorithmen für ILP-Modelle sowie Beweise für relevante Theoreme werden hier nicht gezeigt, da diese für das Verständnis der folgenden Kapitel nicht notwendig sind. Für entsprechende Ausführungen sei auf Vanderbei [47] verwiesen, aus dem auch die hier zusammengefassten Algorithmen und Beispiele entnommen sind.

5.1 Linear Programming (LP)

Unter *Linear Programming* (LP) versteht man einen Ansatz aus der Mathematik, bei dem Probleme durch lineare Gleichungen und Ungleichungen dargestellt werden. Im Fall realer Planungsprobleme werden die zu treffenden Entscheidungen dabei durch die Variablen eines solchen (Un-)Gleichungssystems abgebildet, wohingegen die einzuhaltenden Randbedingungen durch die Gleichungen modelliert werden. Zusätzlich wird eine lineare Zielfunktion auf Basis der gegebenen Variablen formuliert, welche unter Einhaltung des aufgestellten Gleichungssystems optimiert werden soll. Bezüglich TT-RSP müssen beispielsweise Routen und Sendezeitpunkte für jeden Stream eines Verkehrsmusters durch Variablen abgebildet werden, während die Routing- und Schedulingbedingungen nach Kapitel 3.2 in Gleichungen und Ungleichungen überführt werden müssen. Eine mögliche Zielfunktion der Optimierung ist dabei die von den gewählten Sendezeitpunkten abhängige

Summe der Streamlatenzen. Bevor in Kapitel 6 entsprechende Modelle diskutiert werden, soll hier ein einfaches abstraktes Beispiel aus [47, §5.1] zur weiteren Erläuterung dienen:

$$\begin{aligned} \text{Maximize} \quad & 4x_1 + x_2 + 3x_3 \\ \text{Subject to} \quad & x_1 + 4x_2 \leq 1 \\ & 3x_1 - x_2 + x_3 \leq 3 \\ & x_1, x_2, x_3 \geq 0\,. \end{aligned}$$

Eine solche Aufstellung wird auch als Linear Program (LP) bezeichnet. Eine *gültige Lösung* ist eine Variablenbelegung für (x_1, x_2, x_3), welche das Gleichungssystem erfüllt. Ziel eines Algorithmus zum Lösen eines LP ist es, unter allen gültigen Lösungen genau diejenige zu finden, für die die Zielfunktion maximal wird, sofern eine solche Lösung existiert. Andernfalls sollte *Unlösbarkeit* oder *Unbeschränktheit* nachgewiesen werden. Allgemein kann ein LP mit n Entscheidungsvariablen und m Randbedingungen in Standardform folgendermaßen dargestellt werden [47, §2.2]:[1]

$$\begin{aligned} \text{Maximize} \quad & \sum_{j:=1}^{n} c_j x_j \\ \text{Subject to} \quad & \sum_{j:=1}^{n} a_{ij} x_j \leq b_i \qquad \forall i \in \mathbb{N}^+, i \leq m \\ & x_j \geq 0 \qquad \forall j \in \mathbb{N}^+, j \leq n\,. \end{aligned}$$

Mit dem *Simplex*-Algorithmus [47, §§2, 6] und den *Innere-Punkte*-Verfahren [47, §§17–22] stehen Algorithmen zur Verfügung, die LP-Modelle effizient lösen (d.h., die optimale Lösung finden). Die konkreten Algorithmen sollen hier nicht näher betrachtet werden, da sie nicht wesentlich zum Verständnis der folgenden Kapitel beitragen.

Eine besondere und relevante Eigenschaft von LP-Modellen ist die Möglichkeit, eine obere Schranke für den Optimalwert der Zielfunktion zu formulieren, ohne dafür das LP lösen zu müssen. Dies soll hier weiterhin entsprechend [47, §5.1] gezeigt werden. Im bereits eingeführten Beispiel werden hierfür die beiden Ungleichungen mit $y_1 \geq 0$ bzw. $y_2 \geq 0$ multipliziert und dann addiert, was folgende Ungleichung liefert:

$$\begin{aligned} & y_1(x_1 + 4x_2) + y_2(3x_1 - x_2 + x_3) \\ = \; & (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2 x_3 \\ \leq \; & y_1 + 3y_2\,. \end{aligned}$$

Durch Vergleich der Koeffizienten von (x_1, x_2, x_3) in dieser Ungleichung und der Zielfunktion des ursprünglichen (*primalen*) LP lässt sich nun eine obere Schranke für diese Zielfunktion angeben. Konkret muss gefordert werden, dass:

$$\begin{aligned} y_1 + 3y_2 &\geq 4 \\ 4y_1 - y_2 &\geq 1 \\ y_2 &\geq 3\,. \end{aligned}$$

[1] In dieser Arbeit bezeichnen $\mathbb{N}^0$ und $\mathbb{N}^+$ die natürlichen Zahlen mit bzw. ohne Null.

Dann gilt:

$$\begin{aligned} & 4x_1 + x_2 + 3x_3 \\ \leq\ & (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 \\ \leq\ & y_1 + 3y_2\,. \end{aligned}$$

Unter den genannten Bedingungen für y_1, y_2 bildet $y_1 + 3y_2$ somit eine obere Schranke bezüglich der Zielfunktion des primalen LP. Um die Frage zu beantworten, wie gut die bestmögliche Lösung des primalen LP werden kann, muss der kleinstmögliche Wert von $y_1 + 3y_2$ unter Einhaltung der genannten Bedingungen ermittelt werden. Es ist ersichtlich, dass es sich bei dieser Fragestellung selbst um ein LP handelt:

$$\begin{aligned} \text{Minimize} \quad & y_1 + 3y_2 \\ \text{Subject to} \quad & y_1 + 3y_2 \geq 4 \\ & 4y_1 - y_2 \geq 1 \\ & y_2 \geq 3 \\ & y_1, y_2 \geq 0\,. \end{aligned}$$

Allgemein kann dieses *duale* LP eines beliebigen, in Standardform gegebenen primalen LP folgendermaßen beschrieben werden [47, §5.2]:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i:=1}^{m} b_i y_i \\ \text{Subject to} \quad & \sum_{i:=1}^{m} a_{ij} y_i \geq c_j \qquad \forall j \in \mathbb{N}^+,\ j \leq n \\ & y_i \geq 0 \qquad \forall i \in \mathbb{N}^+,\ i \leq m\,. \end{aligned}$$

Grundsätzlich liefert jede gültige Lösung des dualen LP eine obere Schranke für die bestmögliche Lösung des primalen LP. Darüber hinaus kann gezeigt werden (siehe [47, §5.4]), dass das Vorhandensein einer optimalen Lösung für eines der LPs auch das Vorhandensein einer optimalen Lösung für das andere LP impliziert, und dass die Zielfunktionen der beiden LPs in diesem Fall den gleichen Wert annehmen. Aus dieser Eigenschaft ergibt sich ein trivialer Optimalitätsnachweis für gegebene Lösungen der beiden LP: Sofern die Lösungen beim Einsetzen in das jeweilige LP zum gleichen Wert der Zielfunktionen von primalem und dualem LP führen, handelt es sich um optimale Lösungen. Es existieren viele weitere Zusammenhänge zwischen primalem und dualem LP (siehe [47]). Diese werden in modernen Solvern zum effizienten Lösen von LPs ausgenutzt, sind jedoch nicht direkt für das Verständnis der in dieser Arbeit präsentierten Modelle relevant.

Außerdem sei angemerkt, dass die hier gezeigten Methoden und Eigenschaften von Linear Programming auch auf Probleme übertragbar sind, die nicht der hier vorgestellten Standardform entsprechen. So kann eine aus realen Problemen motivierte *Gleichheit* bestimmter Ausdrücke durch zwei *Ungleichungen* abgebildet werden [47, §1.2], und bedeutende Lösungsalgorithmen wie das Simplex-Verfahren sind auf Probleme übertragbar, in denen die Variablen beliebige untere und obere Schranken annehmen (abweichend von 0 und $+\infty$ der hier gezeigten Standardform) [47, §9].

5.2 Integer Linear Programming (ILP)

Viele reale Probleme lassen sich zwar in Form eines LP abbilden, erfordern jedoch als Zusatzbedingung für gültige Lösungen, dass einige oder sogar alle Entscheidungsvariablen ganzzahlige Werte annehmen müssen. Im Rahmen der hier diskutierten zeitgesteuerten Netzwerke müssen die geplanten Sendezeitpunkte beispielsweise in der Regel ganzzahlig sein, da die Sendevorgänge von realen Geräte mit einer diskreten zeitlichen Auflösung initiiert werden. Da grundlegende Kenntnisse des vorherrschenden Lösungsalgorithmus solcher ILP-Probleme für das Verständnis der in dieser Arbeit entworfenen Modelle hilfreich sind, soll dieser hier anhand des Beispiels aus [47, §23.5] erläutert werden. Gegeben sei das ILP:

$$
\begin{aligned}
\text{Maximize} \quad & 17x_1 + 12x_2 \\
\text{Subject to} \quad & 10x_1 + 7x_2 \leq 40 \\
& x_1 + x_2 \leq 5 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{Z}
\end{aligned}
$$

Ignoriert man die Ganzzahlbedingung, so erhält man die sogenannte *LP-Relaxation* des Problems, für welches bereits effiziente Algorithmen (Simplex- und Innere-Punkte-Verfahren) und ein Optimalitätsnachweis für Lösungen existieren. Eine grafische Darstellung des Problems ist in Abbildung 5.1a gezeigt, einschließlich der nicht-ganzzahligen, optimalen Lösung $(x_1, x_2) := (5/3, 10/3)$ der LP-Relaxation. In diesem Beispiel ist ersichtlich, dass das Runden der LP-Lösung nicht zur optimalen Lösung des ILP-Problems führt: Man würde den Punkt $(2, 3)$ erhalten, bei dem es sich nicht um eine gültige Lösung des Problems handelt. Zur Lösung von ILP-Problemen sind demnach Algorithmen erforderlich, die über das Lösen der LP-Relaxation und anschließendes Runden hinaus gehen.

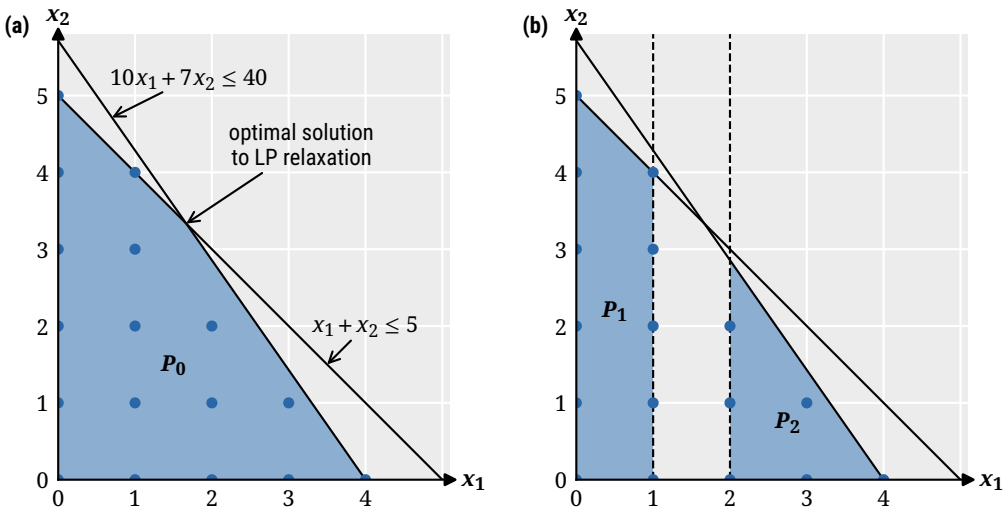


Abbildung 5.1: Grafische Darstellung eines ILP-Beispiels (a) vor und (b) nach dem ersten Branching. Markierte Bereiche kennzeichnen den Lösungsraum der LP-Relaxationen, während hervorgehobene Punkte gültige Lösungen des ILP-Problems sind (nach [47, §23.5]).

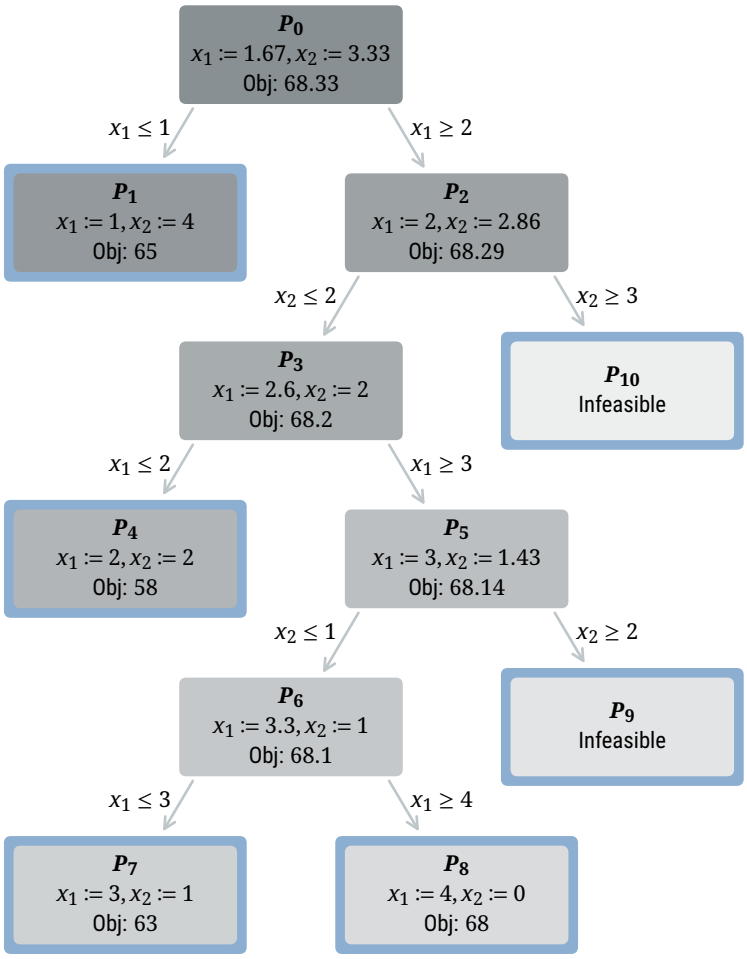


Abbildung 5.2: Vollständiger Branching-Baum des Beispiels (nach [47, § 23.5]). Blau umrandete Teilprobleme erfordern kein weiteres Branching. Die Nummerierung der Teilprobleme entspricht der Bearbeitungsreihenfolge im Branch-and-Bound-Verfahren (auch durch Graustufen gekennzeichnet).

Ein bekanntes und in modernen Solvern implementiertes Lösungsverfahren für ILP-Probleme ist der *Branch-and-Bound*-Algorithmus. Dieser beginnt ebenfalls mit dem Lösen der LP-Relaxation des Problems. Ist die optimale Lösung der LP-Relaxation bereits ganzzahlig, so handelt es sich zugleich um die optimale Lösung des ILP-Problems. Andernfalls wird das Problem durch das Hinzufügen von Zusatzbedingungen in zwei Teilprobleme aufgespalten, welche die gefundene nicht-ganzzahlige Lösung aus dem Wertebereich gültiger Lösungen ausschließen (Branching). Durch Lösen der LP-Relaxationen dieser Teilprobleme wird dann erneut versucht, eine ganzzahlige Lösung zu finden. Das Branching wird gegebenenfalls auch für diese Teilprobleme solange wiederholt, bis ein Teilproblem mit ganzzahliger Lösung gefunden wurde (oder Unlösbarkeit für das jeweilige Teilproblem nachgewiesen wurde). Sobald dieses Vorgehen die erste ganzzahlige Lösung erzeugt hat, müssen außerdem nur noch diejenigen bisher noch nicht bearbeiteten Teilprobleme weiterverfolgt werden, deren optimale Lösung besser ist als die bisher beste ganzzahlige Lösung. Teilprobleme mit schlechterem Zielwert (Obj) müssen nicht weiter betrachtet werden (Bound).

Im vorliegenden Beispiel können die Zusatzbedingungen $x_1 \leq 1$ oder $x_1 \geq 2$ genutzt werden, um die nicht-ganzzahlige Lösung der LP-Relaxation $(5/3, 10/3)$ aus dem Lösungsraum auszuschließen und zwei Teilprobleme $\boldsymbol{P_1}, \boldsymbol{P_2}$ zu erzeugen. Die resultierenden Teilprobleme sind in Abbildung 5.1b gezeigt. Da bei der Erzeugung der Teilprobleme keine ganzzahligen Lösungen des Problems verloren gehen, stellt die beste ganzzahlige Lösung beider Teilprobleme auch die optimale Lösung des ursprünglichen Problems $\boldsymbol{P_0}$ dar. Während die LP-Relaxation von $\boldsymbol{P_1}$ eine ganzzahlige optimale Lösung mit Zielwert 65 in $(1, 4)$ besitzt, ist die Lösung von $\boldsymbol{P_2}$ nicht-ganzzahlig bei einem zugleich besseren Zielwert von 68.29 und erfordert somit weiteres Branching. Dieser Strategie folgend ergibt sich ein Branching-Baum, welcher in Abbildung 5.2 gezeigt ist. Wie zuvor beschrieben, findet immer dann kein weiteres Branching statt, wenn ein Teilproblem eine ganzzahlige optimale Lösung hat, unlösbar ist, oder einen schlechteren Zielwert als die beste bisher gefundene ganzzahlige Lösung besitzt (tritt hier nicht auf). Bei der Bearbeitungsreihenfolge von Teilproblemen wird wie hier gezeigt in der Regel eine *Tiefensuche* eingesetzt, was in [47, §23.5] ausführlich begründet wird. Dennoch bestehen bei der Implementierung von Branch-and-Bound-Algorithmen Freiheitsgrade bezüglich der gewählten Branching-Variable und Branching-Richtung. So wäre im ersten Branching-Schritt des Beispiels auch ein Branching auf x_2 (statt auf x_1) möglich gewesen, und die Tiefensuche hätte auch den jeweils rechten Zweig (nach Abb. 5.2) zuerst erkunden können.

Im Zusammenhang mit den vorgestellten Methoden sollen einige im weiteren Verlauf bedeutende Begriffe eingeführt bzw. konkretisiert werden (vgl. [48]):

Best Bound

Während des Branchings kann zu jedem Zeitpunkt ermittelt werden, welchen Wert die Zielfunktion laut der bisherigen Analysen bestenfalls annehmen kann. Diese *Best Bound* ergibt sich aus dem besten Zielwert aller bisher gelösten LP-Relaxationen von genau den Teilproblemen, für die entweder noch nicht alle Kindknoten (d.h. darunterliegenden Teilprobleme) bearbeitet wurden oder für die kein weiteres Branching erforderlich ist. In Abbildung 5.2 ergibt sich somit beispielsweise nach einmaligem Branching und dem Lösen von $\boldsymbol{P_0} - \boldsymbol{P_2}$ eine Best Bound von 68.29, welche bis zur Bearbeitung von $\boldsymbol{P_{10}}$ bestehen bleibt. Nachdem $\boldsymbol{P_{10}}$ als unlösbar identifiziert wurde, nimmt die Best Bound den Wert 68 aus $\boldsymbol{P_8}$ an.

Optimality Gap

Sofern im Branching bereits ganzzahlige Lösungen gefunden wurden kann durch Vergleich von bisher bester Lösung (*incumbent solution*) und Best Bound erfasst werden, wie weit die aktuelle Lösung *schlimmstenfalls* von der bestmöglichen Lösung entfernt ist. Während der Optimierung mittels Branch-and-Bound verkleinert sich diese *Optimality Gap*, indem bessere Lösungen gefunden werden, oder indem sich die Best Bound durch Bearbeitung stärker eingeschränkter Teilprobleme verschlechtert. Im Beispiel ergibt sich nach dem Lösen von $\boldsymbol{P_0} - \boldsymbol{P_2}$ zunächst eine Gap von $68.29 - 65 = 3.29$, basierend auf der ganzzahligen Lösung aus $\boldsymbol{P_1}$ und der Best Bound aus $\boldsymbol{P_2}$. Diese Gap bleibt bis zur Bearbeitung von Teilproblem $\boldsymbol{P_8}$ bestehen, welche mit einem Zielwert von 68 eine bessere ganzzahlige Lösung als $\boldsymbol{P_1}$ bietet und die Gap somit auf 0.29 reduziert. Nach Bearbeitung von $\boldsymbol{P_{10}}$ und Nachweis von dessen Unlösbarkeit verschlechtert sich die Best Bound auf 68 aus $\boldsymbol{P_8}$ und die Gap sinkt auf 0. Somit ist die Optimalität der Lösung von

P_8 nachgewiesen. Die Optimality Gap bezieht sich immer auf den bestenfalls noch erzielbaren Wert der Zielfunktion und kann absolut oder relativ angegeben werden.

Lösung

Eine Variablenbelegung, die alle Randbedingungen erfüllt, wird in der Literatur vorwiegend als *gültige Lösung (feasible solution)* bezeichnet, im Folgenden jedoch zur Vereinfachung nur als *Lösung*.

Lösungsraum

Der Wertebereich aller gültigen Lösungen (z.B. markierte Bereiche in Abb. 5.1) wird Lösungsraum (*feasible region*) genannt.

Mixed Integer Programming

ILP-Probleme in denen die Ganzzahlbedingung nur für einige Variablen gilt, während andere wiederum kontinuierlich sein dürfen, werden auch als *Mixed Integer Program* (MIP) bezeichnet. Im Verlauf dieser Arbeit wird zur Vereinfachung immer die Bezeichnung ILP verwendet, auch wenn gemischte Variablen vorliegen.

Insgesamt hat dieses Kapitel gezeigt, dass Integer Linear Programming für die Modellierung realer Probleme geeignet ist, sofern sich diese durch lineare Gleichungen und Ungleichungen abbilden lassen. Der Lösungsraum wird dabei durch die genutzten Entscheidungsvariablen und aufgestellten Randbedingungen in eindeutiger Weise beschrieben, wobei das Branch-and-Bound-Verfahren durch die systematische Vorgehensweise grundsätzlich die beste Lösung finden kann, sofern ausreichend Rechenzeit zur Verfügung steht. Gleichzeitig kann der Algorithmus bei vorzeitigem Abbruch (z.B. aufgrund von Zeitlimits) die beste bisher gefundene Lösung unter Angabe einer mathematisch nachgewiesenen Optimality Gap zurückliefern. Die bekannten Lösungsverfahren stehen darüber hinaus bereits in einer Reihe von Solvern zur Verfügung, sodass der wesentliche Implementierungsaufwand sich auf die Problemmodellierung beschränkt.

Kapitel 6

Unicast- und Multicastmodelle für TT-RSP

Bereits im Rahmen der Arbeiten an der OpenFlow-basierten Echtzeitvernetzung [E11] wurde eine einfache Heuristik zur Planung zeitgesteuerter Echtzeitstreams entworfen. Aufgrund fehlender Vergleichswerte in Bezug auf Schedulability, Rechenzeiten und Lösungsgüte war eine konkrete Bewertung des Verfahrens jedoch schwierig. Im gleichen Zeitraum (2015–2016) stieg durch die Verabschiedung verschiedener TSN-Standards die Relevanz von herstellerunabhängigen Echtzeitnetzwerken sowie das Forschungsinteresse an den im zentralen Controller verankerten Planungsverfahren, einschließlich Verfahren zum Lösen von TT-RSP. Um entwickelte Lösungsverfahren zukünftig besser bewerten und vergleichen zu können, sollte somit neben den bisher diskutierten Benchmarkingszenarien außerdem eine *Referenzlösung* entworfen werden, welche Vergleichswerte zur Einordnung der Leistungsfähigkeit liefert. In den folgenden Jahren wurde eine solche Lösung auf Basis von Integer Linear Programming (ILP) erstellt und kontinuierlich weiterentwickelt, um dessen Funktionsumfang und Performance zu verbessern.

In diesem Kapitel werden zunächst die Anforderungen an eine Referenzlösung charakterisiert und die Wahl von ILP als Basis begründet, bevor die erstellten ILP-Modelle vorgestellt und evaluiert werden. Die konkreten Beiträge dieses Kapitels umfassen:

- Das in [E12] veröffentlichte Unicastmodell.
- Das in [E14] veröffentlichte Multicastmodell.
- Eine Vielzahl von Modellvarianten der veröffentlichten Ursprungsmodelle, welche insbesondere der Performanceoptimierung und somit der praktischen Anwendbarkeit der Lösung dienen.
- Die umfassende Evaluation der Modellvarianten auf Basis des Benchmarkings nach Kapitel 4.

Auch wenn einige Bestandteile der Modelle hier in der Reihenfolge präsentiert werden, in der sie auch entwickelt und veröffentlicht wurden, sei angemerkt, dass viele Komponenten zum besseren Verständnis in einer abweichenden Reihenfolge gezeigt werden.

6.1 Zielsetzung der entwickelten Lösungsverfahren für TT-RSP

Nach der Veröffentlichung des Konzepts zu einem OpenFlow-basierten Echtzeitnetzwerk in [E11] zeigte sich, dass vergleichbare Echtzeitnetzwerke durch die Integration von TSN-Funktionen in Bridges zukünftig an Bedeutung gewinnen. Damit einhergehend war ein steigendes Forschungsinteresse an den im zentralen Controller (CNC) platzierten Planungsverfahren für Echtzeitstreams zu erwarten. Um die Leistungsfähigkeit entwickelter Verfahren besser einordnen zu können, ist in diesem Kontext eine Referenzlösung sinnvoll, welche Vergleichswerte hinsichtlich Schedulability, Rechenzeit und Lösungsgüte liefern kann. Eine solche Referenzlösung sollte sich insbesondere durch folgende Aspekte auszeichnen:

- Eine präzise Beschreibungsform der Lösung sollte Nachimplementierungen begünstigen.
- Die Implementierung sollte etablierte und für die Problemstellung übliche Technologien einsetzen, wobei für performancekritische Abschnitte der Implementierung existierende und optimierte Bibliotheken genutzt werden sollten. Somit sollte eine von eigenen Implementierungsentscheidungen weitgehend unabhängige, dem Stand der Technik entsprechende Performance erzielt werden, die als Referenz für weitere Lösungsverfahren dienen kann.
- Das Lösungsverfahren sollte bestmöglich beantworten können, ob eine Lösung für eine Probleminstanz existiert (Lösbarkeitsinformation). Bei der Entwicklung weiterer Lösungsverfahren kann somit im Falle von fehlgeschlagenen Planungsversuchen beantwortet werden, ob dies auf Unzulänglichkeiten des Lösungsverfahrens zurückzuführen ist oder ob die gegebene Probleminstanz keine Lösungen besitzt.
- Das Auffinden optimaler Lösungen hinsichtlich eines gegebenen Optimierungsziels sollte möglich sein, um bei der Entwicklung weiterer Lösungsverfahren einen Referenzwert für die Güte der gefundenen Lösungen bereitzustellen (Optimalitätsinformation). In Anbetracht der vielen verschiedenen Gütekriterien für Schedules (vgl. Kapitel 4.1.1) sollte das Optimierungsziel anpassbar sein.
- Das Lösungsverfahren sollte flexibel und erweiterbar sein, um eine Adaption für verschiedene Anwendungsanforderungen, Technologien (insbesondere Profinet, TTEthernet und TSN) und gegebenenfalls noch ausstehende Veränderungen an TSN zu ermöglichen. Als Beispiel seien hier die in Kapitel 3.2 eingeführten Sonderanforderungen wie Jitter, Queuingmodelle und Frameisolation genannt.

Die eigens entwickelte Lösung aus [E11] ist hinsichtlich dieser Anforderungen nicht ausreichend, da sie als Heuristik auf Basis von *Shortest Path* (SP)-Routing und *As Soon As Possible* (ASAP)-Scheduling nur den ersten Aspekt in Grundzügen erfüllen konnte. Außerdem gab es in 2016[1] weder frei verfügbare Implementierungen zur Lösung von TT-RSP, noch Lösungsansätze aus der Literatur, die den genannten Anforderungen entsprechen. Zuvor veröffentlichte Arbeiten aus der Literatur bearbeiten beispielsweise ausschließlich das Scheduling und vernachlässigen den Einfluss des Routings [3, 2], reduzieren das Problem auf Zeitschlitzreservierungen für eine komplette Route

[1] Beginn der Arbeit an einer eigenen Referenzlösung

anstelle von linkspezifischen Zeitschlitzen [9], oder sind durch den Einsatz umfassender Heuristiken kaum detailgetreu nachimplementierbar [8]. Weitere Details werden diesbezüglich in Kapitel 10 erläutert. Auf Basis dieser Analysen wurde im Anschluss an [E11] mit den Arbeiten einer solchen *erweiterbaren und flexiblen Referenzlösung* für TT-RSP begonnen.

6.1.1 Auswahl der mathematischen Methoden

Zum Lösen von TT-RSP ist eine Vielzahl von bekannten Heuristiken und exakten Lösungsverfahren anwendbar. Seitens der Heuristiken finden sich in der Literatur beispielsweise Lösungen auf Basis von Tabu Search [8], Evolutionären Algorithmen [12] sowie einer Vielzahl problemspezifischer Ansätze [24, 26]. Relevante exakte Lösungsverfahren umfassen ILP [2], *Satisfiability Modulo Theory* (SMT) [3] und *Constraint Programming* (CP) [49]. Aufgrund der aufgestellten Kriterien für die zu entwerfende Referenzlösung wird hier der Einsatz von exakten Lösungsverfahren bevorzugt. Wesentliche Kritikpunkte an bekannten Heuristiken sind eine schwierigere Nachimplementierung sowie fehlende Lösbarkeits- und Optimalitätsinformationen. Nachimplementierungen werden vor allem dadurch erschwert, dass Teile der Lösung in der Regel nicht in Form einer kompakten mathematischen Notation dargestellt werden können. In der Literatur zeigt sich dies beispielsweise durch die regelmäßige Verwendung von Pseudocode, in dem darüber hinaus nicht alle Funktionen ausformuliert werden und somit Teile der Lösung unklar bleiben (siehe z.B. [8, 32, 26]). Fehlende Lösbarkeits- und Optimalitätsinformationen ergeben sich dagegen prinzipbedingt aus dem Lösungsvorgang, bei dem bisher nicht betrachtete Bereiche des Lösungsraumes nicht explizit erfasst und charakterisiert werden. Exakte Lösungsverfahren begegnen diesen Problemen mit einer exakten Spezifikation in einer vorab definierten *formalen Sprache* und einem Lösungsvorgang, der systematisch alle relevanten Bereiche des Lösungsraumes betrachtet (siehe z.B. Branch-and-Bound). Als exaktes Lösungsverfahren wird in dieser Arbeit ILP genutzt, welches konkret die folgenden Funktionen bietet:

- *Ausnutzung existierender Solver:* Bei der Implementierung stehen diverse ILP-Solver zur Verfügung, welche das performancekritische Lösen des ILP-Modells effizient umsetzen.
- *Formalisierung:* Die Formalisierung als Gleichungssystem bietet eine präzise und eindeutige Beschreibung des Problems und des abgedeckten Lösungsraums.
- *Lösungsraumabdeckung:* Bei entsprechender Modellierung ist eine vollständige Abdeckung des Lösungsraums möglich.
- *Lösbarkeitsinformationen:* Ein ILP-Solver kann nicht lösbare Probleminstanzen effizient als solche identifizieren.
- *Flexibles Optimierungsziel:* Das Optimierungsziel im ILP-Modell lässt sich unabhängig von den Randbedingungen flexibel ändern.
- *Optimalitätsinformationen:* Hinsichtlich des gegebenen Optimierungsziels kann der ILP-Solver die Güte einer gefundenen Lösung im Vergleich zur bestmöglichen Lösung der Probleminstanz angeben (Optimality Gap).
- *Erweiterbarkeit:* Zusätzliche Randbedingungen (z.B. Datenabhängigkeit zwischen Streams) können ohne Veränderung des bestehenden Modells ergänzt werden.

Abseits von ILP-Solvern werden die genannten Funktionen gleichermaßen von SMT-Solvern erfüllt. Einen aussagekräftigen Vergleich (bzgl. Performance und Funktionsumfang) der beiden Methoden als Lösungsansatz für TT-RSP bzw. vergleichbaren Problemstellungen gab es jedoch weder zu Beginn der Arbeiten an der Referenzlösung noch zum heutigen Zeitpunkt. Entscheidend für die Wahl von ILP als Basis für die Referenzlösung war daher vor allem die umfassendere Literatur und Softwaredokumentation (z.B. für die ILP-Solver Gurobi und CPLEX). Eine initiale Implementierung beider Verfahren zu Vergleichszwecken wurde nicht in Betracht gezogen, da ein solcher Vergleich auf Basis von „naiven" Implementierungen ohne hinreichende Optimierung beider Verfahren nicht zielführend ist. Dies wird im weiteren Verlauf der Arbeit durch eine Vielzahl von iterativen Entwicklungsschritten der ILP-Modelle verdeutlicht.

6.1.2 Entwicklungsmethodik der ILP-Modelle

Bei einem ILP-basierten Lösungsverfahren liegt der Fokus der Entwicklungsarbeit beim Entwurf von ILP-Modellen, die das gegebene Problem als lineares Gleichungssystem repräsentieren. Anders als in der Literatur zu Lösungsverfahren von TT-RSP gelegentlich suggeriert wird [26, 12], lässt die Verwendung eines exakten Lösungsverfahrens dabei zunächst keine Rückschlüsse auf die tatsächliche Lösungsraumabdeckung oder die erzielbare Performance zu. Die konkreten Eigenschaften eines ILP-basierten Schedulers hängen vielmehr von einer Vielzahl von Designentscheidungen und eingebrachten Optimierungen bei der Modellierung als ILP ab.[2] Der Einfluss solcher Designentscheidungen lässt sich dabei in vielen Fällen direkt mit den in Kapitel 5 erläuterten Lösungsalgorithmen für ILP erklären, was hier an einigen Beispielen veranschaulicht werden soll:

(a) Der identische Lösungsraum kann als ILP oftmals durch sehr unterschiedliche Kombinationen von Entscheidungsvariablen und Randbedingungen ausgedrückt werden.

- Repräsentationen mit weniger Variablen und Randbedingungen können die Branchingtiefe und die benötigte Rechenzeit zum Lösen von LP-Relaxationen reduzieren.
- Unterschiedliche Repräsentationen mit ähnlicher Anzahl von Variablen und Randbedingungen können zu unterschiedlicher Performance führen, da unterschiedliche Variablen zu sehr verschiedenen Teilproblemen im Branching führen.

(b) Vorverarbeitungsschritte können Lösungsmöglichkeiten ausschließen, sodass diese nicht mehr im ILP-Modell enthalten sind. Im Falle von TT-RSP können beispielsweise ungünstige Routingmöglichkeiten der Streams vorab ausgeschlossen werden.

- Derartige Maßnahmen führen zu ILP-Modellen, deren Lösungsraum bedeutend kleiner sein kann, als der des realen Problems. Es handelt sich somit um ein heuristisches Vorgehen.
- Das ILP-Solving profitiert in diesem Fall von kleineren Probleminstanzen.

[2] Der genutzte ILP-Solver hat selbstverständlich ebenfalls einen massiven Einfluss auf die Performance, eine Analyse dieses Aspekts ist jedoch nicht Bestandteil der Arbeit.

(c) Verschiedene Gütekriterien (vgl. Kapitel 4.1.1) in Bezug auf erstellte Schedules werden durch unterschiedliche Optimierungsziele ausgedrückt.

- Unterschiedliche Optimierungsziele führen zu verschiedenen Lösungen in der LP-Relaxation und somit zu verschiedenen Punkten im Lösungsraum, um die herum im Zuge des Branchings Teilprobleme erstellt werden.
- Es ergeben sich sowohl unterschiedliche Branching-Variablen als auch abweichende Suchrichtungen (d.h., welche Teilprobleme in der Tiefensuche zuerst betrachtet werden).

(d) Weitere Designentscheidungen können explizit durch die Lösungsalgorithmen von ILP motiviert sein.

- Zusätzliche Randbedingungen können z.B. den Lösungsraum der LP-Relaxationen genau so einschränken, dass unerwünschte, nicht-ganzzahlige Lösungen entfernt werden, ohne jedoch ganzzahlige Lösungen zu entfernen (vgl. auch Cutting Planes in [48]).
- Dies kann dazu führen, dass früher im Lösungsvorgang ganzzahlige Lösungen gefunden werden und sich die Branchingtiefe verringert.

Trotz dieser grundlegenden Kenntnisse lässt sich im Fall komplexer Probleme wie TT-RSP in vielen Fällen nicht vorhersagen, welche Modellierung tatsächlich am effizientesten ist. Bei der Entwicklung der in dieser Arbeit vorgestellten ILP-Modelle wurden daher kontinuierlich verschiedene Modellvarianten entworfen und mittels des umfassenden Benchmarkings verglichen. Dabei sollte in Übereinstimmung mit der Zielsetzung für eine Referenzlösung stets eine möglichst umfassende Lösungsraumabdeckung beibehalten werden. Zugleich sollten die entworfenen ILP-Modelle bestmöglich hinsichtlich der Performance optimiert werden, um tatsächlich Referenzwerte in Bezug auf Schedulability und Rechenzeit bieten zu können. Darüber hinaus sind die entsprechenden Performanceoptimierungen relevant, um beispielsweise in einem zukünftigen Vergleich exakter Lösungsverfahren (z.B. ggü. SMT) die Leistungsfähigkeit ILP-basierter Scheduler angemessen zu repräsentieren.

6.2 Getroffene Annahmen bezüglich Schedulingergebnissen

Wie in Kapitel 3.2 erläutert, müssen über die grundsätzlichen Schedulingbedingungen (d.h. Routing, Pfadscheduling, Ressourcen- und Anwendungsbedingungen) hinaus weitere Einschränkungen bezüglich der erstellten Schedules getroffen werden. Bei den in diesem Kapitel formulierten Modellen werden daher die folgenden Annahmen getroffen:

- Die *Granularität* beim Erstellen und Lösen von ILP-Modellen beträgt durchgängig $1\,\mu s$. Das bedeutet, dass sämtliche Konstanten und Variablen, die Zeitwerte innerhalb der ILP-Modelle repräsentieren, in diese Zeiteinheit umgerechnet werden. Diese Auflösung ist in der Literatur üblich (vgl. [27, 31]) und ausreichend zur Modellierung von Szenarien mit Linkbandbreiten bis $1\,\mathrm{Gbit/s}$. Aktuelle Messungen aus TSN-Testbeds zeigen, dass diese Auflösung mit heutigen Implementierungen hardwareseitig erzielbar ist, softwareseitige Einschränkungen (z.B. durch den Netzwerkstack des Betriebssystems) jedoch zu Abweichungen in den Sendezeitpunkten

führen können (vgl. [50, 45]). Die erstellten Modelle sind jedoch nicht auf diese Auflösung beschränkt und können auch mit abweichender Auflösung genutzt werden.

- *Jitter* innerhalb der Hyperperiode wird grundsätzlich ausgeschlossen. Zyklische Wiederholungen der Zeitschlitze eines Stream sind also strikt äquidistant. Dieses Vorgehen entspricht frühen Lösungsverfahren aus der Literatur [3] und bietet zudem den Vorteil, dass sich die Modellierung vereinfacht, da für einen Stream keine abweichenden Sendezeitschlitze für dessen zyklische Wiederholungen innerhalb der Hyperperiode modelliert werden müssen. Darüber hinaus bietet diese Modellierung die beste Kompatibilität zu realen Anwendungen, da diese nicht in allen Fällen Jitter erlauben. Zukünftige Modellerweiterungen zum Zulassen von begrenztem Jitter sind möglich.
- Als *Queuingmodell* wird zunächst davon ausgegangen, dass unbegrenztes Reordering innerhalb von Bridges möglich ist. Der Vorteil ist hierbei, dass es sich um das Modell mit dem größtmöglichen Lösungsraum handelt, welches zudem die Modellierung vereinfacht, da keine Bedingungen für die Sende- und Empfangsreihenfolge von Frames an den Bridges formuliert werden müssen. Weitere Queueingmodelle können durch Zusatzbedingungen modelliert werden, was in Kapitel 7.2 durchgeführt wird.
- Da in dieser Arbeit ausschließlich von Ethernet als Layer-2-Protokoll ausgegangen wird, gilt durchgängig $pre := 7\mathrm{B}$, $sfd := 1\mathrm{B}$ und $ifg := 12\mathrm{B}$. Dies ist beispielsweise für Ermittlung der Verzögerungszeiten d_{km}^{fwd}, d_{km}^{rcv} und Zeitschlitzlängen sl_{km} nach Kapitel 3.2 relevant.

6.3 Vorverarbeitung und Definition von Konstanten

Vor der ILP-Modellbildung sollen hier zunächst einige Hilfsausdrücke eingeführt werden, welche im Rahmen der Modellbildung eingesetzt werden. Die entsprechenden Größen werden aus den in Kapitel 3.1 spezifizierten Eingangsdaten abgeleitet und dienen der kompakteren Beschreibung der ILP-Modelle und zugehörigen Modellvarianten.

6.3.1 Verzögerungszeiten und Zeitschlitzlängen

Die für das Scheduling relevanten Größen für Zeitschlitzlänge, Bridgeverzögerung und Empfangsverzögerung wurden bereits in Kapitel 3.2 definiert. Aus diesen werden hier

$$d_{km}^{fwd\prime} := D\left(d_{km}^{fwd}\right) \qquad \text{und} \qquad sl'_{km} := D\left(sl_{km}\right)$$

abgeleitet, wobei zunächst $D(x) := x$ gilt und die Funktion erst im späteren Verlauf angepasst wird. Bei sl'_{km}, $d_{km}^{fwd\prime}$ und d_{km}^{rcv} handelt es sich im Sinne der ILP-Modelle um Konstanten, da sich die konkreten Werte vor der Modellbildung berechnen lassen.

6.3.2 Selektion von Routen (Routenvorverarbeitung)

Vor der Bildung von Variablen und Randbedingungen des ILP findet eine Selektion der möglichen Routen der Streams statt. Dies wurde in [E12] unter der Bezeichnung *route preprocessing* vorgeschlagen.

$$
\begin{aligned}
f_1 &:= (\mathrm{n7}, \{\mathrm{n4}, \mathrm{n6}\}, 50\,\mu\mathrm{s}, 1000\,\mathrm{B}, 50\,\mu\mathrm{s}) \\
V_1^{br} &:= \{\mathrm{n0}, \mathrm{n1}, \mathrm{n2}, \mathrm{n3}\} \\
E_1 &:= \big\{(\mathrm{n1}, \mathrm{n0}), (\mathrm{n3}, \mathrm{n1}), \\
&\qquad (\mathrm{n3}, \mathrm{n2}), (\mathrm{n2}, \mathrm{n0}), \\
&\qquad (\mathrm{n0}, \mathrm{n2}), (\mathrm{n0}, \mathrm{n4}), \\
&\qquad (\mathrm{n7}, \mathrm{n3}), (\mathrm{n2}, \mathrm{n6})\big\}
\end{aligned}
$$

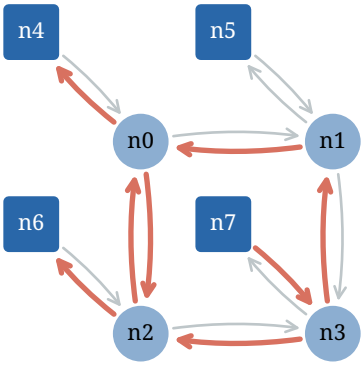


Abbildung 6.1: Reduzierte Kantenmenge $E_1 \subseteq \mathcal{E}$ des Beispielstreams mit ID 1, resultierend aus der Routenvorverarbeitung. Die Knotenmenge V_1^{br} enthält dagegen aufgrund der gegebenen Routingoptionen weiterhin alle Bridges der Topologie.

In diesem Schritt wird jedem Stream k eine Knotenuntermenge $V_k \subseteq \mathcal{V}$ und eine Kantenuntermenge $E_k \subseteq \mathcal{E}$ zugewiesen. Diese Untermengen enthalten alle Knoten bzw. Kanten, die während des Planungsprozesses zur Erstellung einer validen Route für den Stream k zur Verfügung stehen sollen. Im einfachsten Fall sind diese Untermengen jeweils die Vereinigung aller Knoten bzw. Kanten aus allen einfachen, schleifenfreien Pfaden, die zwischen Sender und Empfängern des Streams existieren. Bei dieser Definition von V_k und E_k besteht somit keine Einschränkung des Lösungsraums. Diese Form der Routenvorverarbeitung kommt in dieser Arbeit standardmäßig zum Einsatz, sofern nichts Anderes spezifiziert wurde, und wurde auch in [E12] angewendet. Im Vergleich zur vollständigen Topologie $(\mathcal{V}, \mathcal{E})$ können die so definierten Untermengen dennoch bedeutend kleiner sein, da für einen konkreten Stream k viele Knoten und Kanten der Topologie in keiner validen Route enthalten sind. Abbildung 6.1 zeigt das Ergebnis der Routenvorverarbeitung für einen Stream, wobei die streamspezifische Kantenmenge E_k auf die Hälfte der Kanten aus $\mathcal{E}$ reduziert werden konnte. Da ein Großteil der Entscheidungsvariablen und Randbedingungen des im Folgenden präsentierten ILP *für jede Kante* bzw. *für jeden Knoten*, die bzw. der durch Stream k nutzbar ist, definiert wird, hat dieser Vorverarbeitungsschritt bedeutenden Einfluss auf die Modellgröße.

Aufbauend auf diesen aus der Routenvorverarbeitung hervorgehenden Definitionen sollen an dieser Stelle weitere Knoten- und Kantenuntermengen definiert werden, von denen bei der Modellbildung regelmäßig Gebrauch gemacht wird. Die Knotenmenge $V_k^{br} := V_k \setminus (\{f_k.src\} \cup f_k.dsts)$ umfasst alle für k nutzbaren Bridge-Knoten. Basierend auf E_k und einem gegebenen Knoten i werden außerdem die Kantenmengen

$$E_{ki}^{\rightarrow} := \{m \in E_k \mid m_0 = i\} \qquad \text{und} \qquad E_{ki}^{\leftarrow} := \{m \in E_k \mid m_1 = i\}$$

definiert, welche die für k nutzbaren und von i aus- bzw. eingehenden Kanten enthalten. Da die Routenvorverarbeitung vor der ILP-Modellbildung ausgeführt wird, sind die resultierenden Mengen im Sinne des ILP ebenfalls konstant.

An dieser Stelle sei angemerkt, dass sich in Anhang A eine Übersicht über alle in der Problemformalisierung und ILP-Modellierung häufig genutzten mathematischen Symbole befindet, welche als „Lesehilfe“ für dieses und die folgenden Kapitel dienen soll.

6.4 ILP-basiertes *Joint Routing and Scheduling*

Das erste, ausschließlich für Unicaststreams geeignete und hier daher als *UC* bezeichnete ILP-Modell wurde 2017 in [E12] veröffentlicht. Wie bereits in Kapitel 6.1 ausführlich erläutert, lag die Zielsetzung bei der Entwicklung einer flexiblen Referenzlösung für TT-RSP, welche sich gegenüber früher veröffentlichten ILP/SMT-basierten Lösungen durch die gemeinsame Modellierung von Routing und Scheduling abhebt (JRaS, von *Joint Routing and Scheduling*). Entsprechend dieser neuen Integration des Routings lag der Fokus der damaligen Evaluation beim Vergleich mit Lösungsverfahren mit separaten Schritten für Routing und Scheduling (SRaS, von *Separate Routing and Scheduling*).

In diesem Kapitel soll das Modell *UC* vorgestellt und der Vergleich zwischen SRaS und JRaS aufgegriffen werden. Gegenüber der ursprünglichen Veröffentlichung wird das Modell einschließlich später entwickelter Verbesserungen vorgestellt. Diese umfassen:

- Adaption an heterogene Netze: Bridge- und Linkeigenschaften werden der Topologie entnommen (siehe e_m, v_i in Kapitel 3.1). In [E12] wurden noch homogene Netzwerke angenommen.
- Auswahl verschiedener ILP-Optimierungsziele. In [E12] wurde ausschließlich mit einem Optimierungsziel bezüglich Streamlatenzen gearbeitet.
- Diverse Modelloptionen zur Verbesserung der Performance.

Der zum Abschluss des Kapitels erneut durchgeführte Vergleich von SRaS und JRaS berücksichtigt sowohl die Modellverbesserungen als auch die Benchmarkingszenarien, welche einen bedeutend höheren Testumfang bieten als die ursprüngliche Evaluation.

6.4.1 Definition von ILP-Entscheidungsvariablen und Hilfsausdrücken

Entsprechend der Problemstellung von TT-RSP muss ein zugehöriges ILP-Modell die Routenfindung sowie den Sendezeitplan durch Entscheidungsvariablen abbilden, für die der ILP-Solver dann eine gültige Belegung unter gegebenen Randbedingungen finden muss.

Bezüglich des Routings existiert für jeden Stream k und jeden Link $m \in E_k$ eine binäre Pfadvariable $\mathfrak{p}_{km}$, wobei $\mathfrak{p}_{km} = 1$ in einer ILP-Lösung anzeigt, dass k über m geroutet wird. Das vollständige Routing eines Verkehrsmusters ist in der ILP-Lösung also an allen auf 1 gesetzten Pfadvariablen ablesbar. Der zugewiesene Sendezeitpunkt des Streams k wird für jeden Link $m \in E_k$ durch den linearen Ausdruck $\boldsymbol{t}_{km}^{\boldsymbol{abs}} := \mathfrak{v}_{km} \cdot f_k.ct + \mathfrak{t}_{km}$ bestimmt. Darin gibt die ganzzahlige Entscheidungsvariable $0 \leq \mathfrak{v}_{km}$ den Offset des Sendezeitpunkts als Vielfaches der Streamzykluszeit an,[3] während die ganzzahlige, beschränkte Entscheidungsvariable $0 \leq \mathfrak{t}_{km} \leq f_k.ct - 1$ den genauen Sendezeitpunkt innerhalb der Zykluszeit festlegt. Diese zweiteilige Modellierung des Sendezeitpunkts wird hier gewählt, da die Modellierung der Konfliktfreiheit von Zeitschlitzen eine Projektion des Sendezeitpunkts in den Wertebereich $[0, f_k.ct - 1]$ erfordert, welche somit direkt durch $\mathfrak{t}_{km}$ zur Verfügung steht.[4] Eine ILP-Lösung muss allen laut der Pfadvariablen genutzten Links auch valide Sendezeitpunkte zuweisen, während Sendezeitpunkte auf ungenutzten Links nicht von Interesse sind. Nach dem

[3] auf dem ersten Link eines Pfades (d.h. $m_0 = f_k.src$) gilt außerdem $\mathfrak{v}_{km} \leq 0$, sodass die Variable stets 0 wird

[4] eine Modulo-Operation auf ILP-Variablen kann nicht als Teil einer Randbedingung angegeben werden

Lösen des ILP entspricht die Startzeit eines zugewiesenen Zeitschlitzes dem Wert von $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$, während dessen Ende sich aus $\boldsymbol{t}^{\boldsymbol{abs}}_{km} + sl_{km}$ ergibt. Wie an dieser Stelle eingeführt, sind auch im Folgenden alle Entscheidungsvariablen des ILP durch Frakturschrift $(\mathfrak{p}, \mathfrak{t}, \mathfrak{o})$ hervorgehoben und alle linearen Ausdrücke, die Entscheidungsvariablen enthalten, durch Fettschrift.

An dieser Stelle soll außerdem ein linearer Ausdruck eingeführt werden, der die Ende-zu-Ende-Latenz eines Streams k mit Sender $i \coloneqq f_k.src$ zu einem Empfänger $j \in f_k.dsts$ erfasst und im Folgenden zur einfacheren Darstellung einiger Randbedingungen genutzt wird.

$$\boldsymbol{lt}_{kj} \coloneqq \sum_{m \in E^{\leftarrow}_{kj}} \left(\boldsymbol{t}^{\boldsymbol{abs}}_{km} + d^{rcv}_{km}\right) - \sum_{m \in E^{\rightarrow}_{ki}} \boldsymbol{t}^{\boldsymbol{abs}}_{km}$$

Der gegebene Ausdruck bildet die Ende-zu-Ende-Latenz nur für den Spezialfall korrekt ab, in dem Endgeräte (d.h. Sender und Empfänger) mit nur einem Link an die Netzwerktopologie angebunden sind. In diesem Fall bestehen die beiden Summen jeweils aus nur einem Element, sodass die erste Summe den Empfangszeitpunkt des vollständigen Frames am Empfänger j und die zweite Summe den Sendebeginn am Sender i wiedergibt. Eine Modellerweiterung für Topologien mit mehrfach verbundenen Endgeräten ist trivial, wurde jedoch im Rahmen dieser Arbeit nicht vorgenommen, da alle getesteten Szenarien dem genannten Spezialfall angehören.

Es sei angemerkt, dass lineare Ausdrücke wie $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$, $\boldsymbol{lt}_{kj}$ in dieser Arbeit nur der besseren Lesbarkeit von Bedingungen dienen. Sie stellen kein besonderes Element im Sinne der ILP-Formulierung dar. In der Implementierung der ILP-Modelle werden die linearen Ausdrücke während der Modellbildung entsprechend ihrer Definition ersetzt, sodass der ILP-Solver keinerlei Kenntnis über sie besitzt.

6.4.2 Randbedingungen des Unicastmodells *UC*

Die Randbedingungen des ILP-Modells werden hier entsprechend der in Kapitel 3.2 festgelegten grundlegenden Anforderungen an einen gültigen Schedule in Routingbedingungen, Ressourcenbedingungen, Pfadscheduling, und Anwendungsbedingungen unterteilt. Da für jeden Stream im Verkehrsmuster die gleichen Randbedingungen bezüglich Routing und Scheduling gelten, werden die entsprechenden Gleichungen hierfür in der Regel $\forall k \in F$ spezifiziert. Da Quantoren kein gültiger Bestandteil eines ILP sind, werden $\forall$-Quantoren in entsprechenden Implementierungen in Form von Schleifen ausgerollt, um eine Vielzahl von Instanzen der jeweiligen Randbedingung zu erzeugen.

6.4.2.1 Routingbedingungen

Eine Route wird hier nach (6.1) initiiert, indem am Sender genau ein ausgehender Link mehr aktiv[5] sein muss, als eingehende Links aktiv sind. An allen Bridgeknoten müssen dagegen laut (6.2) immer gleich viele eingehende und ausgehende Links aktiv sein, was einer einfachen Weiterleitungsbedingung entspricht. Die Bridgeknoten agieren also weder als Quelle noch als Senke für den Stream.

$$\forall k \in F,\ i \coloneqq f_k.src \qquad \sum_{m \in E^{\rightarrow}_{ki}} \mathfrak{p}_{km} - \sum_{m \in E^{\leftarrow}_{ki}} \mathfrak{p}_{km} = 1 \tag{6.1}$$

[5] „aktiv“ meint hier und im Folgenden „in der Route des Streams k enthalten“

$$\forall k \in F, \forall i \in V_k^{br} \qquad \sum_{m \in E_{ki}^{\rightarrow}} \mathfrak{p}_{km} - \sum_{m \in E_{ki}^{\leftarrow}} \mathfrak{p}_{km} = 0 \tag{6.2}$$

$$\forall k \in F, \forall i \in V_k \qquad \sum_{m \in E_{ki}^{\rightarrow}} \mathfrak{p}_{km} \leq 1 \tag{6.3}$$

Da (6.2) zulässt, dass eine Route zusätzliche unerwünschte Schleifen enthält, werden diese hier durch (6.3) verhindert, indem die Anzahl aktiver ausgehender Links an jedem Knoten auf eins beschränkt wird. Der Empfänger eines Streams ist durch die Routingbedingungen nur implizit gegeben, da dieser als einziger Knoten nicht durch (6.2) zur Fortsetzung der Route gezwungen ist.

6.4.2.2 Pfadscheduling

Entlang der aktiven Route eines Streams muss für jeden einzelnen Link ein Sendezeitpunkt festgelegt werden. Da Routing und Scheduling im hier vorliegenden Ansatz nicht in separaten Schritten, sondern durch ein einziges ILP-Modell gelöst werden sollen, ist das Routing zum Zeitpunkt der Modellbildung noch unbekannt. Daher müssen die Schedulingbedingungen die zeitlichen Abhängigkeiten für jedes mögliche Routing ausdrücken.

Im Modell *UC* wird hierfür zunächst der Sendezeitpunkt ungenutzter Links (d.h. $\mathfrak{p}_{km} = 0$) durch (6.4) auf 0 gesetzt (laut Beschränkung der Variablen in Kapitel 6.4.1 kann $\boldsymbol{t}_{km}^{\boldsymbol{abs}}$ nicht negativ werden). Aktive Links werden dagegen bei hinreichend großer Konstante $M_{6.4}$ nicht durch (6.4) im Sendezeitpunkt eingeschränkt. Eine sinnvolle Wahl solcher Konstanten ist nicht trivial und wird daher separat in Kapitel 9.2 diskutiert. Aufgrund der Datenabhängigkeit zwischen aufeinanderfolgenden Links muss der Sendezeitpunkt auf dem aktiven ausgehenden Link einer Bridge außerdem entsprechend der involvierten Bridge- und Leitungsverzögerungen gegenüber dem Sendezeitpunkt des genutzten eingehenden Links verschoben sein. Diese Datenabhängigkeit wird in (6.5) in einer Gleichung pro Bridgeknoten modelliert.

$$\forall k \in F, \forall m \in E_k \qquad \boldsymbol{t}_{km}^{\boldsymbol{abs}} \leq M_{6.4} \cdot \mathfrak{p}_{km} \tag{6.4}$$

$$\forall k \in F, \forall i \in V_k^{br} \qquad \sum_{m \in E_{ki}^{\rightarrow}} \boldsymbol{t}_{km}^{\boldsymbol{abs}} - \sum_{m \in E_{ki}^{\leftarrow}} \boldsymbol{t}_{km}^{\boldsymbol{abs}} \geq \sum_{m \in E_{ki}^{\leftarrow}} d_{km}^{fwd\prime} \cdot \mathfrak{p}_{km} \tag{6.5}$$

Hierbei wird ausgenutzt, dass sich aufgrund der anderen bereits aufgestellten Bedingungen die Sendezeitpunkte der eingehenden und ausgehenden Links eines Knotens in jeweils einer Summe zusammenfassen lassen: Nach Berücksichtigung von (6.1)–(6.4) wird in jeder Summenformel in (6.5) immer nur ein Element ungleich Null. Auf der linken Seite der Gleichung drückt die erste Summe somit den Sendezeitpunkt auf dem aktiven ausgehenden Link der Bridge i aus (ungenutzte Links haben Sendezeitpunkt 0 nach (6.4)), während die zweite Summe den Sendezeitpunkt des genutzten eingehenden Links bestimmt, und die rechte Seite die durch genau diesen eingehenden Link verursachte Verzögerung beinhaltet. Abschließend sei angemerkt, dass (6.5) für einen überhaupt nicht in der Route des Streams enthaltenen Knoten stets erfüllt ist, da alle Summenformeln Null ergeben.

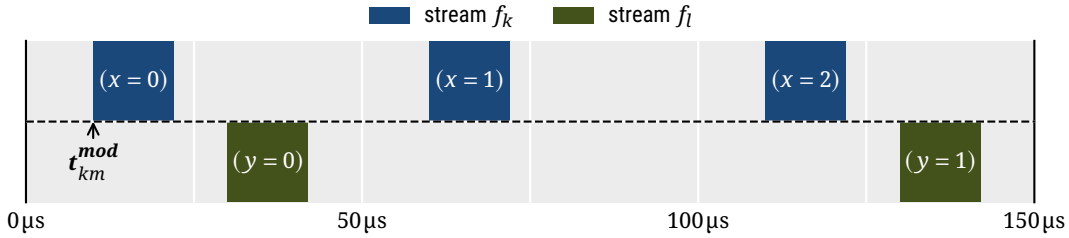


Abbildung 6.2: Zeitschlitzwiederholungen zweier Streams k, l mit $f_k.ct := 50\,\mu s$, $f_l.ct := 100\,\mu s$ auf einem Link m. Konfliktfreiheit muss zwischen allen Zeitschlitzpaaren von k und l innerhalb der gemeinsamen Hyperperiode $hp_{kl} := 100\,\mu s$ bis hin zur Wiederholung $(x, y) := (2, 1)$ gelten.

6.4.2.3 Anwendungsbedingungen

Die Einhaltung der in den Streamspezifikationen angegebenen maximal zulässigen Latenzen wird in (6.6) gefordert.

$$\forall k \in F, \forall j \in f_k.dsts \qquad \boldsymbol{lt}_{kj} \;\leq\; f_k.ml \tag{6.6}$$

Dabei wird auf den bereits definierten linearen Ausdruck $\boldsymbol{lt}_{kj}$ zurückgegriffen, der die Latenz von Stream k zum Empfänger j angibt und auf Basis der Sendezeitpunkte an Sender und Empfänger definiert ist.

6.4.2.4 Ressourcenbedingungen

Vergleichsweise komplex ist dagegen die Formulierung der Konfliktfreiheit von Zeitschlitzen, welche zwischen allen Streams, auf allen Kanten und für alle zyklischen Wiederholungen der Zeitschlitze gelten muss. Formal lassen sich die relevanten Kombinationen von Streams, Kanten und Zeitschlitzwiederholungen als Menge von Tupeln

$$RC \;:=\; \left\{(k, l, m, x, y) \in F \times F \times \mathcal{E} \times \mathbb{N}^0 \times \mathbb{N}^0 \;\middle|\; k < l,\, m \in E_{kl}^{\cap},\, x \leq \frac{hp_{kl}}{f_k.ct},\, y \leq \frac{hp_{kl}}{f_l.ct}\right\}$$

ausdrücken. In RC (von *resource constraints*) werden zunächst alle Kombinationen von Streams (k, l) ohne Permutationen (entspricht $k < l$)[6] gebildet. Für diese Streams (k, l) muss ein ILP-Modell die Konfliktfreiheit auf allen durch beide Streams nutzbaren Kanten $E_{kl}^{\cap} := E_k \cap E_l$ sicherstellen. Auf jeder dieser gemeinsamen Kanten sind hierfür paarweise alle Zeitschlitzwiederholungen zu betrachten, welche in der Hyperperiode $hp_{kl} := \mathrm{lcm}(f_k.ct, f_l.ct)$ der beiden Streams liegen.[7] Diese Zeitschlitzwiederholungen der Streams k und l werden beginnend bei Null durchnummeriert, was in der Definition von RC durch x und y erfasst ist (siehe auch Abb. 6.2). Als Startwert für die Zeitschlitzwiederholungen muss außerdem die Projektion des jeweiligen Sendezeitpunkts $\boldsymbol{t}_{km}^{\boldsymbol{abs}}$ in den Wertebereich $[0, f_k.ct - 1]$ bekannt sein, für die hier die Bezeichnung $\boldsymbol{t}_{km}^{\boldsymbol{mod}}$ eingeführt wird (siehe Abb. 6.2). Nach der Definition der ILP-Variablen aus Kapitel 6.4.1 gilt hier zunächst $\boldsymbol{t}_{km}^{\boldsymbol{mod}} := \mathrm{t}_{km}$. In den im Rahmen dieser Arbeit entwickelten ILP-Modellen wird die Konfliktfreiheit für jedes in RC

[6] an dieser Stelle sei angenommen, dass für F eine Ordnungsrelation definiert ist

[7] lcm ist eine Funktion zur Bestimmung des kleinsten gemeinsamen Vielfachen

referenzierte Paar von Zeitschlitzen durch eine Entweder-oder-Bedingung beschrieben: Entweder muss sich das Zeitschlitzende von Stream k vor dem Beginn des Zeitschlitzes von l befinden oder umgekehrt, was den Bedingungen (6.7) bzw. (6.8) entspricht. Diese Bedingungen werden auf Basis der Mengen $RC^{k..l}$ bzw. $RC^{l..k}$ erzeugt, für die hier zunächst $RC^{k..l} := RC$ und $RC^{l..k} := RC$ gilt.

$$\forall (k,l,m,x,y) \in RC^{k..l} \quad \begin{aligned} &(\boldsymbol{t}_{lm}^{\boldsymbol{mod}} + y \cdot f_l.ct) \; - \; (\boldsymbol{t}_{km}^{\boldsymbol{mod}} + x \cdot f_k.ct) \\ \geq \; &sl'_{km} \; - \; M_{6.7} \cdot (3 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm}) \end{aligned} \tag{6.7}$$

$$\forall (k,l,m,x,y) \in RC^{l..k} \quad \begin{aligned} &(\boldsymbol{t}_{km}^{\boldsymbol{mod}} + x \cdot f_k.ct) \; - \; (\boldsymbol{t}_{lm}^{\boldsymbol{mod}} + y \cdot f_l.ct) \\ \geq \; &sl'_{lm} \; - \; M_{6.8} \cdot (2 + \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm}) \end{aligned} \tag{6.8}$$

Die beiden Gleichungen sind für ein konkretes $(k,l,m,x,y) \in RC$ immer gemeinsam zu betrachten. Die rechtsseitigen M-Terme modellieren hier sowohl die Abhängigkeit der vorliegenden Ressourcenbedingungen vom Routing als auch die Darstellung einer Entweder-oder-Bedingung als ILP. Nutzt einer der Streams (k,l) den Link m laut Belegung der Routingvariablen nicht, so werden beide Bedingungen durch Subtraktion einer großen Konstante auf der rechten Seite der Gleichung trivial erfüllt. Sind dagegen beide Streams über m geroutet, so legt der Wert der binären Hilfsvariable $\mathfrak{a}_{klmxy}$ fest, welche der beiden Gleichungen aktiv ist, während die andere durch Subtraktion einer großen Konstanten trivial erfüllt wird. Für $\mathfrak{a}_{klmxy} = 1$ ist beispielsweise (6.7) aktiv. Die resultierende Bedingung besagt dann, dass die x-te Zeitschlitzwiederholung von Stream k enden muss, bevor der y-te Zeitschlitz von Stream l beginnt.

Die bis hierher eingeführten Entscheidungsvariablen und Randbedingungen bilden ein vollständiges, zum Lösen von TT-RSP ausreichendes ILP-Modell. Ein solches vollständiges Modell wird im Folgenden auch als *Basismodell* bezeichnet und trägt einen eindeutigen Namen (hier *UC*).

6.4.3 ILP-Optimierungsziele

Wie in Kapitel 5 erläutert, kann in einem ILP-Modell ein Optimierungsziel angegeben werden. Dieses beschreibt eine Gütefunktion, hinsichtlich der ein Solver gezielt nach der bestmöglichen Lösung suchen soll. Das bisher gezeigte Modell *UC* wurde in [E12] ausschließlich mit einem Optimierungsziel bezüglich Streamlatenzen veröffentlicht und evaluiert. Seitdem haben Folgearbeiten [E14, E15] den bedeutenden Performanceeinfluss der Optimierungsziele gezeigt. Dieser Einfluss soll hier daher für den Vergleich zwischen JRaS und SRaS aufgegriffen werden. Dementsprechend werden an dieser Stelle die Optimierung der Streamlatenzen sowie drei weitere Optimierungsziele vorgestellt.

6.4.3.1 Optimierung der Streamlatenzen

Die ursprüngliche Optimierung der Streamlatenzen ist insbesondere durch Anwendungen aus der Regelungstechnik motiviert. Soll eine Regelschleife derart realisiert werden, dass die Kommunikation zwischen Sensoren, Aktoren und Controller über ein Echtzeitnetzwerk erfolgt, so ist die Latenz der Netzwerkkommunikation ausschlaggebend für die Stabilität und Regelgüte der Regelschleife. Während die Mindestanforderung einer Anwendung (z.B. Stabilitätskriterium) hier bereits durch

$f_k.ml$ spezifiziert werden sollte, kann durch eine Optimierung der Streamlatenzen über diese Mindestanforderung hinaus typischerweise eine bessere Regelgüte erreicht werden (vgl. [20]), was sich beispielsweise in einer geringeren Abweichung einer Regelgröße vom Sollwert ausdrückt.

Optimierungsziel *LT* Auf Basis des bereits definierten Ausdrucks für Streamlatenzen kann das gewünschte Ziel als

$$\text{Minimize} \sum_{k \in F} \sum_{j \in f_k.dsts} \boldsymbol{lt}_{kj}$$

formuliert werden. Aufgrund der separaten Betrachtung aller Empfänger eines Multicaststreams führt *LT* für Unicast- und Multicaststreams gleichermaßen zu latenzoptimierten Ergebnissen.

6.4.3.2 Optimierung von Pfadlängen und Netzwerkauslastung

Ein weiteres intuitives Optimierungsziel von TT-RSP ist die Erstellung von Schedules mit geringer Netzwerkauslastung, da in diesem Fall mehr Ressourcen für weitere Streams aus nicht zeitgesteuerten Verkehrsklassen übrig bleiben. Ein entsprechende ILP-Formulierung wurde in [E14] veröffentlicht.

Optimierungsziel *SP* Eine geringe Netzwerkauslastung kann im einfachsten Fall durch Nutzung von möglichst wenigen Netzwerklinks für jeden Stream erreicht werden, was als

$$\text{Minimize} \sum_{k \in F} \sum_{m \in E_k} \mathfrak{p}_{km}$$

formuliert werden kann. Die genaue Interpretation dieses Ziels hängt von weiteren Faktoren ab. Für homogene Netzwerke und Unicaststreams bedeutet die Minimierung der Anzahl genutzter Kanten, dass gleichermaßen kürzeste Pfade im Sinne der Hopanzahl und Latenz (daher der Name *SP*, von *Shortest Paths*) sowie die minimale durchschnittliche Linkauslastung gesucht werden. Für Multicaststreams entspricht das Ziel weiterhin einer Optimierung der Linkauslastung, erzeugt aber im Allgemeinen keine kürzesten Pfade. Für inhomogene Netzwerke wird dagegen auch die Optimierung der Linkauslastung verfehlt, da in diesem Fall eine passende Gewichtung der Pfadvariablen notwendig wäre. Dies wäre problemlos umsetzbar (siehe Anhang C.1), wurde jedoch im Rahmen der Arbeit nicht berücksichtigt, da ausschließlich mit homogenen Netzwerken getestet wurde.

6.4.3.3 Hierarchische Optimierung

Ein weiterer Aspekt, der bei der Formulierung von Optimierungszielen berücksichtigt werden sollte, ist, dass das gesetzte Ziel die Suchrichtung des ILP-Solvers beeinflusst und somit auch ein Einfluss darauf besteht, wie schnell zu Beginn des Optimierungsprozesses überhaupt Lösungen gefunden werden. In [E14] wurde gezeigt, dass ein ILP-Modell für TT-RSP bei Verwendung des Ziels *SP* in bestimmten Fällen wesentlich schneller initiale Lösungen findet als bei Verwendung von *LT*. Problematisch am Optimierungsziel *SP* ist jedoch, dass der ILP-Solver hierbei keine latenzoptimierten Sendezeitpunkte auswählt, sodass trotz kurzer Pfade hohe Latenzen durch Buffering entstehen können. Eine sinnvolle Strategie ist es daher, den Optimierungsprozess mit einem Ziel zu starten, das das Finden initialer Lösungen begünstigt (hier *SP*) und erst später auf ein anderes Optimierungsziel (hier *LT*) umzuschalten, wobei die bereits gefundene Lösung als Startpunkt für die weitere Optimierung genutzt wird. Ein entsprechendes Vorgehen wird von ILP-Solvern wie Gurobi und CPLEX

in Form von *hierarchischen Optimierungszielen* unterstützt und in [E14] bezüglich der Pfad- und Latenzoptimierung für TT-RSP vorgeschlagen.

Optimierungsziel *SPLT* Bei Nutzung von *SPLT* wird der Optimierungsprozess mit dem Ziel *SP* gestartet und nach dem Finden einer hinreichend guten Lösung die Optimierung mit dem Ziel *LT* fortgesetzt. Standardmäßig wird das Ziel *SP* dabei bis zum Finden der optimalen Lösung optimiert, bevor zum Optimierungsziel *LT* gewechselt wird. Bei der Optimierung von *LT* werden dann außerdem nur Lösungen berücksichtigt, die keine Verschlechterung bezüglich des bereits optimierten Ziels *SP* darstellen.

6.4.3.4 Lösungsvorgang ohne Zielfunktion

Die gezeigten Optimierungsziele werden in dieser Arbeit wie schon in [E14] mit dem Lösen der entsprechenden ILP-Modelle ohne Zielfunktion verglichen.

Optimierungsziel *NONE* Mit *NONE* sind Modellvarianten gekennzeichnet, bei denen im ILP überhaupt kein Optimierungsziel gesetzt wurde. Der Suchvorgang wird abgebrochen, sobald eine gültige Lösung bezüglich der Randbedingungen gefunden wurde.

6.4.3.5 Namensgebung und Abbruchbedingungen

Das in einem ILP-Modell angewendete Optimierungsziel wird in dieser Arbeit zusätzlich zum Modellnamen als Subskript genannt, z.B. UC_{NONE}. Mit Ausnahme von *NONE* können die Optimierungsziele außerdem mit einer Optimality Gap (siehe Kapitel 5) genutzt werden, was zu einem Abbruch der Optimierung beim Erreichen einer definierten Lösungsgüte führt. Die Gap wird bei der Namensgebung als Suffix des Optimierungsziels gesetzt. Das Optimierungsziel *LT50* bedeutet beispielsweise, dass eine Gap von 50% auf Optimierungsziel *LT* angewendet wurde. Der Optimierungsvorgang wird dann abgebrochen, sobald eine gefundene Lösung bezüglich der Zielfunktion einen Wert erreicht, der weniger als das Doppelte der optimalen Lösung beträgt. Bei einem hierarchischen Ziel wie *SPLT* gilt die Gap für beide Ziele, sodass nach Erreichen der Gap bezüglich *SP* auf das Ziel *LT* gewechselt wird und dieses ebenfalls nur bis zur gewünschten Gap optimiert wird.

6.4.4 Modelloptionen *bpl*, *ia* und *rcr*

Neben den zuvor gezeigten alternativen Optimierungszielen wurde das in [E12] veröffentlichte ILP-Modell *UC* im Rahmen von Folgearbeiten auch um Modellvarianten erweitert, die der Verbesserung der Performance dienen. Diese werden hier als *Modelloptionen* bezeichnet, da es sich um Modifikationen des Basismodells handelt, die in der Implementierung an- und ausgeschaltet werden können. Einige der wesentlichen Verbesserungen sollen hier in die Evaluation von *UC* sowie den folgenden Vergleich mit SRaS-Lösungsverfahren einfließen. Diese erst später (in Bezug auf die Veröffentlichung von *UC*) entwickelten Verbesserungen werden hier aus folgenden Gründen aufgegriffen:

- Verdeutlichung erzielter Performancefortschritte seit Veröffentlichung von [E12]
- Potentieller Einfluss der Verbesserungen auf den Vergleich zu SRaS-Lösungsverfahren

- Aufgrund des seit der ursprünglichen Veröffentlichung deutlich gesteigerten Testumfangs (Benchmarkingszenarien) sollen möglichst viele der Vergleiche aus Rechenzeitgründen auf performanten Modellvarianten basieren und nicht auf prinzipiell obsoleten Modellen

6.4.4.1 Begrenzung von Pfadlängen (*bpl*)

Die erste Modelloption betrifft die Routenvorverarbeitung, bei der im Basismodell davon ausgegangen wird, dass jeder Stream k jeden Pfad nutzen darf (siehe Kapitel 6.3). Die bei der Modellbildung eingesetzten Knoten- und Kantenmengen V_k bzw. E_k sind gegenüber der Netzwerktopologie $(\mathcal{V}, \mathcal{E})$ somit nur um solche Knoten und Kanten reduziert, die in keinem gültigen Pfad von k vorkommen. Um in großen Topologien mit vielen alternativen Pfaden die Modellgröße und damit einhergehend die Rechenzeit zu reduzieren, wurde eine Option zur Begrenzung der Pfadlängen entwickelt.

Modelloption *bpl* Bei dieser Modelloption werden die für einen Stream bei der Modellbildung zu berücksichtigenden Pfade bezüglich der Länge begrenzt (*bounded path length*, *bpl*). Dabei wird für einen Stream k mit Sender $i := f_k.src$ und Empfänger $j \in f_k.dsts$ zunächst die Pfadlänge pl_{ij}^{sp} des kürzesten Pfades (bzgl. Anzahl genutzter Netzwerklinks) ermittelt. Bezüglich der erlaubten Pfadlänge wird außerdem ein relativer Cutoff $cut^{pl,rel}$ sowie ein absoluter Cutoff $cut^{pl,abs}$ festgelegt. Um bei der ILP-Modellbildung berücksichtigt zu werden muss die Länge pl eines Pfades von i nach j dann

$$pl \leq cut^{pl,rel} \cdot pl_{ij}^{sp} \qquad \text{und} \qquad pl \leq cut^{pl,abs}$$

erfüllen. Eine sinnvolle Wahl von $cut^{pl,rel}$ und $cut^{pl,abs}$ hängt dabei im Wesentlichen von der Topologie ab und kann beispielsweise als zusätzliche Topologieinformation zur Verfügung gestellt werden. In der Implementierung werden die relevanten Pfade mit Hilfe einer Funktion der Graphenbibliothek `NetworkX` ermittelt, welche alle einfachen Pfade von i nach j unter einem gegebenem Cutoff zurückliefert. Aus diesen Pfaden werden die Knoten- und Kantenmengen V_k und E_k des Streams k gebildet. Im Falle von Multicaststreams wird die Pfadberechnung für jeden Empfänger einzeln ausgeführt und die Ergebnisse in V_k und E_k zusammengeführt, ohne dass die relevanten Daten pro Empfänger erhalten bleiben. Kommt ein Sender/Empfänger-Paar im Verkehrsmuster mehrfach vor, so wird das Ergebnis wiederverwendet und die Pfadberechnung nicht erneut ausgeführt. Sofern einer der Cutoffs nicht in den Topologiedaten angegeben wurde, wird die entsprechende Bedingung ignoriert. Weitere Implementierungsdetails sollen hier nicht diskutiert werden, da der Performanceeinfluss in der Regel gering ist.

Im Rahmen der Benchmarkingszenarien wurden für $ring$ überhaupt keine Cutoffs definiert, sodass stets beide möglichen Pfade als Routingoption zur Verfügung stehen. Für $mesh$ wurde $cut^{pl,rel} := 3$ und ein von der Topologiegröße abhängiger absoluter Cutoffs genutzt, während für $fattree$ die Cutoffs $cut^{pl,rel} := 1$ und $cut^{pl,abs} := 6$ definiert wurden. Für alle Topologien des Benchmarkings sind die Cutoffs in den Topologiedaten in [E18] enthalten. Die Modelloption *bpl* zählt zu den wenigen Modellvarianten in dieser Arbeit, die nicht den vollständigen Lösungsraum von TT-RSP erhalten und zählt somit zu den heuristischen Modellanpassungen (vgl. Kapitel 6.1.2). Diese Einschränkung basiert auf der intuitiven Idee, dass gute Lösungen von TT-RSP in der Regel nicht auf Basis von Routen gebildet werden können, die bedeutend länger und somit gleichermaßen ressourcenbelastender sind, als der kürzeste Pfad.

6.4.4.2 Ganzzahlige Randbedingungen (*ia*)

Die nächste Option befasst sich damit, dass in Schedulingbedingungen wie z.B. (6.5) die Differenz zwischen ganzzahligen Sendezeitpunkten mit nicht-ganzzahligen Verzögerungszeiten verglichen wird. In solchen Fällen lässt sich ohne Ausschluss gültiger Lösungen eine strengere Randbedingung formulieren, indem die betrachteten Verzögerungszeiten aufgerundet werden.

Modelloption *ia* Bei der Modelloption *ia* (von *integer alignment*) wird die in der Vorverarbeitung (siehe Kapitel 6.3) definierte Funktion $D(x) = x$ durch $D(x) = \lceil x \rceil$ ersetzt. Dementsprechend werden alle in der Vorverarbeitung ermittelten Zeitschlitzlängen sl'_{km} und Verzögerungszeiten $d_{km}^{fwd\prime}$ auf Ganzzahlwerte aufgerundet. Dies führt in den Gleichungen (6.5), (6.7) und (6.8) zu strengeren Randbedingungen.

Die so entstehenden strengeren Randbedingungen führen nicht zu einem Verlust gültiger Ganzzahl-Lösungen des ILP, sondern verkleinern lediglich den LP-Lösungsraum seitens der Schedulingvariablen. Hiermit kann die Modelloption den Lösungsvorgang eines ILP-Solvers beschleunigen, da die im Branching zu lösenden LP-Relaxationen (d.h. Aufhebung der Ganzzahlbedingung von Variablen) stärker beschränkt werden. Insbesondere kann diese zusätzliche Beschränkung dazu führen, dass beim Lösen der LP-Relaxationen seltener nicht-ganzzahlige Lösungen bezüglich der Schedulingvariablen gefunden werden und die damit einhergehenden Branchingschritte wegfallen (vgl. Kapitel 5.2).

6.4.4.3 Reduktion von Ressourcenbedingungen (*rcr*)

Abschließend sei vor der Evaluation von *UC* eine Modelloption genannt, die die Anzahl modellierter Ressourcenbedingungen (Konfliktfreiheit von Zeitschlitzen) bedeutend reduziert. Bei genauer Betrachtung von (6.7) und (6.8) fällt auf, dass die Gleichungen auch für Zeitschlitzwiederholungen (x, y) erzeugt werden, für die aufgrund anderer Randbedingungen gar kein Konflikt existieren kann. Hierzu seien beispielsweise zwei Streams (k, l) mit $f_k.ct := 50\,\mu s$ und $f_l.ct := 100\,\mu s$ angenommen, sodass (x, y) laut Definition von *RC* die Werte $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$, $(2, 0)$ und $(2, 1)$ annehmen kann. Die aus diesen (x, y)-Werten resultierenden Wertebereiche für den Beginn der Zeitschlitze von k bzw. l sind in Abbildung 6.3 gezeigt. Darin erkennbar sind vier (x, y)-Kombinationen, die das Weglassen von Randbedingungen und Entscheidungsvariablen aus (6.7) und (6.8) erlauben:

(a) Die Bedingungen für $(x, y) := (2, 1)$ sind redundant zu $(0, 0)$, da es sich um eine zyklische Wiederholung nach einer Hyperperiode handelt.

(b) Für $(0, 1)$ kann aufgrund der Wertebereiche keine Überlappung der Zeitschlitze von k und l entstehen, sodass die entsprechenden Randbedingungen obsolet sind.

(c) Für $(2, 0)$ kann der Zeitschlitz von l nur vor dem von k liegen. Die zugehörige Hilfsvariable $\mathfrak{a}_{klmxy}$ muss also den Wert 0 annehmen, um (6.7) zu erfüllen, während (6.8) aktiv ist und die Konfliktfreiheit direkt an der Grenze der Wertebereiche von l und k sicherstellt. Zur Vereinfachung des ILP ließe sich also die Entscheidungsvariable $\mathfrak{a}_{klmxy}$ durch die Konstante 0 ersetzen und (6.7) weglassen.

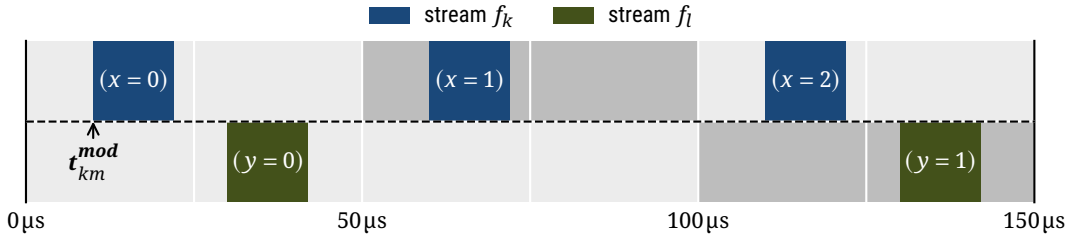


Abbildung 6.3: Zeitschlitzwiederholungen zweier Streams innerhalb der gemeinsamen Hyperperiode. Graustufen kennzeichnen den in UC durch die ILP-Variablen möglichen Wertebereich des jeweiligen Zeitschlitzanfangs. Der Zeitschlitzanfang von l für $y = 0$ ist beispielsweise auf das Intervall $[0\,\mu s, 99\,\mu s]$ begrenzt.

(d) Äquivalent zu (c) kann für $(1, 1)$ der Zeitschlitz von k nur vor dem von l liegen. Die zugehörige Hilfsvariable $\mathfrak{a}_{klmxy}$ muss also den Wert 1 annehmen, um (6.8) zu erfüllen, während (6.7) aktiv ist und die Konfliktfreiheit direkt an der Grenze der Wertebereiche von k und l sicherstellt. Zur Vereinfachung des ILP ließe sich also die Entscheidungsvariable $\mathfrak{a}_{klmxy}$ durch die Konstante 1 ersetzen und (6.8) weglassen.

Die entsprechende Optimierung des ILP wurde in [E14] vorgestellt und wird hier formal beschrieben.

Modelloption *rcr* Bei Aktivierung der Modelloption *rcr* (von *resource constraint reduction*) werden die beschriebenen Sonderfälle (a)–(d) berücksichtigt, um die Anzahl der in (6.7) und (6.8) erzeugten Randbedingungen zu verringern. Hierzu wird zunächst der boolesche Ausdruck

$$rc^{ex}_{klmxy} \coloneqq \left(\min(x \cdot f_k.ct/hp_{kl}, y \cdot f_l.ct/hp_{kl}) > 0\right) \vee \left((x+1) \cdot f_k.ct - 1 + sl'_{km} \leq y \cdot f_l.ct\right) \vee \left((y+1) \cdot f_l.ct - 1 + sl'_{lm} \leq x \cdot f_k.ct\right)$$

definiert. Dieser wird *wahr* für die in (a) und (b) beschriebenen Fälle. Entsprechend der Beschreibung von (c) und (d) wird außerdem die Definition von $\mathfrak{a}_{klmxy}$ angepasst, um die zum Zeitpunkt der Modellbildung bereits bekannte Zeitschlitzreihenfolge abzubilden.

$$\mathfrak{a}_{klmxy} \coloneqq \begin{cases} 0 & \left((y+1) \cdot f_l.ct - 1 \leq x \cdot f_k.ct + sl'_{km}\right) \\ 1 & \left((x+1) \cdot f_k.ct - 1 \leq y \cdot f_l.ct + sl'_{lm}\right) \\ \text{binäre Entscheidungsvariable} & \text{sonst (Zeitschlitzreihenfolge bei Modellbildung noch unbekannt)} \end{cases}$$

Auf dieser Basis werden abschließend $RC^{k..l}$ und $RC^{l..k}$ neu definiert.

$$RC^{k..l} \coloneqq \left\{(k, l, m, x, y) \in RC \mid \neg rc^{ex}_{klmxy},\ \mathfrak{a}_{klmxy} \neq 0\right\}$$

$$RC^{l..k} \coloneqq \left\{(k, l, m, x, y) \in RC \mid \neg rc^{ex}_{klmxy},\ \mathfrak{a}_{klmxy} \neq 1\right\}$$

Bei Verwendung der Option *rcr* werden weiterhin (6.7) und (6.8) zur ILP-Modellbildung eingesetzt, jedoch unter Berücksichtigung der hier gezeigten neuen Definition von $RC^{k..l}$, $RC^{l..k}$ und $\mathfrak{a}_{klmxy}$.

Die Modelloption *rcr* reduziert die Anzahl der Randbedingungen (6.7)/(6.8) und Entscheidungsvariablen $\mathfrak{a}_{klmxy}$, ohne dabei den modellierten Lösungsraum zu verändern. Die verringerte Modellgröße soll dabei zu kürzeren Rechenzeiten bzgl. der LP-Relaxationen führen, als auch zu weniger Branchingschritten aufgrund der reduzierten Binärvariablen.

6.4.4.4 Namensgebung

Wie an den drei bisher gezeigten Modelloptionen erkennbar, handelt es sich bei diesen um Modifikationen des Basismodells. Dementsprechend wird der Einsatz dieser Optionen als Superskript am Modellnamen gekennzeichnet. UC_{LT50}^{rcr} bezeichnet also einen Scheduler mit Basismodell *UC*, aktiver Modelloption *rcr* sowie dem Optimierungsziel *LT* mit einer Optimality Gap von $50\,\%$. Eine Auflistung und Kurzbeschreibung aller entworfenen Basismodelle, Modelloptionen und Optimierungsziele findet sich in Anhang B.

6.4.5 Evaluation von *UC* mit Modelloptionen und Optimierungszielen

Bevor weitere Modellvarianten erläutert werden, soll eine erste Evaluation von *UC* erfolgen und die Effektivität der bisher gezeigten Modelloptionen und Optimierungsziele bewertet werden. Aus Rechenzeitgründen können dabei hier und im weiteren Verlauf der Arbeit nicht alle möglichen Kombinationen von Modelloptionen und Optimierungszielen getestet werden. Daher erfolgt die Bewertung der Modelloptionen unter Verwendung von nur zwei Optimierungszielen, bevor anschließend alle vier Optimierungsziele unter Nutzung des verbesserten Modells (d.h. aktives *bpl*, *rcr* und *ia*) verglichen werden.

Alle hier und im Folgenden durchgeführten Evaluationen werden mit der nach Kapitel 4.3 konfigurierten Testplattform durchgeführt. Neben den Abbruchbedingungen des jeweiligen Optimierungsziels (siehe Kapitel 6.4.3) gilt somit auch hier ein Zeitlimit von $1200\mathrm{s}$ pro Probleminstanz. Wird ein Optimierungsvorgang durch das Zeitlimit abgebrochen, so wird im Allgemeinen die beste bis dahin gefundene Lösung vom ILP-Solver zurückgegeben. Das Zeitlimit bezieht sich dabei ausschließlich auf den ILP-Optimierungsprozess, sodass die Gesamtzeit inklusive Vorverarbeitung und Modellbildung geringfügig höher ausfallen kann. Die Vorverarbeitung beinhaltet in den hier betrachteten Modellvarianten nur die Routenberechnungen der Option *bpl* und ist in den Ergebnissen als *route pp* gekennzeichnet, während das Aufstellen aller Randbedingungen unter Nutzung der Python API von Gurobi als *modeling* aufgefasst wird. Der *optimize*-Schritt beinhaltet ausschließlich die Rechenzeit des Solvers Gurobi, unabhängig davon, ob tatsächlich eine Optimierung stattfindet (Ziele *SPLT25*/*LT25*) oder nur eine erste Lösung gesucht wird (Ziele *1SP*/*NONE*). Die Schritte *modeling* und *optimize* enthalten demnach auf der einen Seite die Rechenzeit des im Rahmen dieser Arbeit implementierten Python-Codes (*modeling*) und auf der anderen Seite die Rechenzeit des genutzten Solvers (*optimize*).

6.4.5.1 Evaluation der Modelloptionen

Bei der Bewertung der Modelloptionen *ia* und *rcr* kommen die Optimierungsziele *SPLT25* und *NONE* zum Einsatz, sodass die erzielten Fortschritte sowohl für optimierende als auch nicht-optimierende ILP-Modelle abgebildet werden. Die Wahl des Optimierungsziels für die optimierende Variante fiel auf *SPLT25*, da *SP* alleine grundsätzlich keine optimierten Streamlatenzen liefert, während *LT* laut Erkenntnissen aus [E14] in Kombination mit einem Multicastmodell eine schlechtere Performance als *SPLT* zeigte. Ob dies auf das hier getestete Unicastmodell zutrifft, wird im späteren Vergleich der

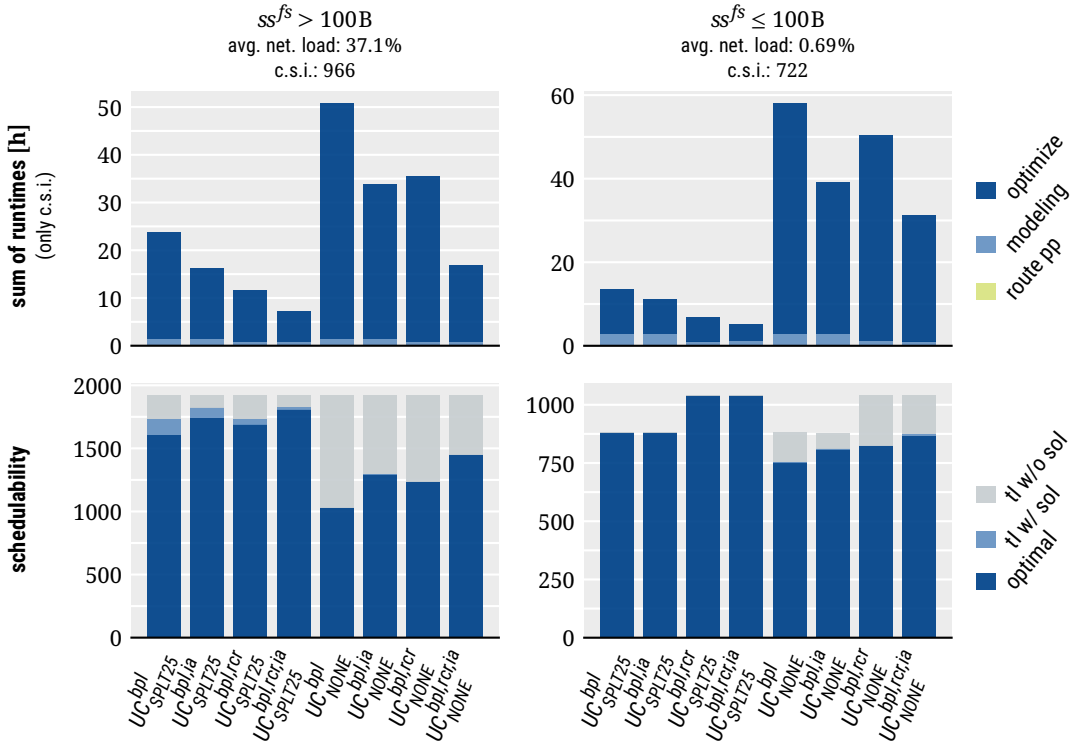


Abbildung 6.4: Schedulability und c.s.i.-Rechenzeiten für *UC* bei Aktivierung von *ia* und *rcr*.

Optimierungsziele noch untersucht. Die Optimality Gap von 25 % wurde gewählt, um einen sinnvollen Ausgleich zwischen Ergebnisgüte und Rechenzeit herzustellen, da [E14] ebenfalls gezeigt hat, dass eine Optimierung ohne Gap mit einem massiven Anstieg der Rechenzeit einhergeht. Der konkrete Wert von 25 % basiert jedoch nicht auf einer tiefergehenden Analyse dieses Tradeoffs. Da es in der Praxis vor allem vom konkreten Einsatzbereich des Schedulers abhängt, wie stark Rechenzeit oder Ergebnisgüte priorisiert werden sollten, soll dies hier nicht weiter untersucht werden.

Die Ergebnisse für die acht sich aus *ia*, *rcr*, *SPLT25* und *NONE* ergebenden ILP-Modellvarianten sind in Abbildung 6.4 gezeigt. Die Modelloption *bpl* ist dabei immer aktiviert: Ein Test ohne diese Modelloption wurde nicht durchgeführt, da die Routenvorverarbeitung ohne diese Option alle einfachen Pfade einer Topologie auflistet, was insbesondere in größeren Topologien des Benchmarkings problematisch sein kann. Des Weiteren sei angemerkt, dass für die ILP-Modelle ohne *rcr* nur 880 der 1040 Probleminstanzen mit niedriger Netzwerkauslastung ($ss^{fs} \leq 100\text{B}$) getestet wurden. Die fehlenden Probleminstanzen[8] wurden wegen zu hoher Speicherauslastung abgebrochen und sind daher im Ergebnisdiagramm *nicht* in den regulär ungelösten Instanzen (*tl w/o sol*) enthalten, sondern überhaupt nicht dargestellt.

Wie aufgrund der Funktionsweise der Modelloptionen *ia* und *rcr* zu erwarten, verbessern sich Schedulability und Rechenzeit in Kombination mit beiden Optimierungszielen signifikant bei Hinzuschalten der Optionen. Bezüglich der Schedulability ist der deutlichste Effekt für das Optimierungsziel

[8] Es fehlen: TC-TS/$ring$/$|\mathcal{V}^{es}|$:= 96, TC-TS/$mesh$/$|\mathcal{V}^{es}|$:= 95, TC-SSS/$ring$/$|F|$:= 111, TC-SSS/$mesh$/$|F|$:= 107 (d.h., Szenarien, die aufgrund von Topologiegröße und Streamanzahl zu besonders großen Modellen führen)

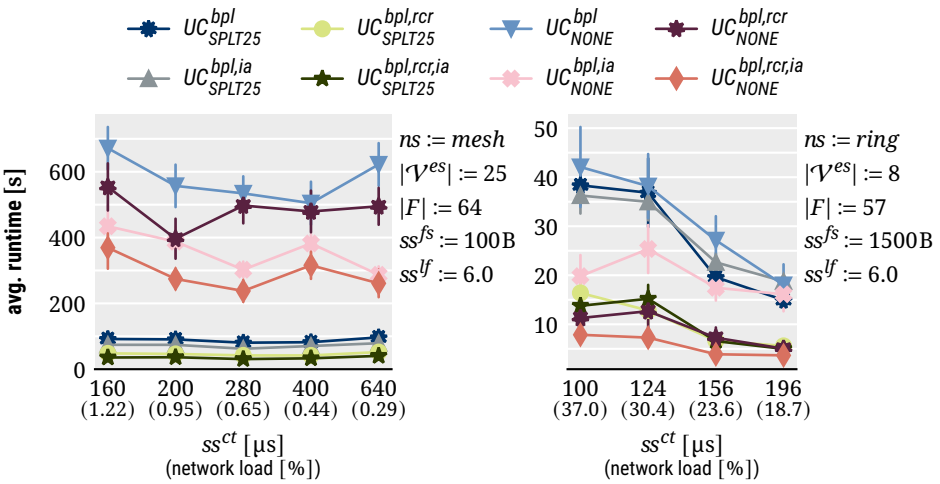


Abbildung 6.5: Ausschnitt der Rechenzeiten für TC-DS (links) und TC-L (rechts) bei Aktivierung von *ia* und *rcr*.

NONE und Szenarien mit hoher Netzwerkauslastung zu verzeichnen: $UC^{bpl,rcr,ia}_{NONE}$ konnte 421 Instanzen mehr lösen als UC^{bpl}_{NONE}. Die c.s.i.-Rechenzeit sinkt mit 69.8 % am stärksten für $UC^{bpl,rcr,ia}_{SPLT25}$ gegenüber UC^{bpl}_{SPLT25} im Falle der Szenarien mit hoher Last. Der individuelle Einfluss von *ia* bzw. *rcr* hängt dabei von den Eingangsdaten und dem genutzten Optimierungsziel ab. Abbildung 6.5 zeigt beispielsweise:

- In TC-DS profitiert UC^{bpl}_{SPLT25} stärker vom Hinzuschalten von *rcr* als von *ia*, während UC^{bpl}_{NONE} stärker von *ia* profitiert.
- In TC-L kehrt sich dieser Effekt für UC^{bpl}_{NONE} um, sodass hier auch für das Optimierungsziel *NONE* die Modelloption *rcr* einen größeren Effekt als *ia* zeigt.

Trotz dieser individuellen Unterschiede ist das Hinzuschalten beider Optionen über alle Testcases hinweg in keinem Fall von Nachteil. Daher werden diese im Folgenden grundsätzlich aktiviert.

6.4.5.2 Evaluation der Optimierungsziele

Für das verbesserte Modell (*bpl*, *rcr* und *ia* aktiv) sollen die in Kapitel 6.4.3 vorgestellten Optimierungsziele verglichen werden. Hinzu kommt dementsprechend das Ziel *LT*, welches im Gegensatz zu *SPLT* eine sofortige Optimierung der Latenzen ohne vorige Optimierung der Pfadlängen fordert. Es wird hier als *LT25* äquivalent zu *SPLT25* mit einer Optimality Gap von 25 % genutzt. Ein weiteres Optimierungsziel ist motiviert durch die auffälligen Ergebnisse des vorherigen Tests.

Auffällig ist (siehe Abb. 6.4), dass der optimierende Scheduler $UC^{bpl,rcr,ia}_{SPLT25}$ im Allgemeinen eine bessere Schedulability und Rechenzeit bietet als $UC^{bpl,rcr,ia}_{NONE}$. Dieses Verhalten ist nicht intuitiv erklärbar, da bei *NONE* nur bis zur ersten gefundenen Lösung gesucht wird, während der ILP-Solver bei *SPLT25* darüber hinaus nach besseren Lösungen bezüglich des Optimierungsziels sucht und dementsprechend auch eine deutlich bessere Ergebnisgüte aufweist. Das gleiche Verhalten wurde auch schon in [E15] beobachtet. Zurückzuführen ist dieses unerwartete Verhalten auf den Einfluss des Optimierungsziels auf die Suchstrategie des ILP-Solvers: Wie an den grundlegenden Lösungsverfahren für ILPs ersichtlich (vgl. Kapitel 5.2), verändern sich durch eine Anpassung des Optimierungsziels sowohl die

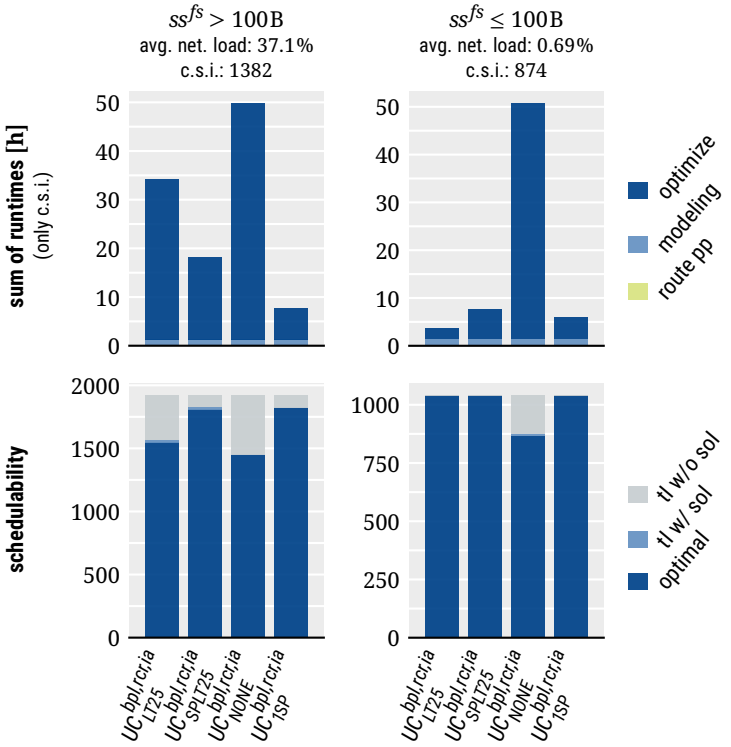


Abbildung 6.6: Schedulability und c.s.i.-Rechenzeiten für *UC* und vier Optimierungsziele.

gefundenen Lösungen der LP-Relaxationen als auch die damit verbundenen Branching-Variablen und die bevorzugte Branching-Richtung. Um eine schnelle, nicht-optimierende Lösungssuche abzubilden und gleichzeitig vom positiven Einfluss des Optimierungsziels zu profitieren, wird hier daher das Optimierungsziel *1SP* eingeführt. Dies entspricht der in Kapitel 6.4.3 definierten Optimierung von Pfadlängen (*SP*), wobei die Lösungssuche jedoch nach dem Auffinden der ersten gültigen Lösung abgebrochen wird. Da es sich bei dem für den ILP-Solver definierten Optimierungsziel um das gleiche Ziel handelt wie im ersten Optimierungsschritt von *SPLT25*, kann davon ausgegangen werden, dass grundsätzlich die gleiche Schedulability erzielt wird und außerdem durch den frühzeitigen Abbruch der Optimierung auch kürzere Rechenzeiten ermöglicht werden.

Die zusammengefassten Ergebnisse für die vier Optimierungsziele sind in Abbildung 6.6 zu sehen. Die wesentlichen Erkenntnisse sind:

- *1SP* erzielt wie erwartet die Schedulability von *SPLT25* bei geringerer Rechenzeit aufgrund des frühzeitigen Abbruchs der Optimierung.
- *1SP* bietet neben der besseren Schedulability gegenüber *NONE* auch eine um $84.5\,\%$ ($ss^{fs} > 100\text{B}$) bzw. $88.2\,\%$ ($ss^{fs} \leq 100\text{B}$) geringere c.s.i.-Rechenzeit.
- Bei hoher Netzwerkauslastung ($ss^{fs} > 100\text{B}$) hat *SPLT25* eine bessere Schedulability und eine fast halbierte c.s.i.-Rechenzeit gegenüber *LT25*.
- Bei niedriger Netzwerkauslastung ($ss^{fs} \leq 100\text{B}$) hat *SPLT25* mehr als die doppelte c.s.i.-Rechenzeit von *LT25*.

- *NONE* bietet im Allgemeinen eine schlechtere Schedulability und c.s.i.-Rechenzeit als alle anderen Optimierungsziele.

An dieser Stelle sei noch einmal auf die große Bedeutung der umfangreichen Benchmarkingszenarien verwiesen: Je nach betrachteten Eingangsdaten sind die Relationen zwischen *SPLT25* und *LT25* genau entgegengesetzt. Bei Betrachtung der einzelnen Testcases (siehe [E19]) lassen sich dementsprechend ebenfalls verschiedenste Relationen zwischen den beiden Optimierungszielen auffinden. Sowohl die Vorteile von *SPLT25* gegenüber *LT25* bei hoher Last als auch die allgemeinen Vorteile von *1SP* gegenüber *NONE* sind auf das veränderte Branching-Verhalten des ILP-Solvers aufgrund des primären Optimierungsziels *SP* zurückzuführen. Spezifischere Details (z.B. der exakte Branching-Baum) lassen sich dabei nicht angeben, da der genutzte ILP-Solver keinen Einblick in den Lösungsweg gewährt. Allgemein lässt sich jedoch sagen, dass *NONE* problematisch ist, da dem Solver keine sinnvolle Suchrichtung vorgegeben wird, während *SP* oder *LT* zu einer Bevorzugung von Bereichen des Lösungsraumes führen, die weniger aktive Links besitzen. Dies führt zu kurzen Routen und somit geringerer durchschnittlicher Netzwerkauslastung und zugleich weniger potentiellen Zeitschlitzkonflikten, was das Auffinden initialer und auch optimierter Lösungen begünstigt.

Ein Vergleich dieser Ergebnisse mit der ursprünglichen Veröffentlichung von *UC* in [E12] ist hier nicht möglich, da in dieser ausschließlich mit dem Ziel *LT*, ohne *rcr*/*ia*, und mit deutlich weniger Szenarien getestet wurde. Im Folgenden werden alle hier gezeigten Optimierungsziele grundsätzlich weitergenutzt, um auch für weitere ILP-Modellvarianten das mögliche Zusammenspiel zu analysieren. Sofern aus Rechenzeitgründen nicht alle Varianten getestet werden können, so werden im Folgenden bevorzugt *1SP* und *SPLT25* genutzt, um eine nicht-optimierende sowie eine optimierende Konfiguration abzubilden. Während die Wahl von *1SP* gegenüber *NONE* auf Basis der gezeigten Ergebnisse offensichtlich ist, wird *SPLT25* gegenüber *LT25* bevorzugt, da der Performancevorteil bei hoher Netzwerkauslastung im Kontext der genutzten Benchmarkingszenarien von größerer Bedeutung ist.

6.4.6 Modelloptionen für separates Routing (SRaS)

Eine wesentliche Zielsetzung der Veröffentlichung des ILP-Modells *UC* in 2017 war es, einen Vergleich des vorgestellten JRaS-Modells zu den bis dahin vorherrschenden SRaS-Lösungsverfahren bezüglich der Performanceindikatoren (Schedulability, Rechenzeit, Streamlatenzen) zu bieten. Dies soll hier unter Verwendung des durch *bpl*, *ia* und *rcr* verbesserten Modells und unter Verwendung der verschiedenen Optimierungsziele aufgegriffen werden. Hierfür müssen zunächst konkrete SRaS-Lösungsverfahren definiert werden, die zum Vergleich herangezogen werden sollen.

Für den Schedulinganteil eines SRaS-Verfahrens wurde in [E12] das ILP-Modell von *UC* auf ein äquivalentes ILP-Modell reduziert, das ausschließlich das Schedulingproblem widerspiegelt. Aus Gründen der Wartbarkeit der Implementierung wurde dagegen für den hier durchgeführten Vergleich kein solches dediziertes Modell erstellt, sondern eine strengere Routenvorverarbeitung in Kombination mit dem optimierten Modell $UC^{bpl,rcr,ia}$ verwendet. Für die Routenvorverarbeitung kommen hierbei verschiedene Verfahren zum Einsatz, die jedem Stream k bereits einen konkreten Pfad zuweisen. Die daraus hervorgehenden Knoten- und Kantenmengen V_k bzw. E_k enthalten also ausschließlich die Knoten und Kanten dieses Pfades. Wird nun auf Basis dieser Mengen das Modell

UC erstellt, so besitzen die Routingbedingungen nur eine einzige, triviale Lösung. Diese kann vom ILP-Solver im sogenannten *presolve* ermittelt und anschließend ein reduziertes Modell abgeleitet werden. Im Wesentlichen übernimmt hier also der ILP-Solver den in [E12] noch manuell durchgeführten Reduktionsschritt.

Im Routinganteil von SRaS kamen in [E12] zwei Varianten zum Einsatz:

- Ein einfaches *Shortest Path* (SP)-Routing auf Basis von Algorithmen der Graphenbibliothek `NetworkX` [74]. Sofern zwischen zwei Endgeräten mehrere Streams im Verkehrsmuster enthalten waren und die Topologie zwischen diesen außerdem mehrere gleich lange Pfade geboten hat, wurden die Streams außerdem gleichmäßig auf die verschiedenen Pfade verteilt.
- Ein ILP-basiertes *Load Balanced* (LB)-Routing, dass das Routing des gesamten Verkehrsmusters dahingehend optimiert, dass die höchste auftretende Linkauslastung eines beliebigen Links der Topologie minimal wird. Durch ein gewichtetes Optimierungsziel wurde sekundär außerdem wieder auf kürzeste Pfade optimiert.

Das SP-Routing sollte in der Schedulingphase geringe Streamlatenzen ermöglichen, wohingegen das LB-Routing eine bessere Schedulability auf Kosten der Streamlatenzen bieten sollte. Die beiden Ansätze sollen hier in verbesserter Form zum Einsatz kommen. Da es sich bei der hier vorliegenden Umsetzung der SRaS-Verfahren im Wesentlichen um eine spezielle Routenvorverarbeitung unter Weiternutzung des Basismodells *UC* handelt, werden die Verfahren hier als Modelloption beschrieben. Im Gegensatz zu [E12] wird hier bereits das SP-Routing auf Basis eines ILP-Modells realisiert.

> **Modelloption *r-nwsp-lu1*** Bei dieser Modelloption wird für das gesamte Verkehrsmuster ein Routing auf Basis eines ILP-Modells erstellt. Die einzelnen Pfade der Streams werden hinsichtlich der Streamlatenzen (d.h. kürzeste Pfade im Sinne der Latenz) optimiert. Die Streamlatenz wird dabei wie schon die ideale Latenz in Kapitel 4.2.2.2 ausschließlich anhand Stream-, Bridge- und Leitungsparametern (Framegröße, Linkbandbreite, usw.) ermittelt und berücksichtigt keinerlei Queuing durch Ressourcenkonflikte, daher die Namensgebung *nwsp* von *no-wait shortest path*. Das Routing aller Streams darf dabei allerdings keinen Link der Topologie zu mehr als 100 % der Linkbandbreite auslasten, da dies im nachfolgenden Schedulingschritt grundsätzlich nicht lösbar wäre (Namensgebung *lu1*, von *link utilization < 1*).

Die Berücksichtigung der Linkauslastung ist der wesentliche Unterschied zum SP-Routing aus [E12] und zugleich der Grund für den Einsatz eines ILP-Modells anstelle eines einfachen Shortest Path Algorithmus. Zum Vergleich mit dem SP-Routing aus [E12] wurde folgende weitere Variante genutzt.

> **Modelloption *r-nwsp*** Erstellung eines Routings mit dem gleichen Modell wie *r-nwsp-lu1*, jedoch ohne Überprüfung der Linkauslastung.

Es sei angemerkt, dass diese Modelloption nicht in jedem Fall zu den gleichen Ergebnissen wie das SP-Routing aus [E12] führt, da hier bei Vorhandensein mehrerer gleich langer Pfade zufällig ein Pfad ausgewählt wird. Abschließend wird das verbesserte LB-Routing beschrieben.

> **Modelloption *r-lb-nwsp*** Auch bei dieser Modelloption wird für das gesamte Verkehrsmuster ein Routing auf Basis eines ILP-Modells erstellt. Als primäres Optimierungsziel wird die höchste in der Topologie auftretende Linkauslastung minimiert (Namensgebung *lb*, von *load balanced*). In Form eines hierarchischen Optimierungsziels werden anschließend die Pfadlängen bezüglich der Streamlatenzen optimiert. Wie schon in *r-nwsp-lu1* berücksichtigen die Streamlatenzen hierbei kein Queuing.

Auch wenn die Modelloption *r-lb-nwsp* wie schon in [E12] ILP-basiert sowie mit den gleichen Optimierungszielen arbeitet, gibt es wesentliche Unterschiede. Zum einen wurde das ILP-Modell von ausschließlich unicastfähigen Routingbedingungen auf eine später entwickelte multicastfähige Formulierung abgeändert. Andererseits wurde das in [E12] genutzte gewichtete Optimierungsziel zur gleichzeitigen Optimierung von Pfadlängen und maximaler Linkauslastung durch ein zu diesem Zweck geeigneteres hierarchisches Optimierungsziel ersetzt (vgl. *SPLT* in Kapitel 6.4.3).

Die Benchmarkingergebnisse (Details im Folgekapitel) von *r-lb-nwsp* und *r-nwsp-lu1* zeigten das Problem auf, dass in einigen Szenarien Routen gewählt wurden, die die Latenzanforderungen einiger Streams bereits ohne Buffering durch Zeitschlitzkonflikte nicht erfüllen konnten und somit im Scheduling zu unlösbaren Modellen geführt haben. Infolge dessen wurden latenzsensitive Varianten dieser Routingverfahren entworfen.

Modelloption *r-nwsp-lu1-lta* Routingverfahren nach *r-nwsp-lu1* mit Zusatzbedingung zur Überprüfung gegebener Latenzanforderungen (Namensgebung *lta*, von *latency-aware*). Konkret wird hierbei für jede zugewiesene Route überprüft, ob diese zumindest ohne den Einfluss von Queueing die Latenzanforderung $f_k.ml$ des jeweiligen Streams erfüllen kann.

Modelloption *r-lb-nwsp-lta* Routingverfahren nach *r-lb-nwsp* mit Zusatzbedingung zur Überprüfung gegebener Latenzanforderungen (entsprechend *r-nwsp-lu1-lta*).

Alle fünf hier beschriebenen Routingverfahren wurden mithilfe geringfügiger Modifikation des gleichen ILP-Modells realisiert, welches in Anhang C.2 gezeigt ist. Auf eine direkte Nachbildung der Routingverfahren aus [E12] wurde verzichtet, da die Ergebnisgüte von *r-nwsp-lu1* und *r-lb-nwsp* gegenüber den dortigen Verfahren überlegen ist und die Rechenzeit des Routingschritts in anspruchsvollen Szenarien von geringer Relevanz ist.

6.4.7 Vergleich mit Lösungsverfahren mit separatem Routing

Die zuvor vorgestellten SRaS-Verfahren werden unter Nutzung der Optimierungsziele *1SP* und *SPLT25* verglichen, um sowohl den nicht-optimierenden als auch den optimierenden Fall abzubilden. Für die SRaS-Verfahren ist die in beiden Optimierungszielen enthaltene Zielfunktion bezüglich kurzer Pfade (*SP*) irrelevant. Dies wird hier ignoriert, da mit deren Nutzung grundsätzlich auch keine Nachteile verbunden sind. Außerdem kommt hier auch für die SRaS-Verfahren die Modelloption *bpl* zwecks Routenvorverarbeitung zum Einsatz. Dies bedeutet, dass bereits vor dem Routingschritt mögliche Pfade für jeden Stream vorselektiert werden und auch im separaten Routing nur diese zur Verfügung stehen. Dieses Vorgehen wurde gewählt, weil den zu vergleichenden JRaS- und SRaS-Verfahren somit grundsätzlich der gleiche Lösungsraum zur Verfügung steht und die Verfahren sich nur darin unterscheiden ob dieser ganzheitlich oder in zwei Schritten bearbeitet wird. Insgesamt ergibt sich somit der folgende Planungsablauf bzw. Kennzeichnung der Rechenzeiten in den Ergebnissen:

1. *route pp:* `NetworkX`-basierte Routenvorverarbeitung der Modelloption *bpl*, welche jedem Stream k eine Knoten- und Kantenmenge V_k bzw. E_k zuweist.
2. *routing:* Bei aktivierter SRaS-Option (Präfix *r-**) folgt der ILP-basierte Routingschritt auf Basis von V_k/E_k der vorigen Routenvorverarbeitung. Rechenzeiten für diesen Schritt enthalten

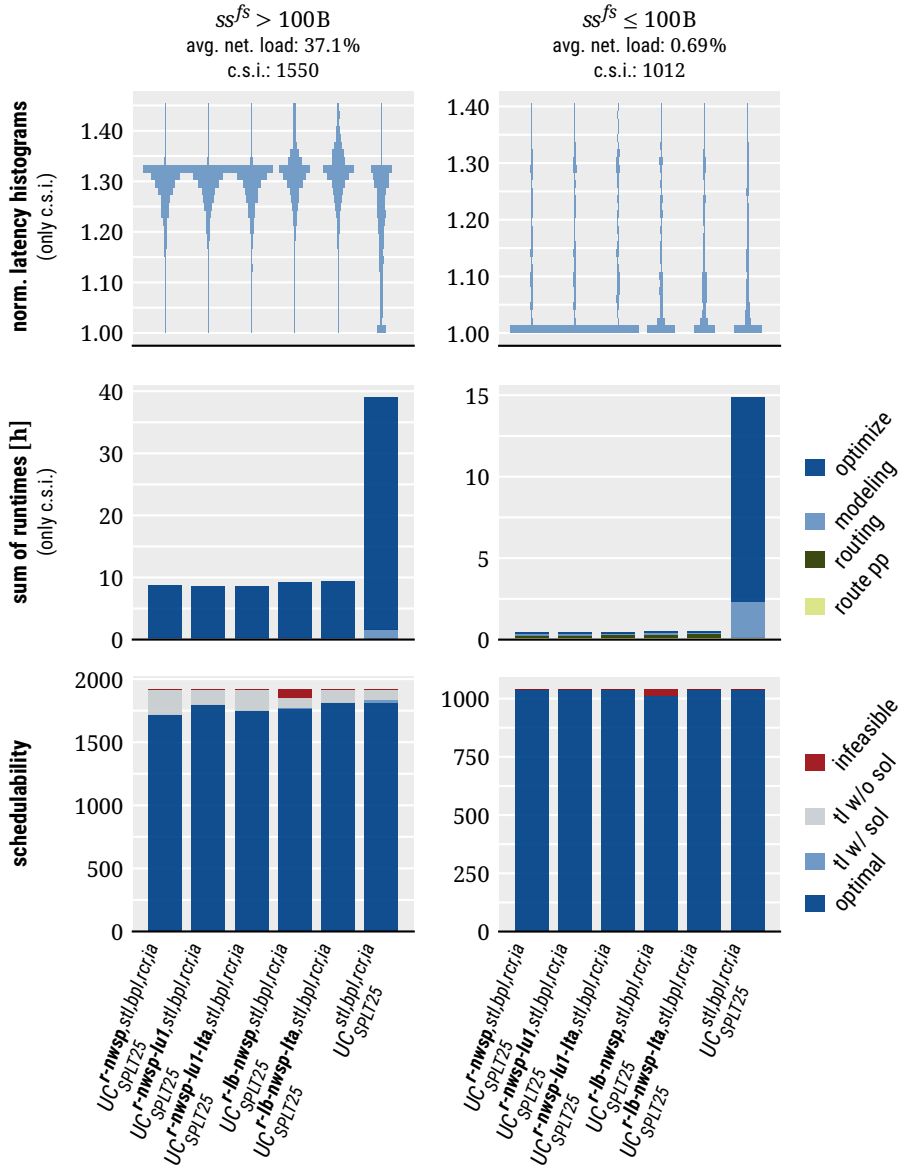


Abbildung 6.7: Schedulability, c.s.i.-Rechenzeiten und Latenzen für SRaS-Verfahren.

Modellbildung in der Python API und Lösen durch den Solver. In diesem Schritt werden neue Mengen V_k/E_k für jeden Stream k gebildet, welche nun nur noch eine Route enthalten.

3. *modeling:* Modellbildung des Scheduling-ILPs auf Basis des aktuellen V_k/E_k.
4. *optimize:* Lösen des Scheduling-ILPs durch den Solver.

Ausgeführt wurden die ILP-Modelle des Routingschritts außerdem stets mit einer Optimality Gap von 10 %, einem Zeitlimit von 600 s und dem Threadlimit des Schedulingmodells (6 Threads). Von einer gezielten Optimierung dieser Parameter wurde abgesehen, da die so erzielten Ergebnisse zeigen, dass der Routingschritt einen vergleichsweise geringen Anteil zur Gesamtlaufzeit der Scheduler beiträgt. Des Weiteren wurde der Mechanismus des Zeitlimits pro Probleminstanz dahingehend

verändert, dass das Zeitlimit von $1200\,\mathrm{s}$ nun auf den gesamten Lösungsvorgang angewendet wird, und nicht wie zuvor ausschließlich auf das Scheduling-ILP. Der neue Mechanismus wird durch die Modelloption *stl* (von *strict time limit*) gekennzeichnet. Die Ergebnisse für den optimierenden Fall (*SPLT25*) sind in Abbildung 6.7 zusammengefasst. Die wesentlichen Merkmale sind:

- JRaS ($UC^{stl,bpl,rcr,ia}$) besitzt im Falle hoher Netzwerkauslastung geringfügige Vorteile bei der Schedulability und bezüglich der erzielten Streamlatenzen.
- Vorteile von JRaS werden durch einen massiven Anstieg der Rechenzeiten erkauft, wobei der relative Anstieg gegenüber SRaS insbesondere bei niedriger Last höher ausfällt.
- Das LB-Routing besitzt erwartungsgemäß Nachteile bei den Streamlatenzen, zeigt in der latenzsensitiven Variante *r-lb-nwsp-lta* jedoch die beste Schedulability aller SRaS-Verfahren.
- Die Varianten des SP-Routings führen zu besseren Streamlatenzen als das LB-Routing, liegen jedoch gegenüber JRaS zurück.
- Ausgeprägte Nachteile bezüglich der Schedulability ergeben sich für das SP-Routing nur in der Modellvariante ohne Überprüfung der Linkauslastung (*r-nwsp*).

Obwohl es bezüglich des Gesamteindrucks Übereinstimmungen mit dem Vergleich der Verfahren in [E12] gibt, erlaubt das hier genutzte Benchmarking eine wesentlich detailliertere Analyse und zeigt neue Aspekte auf. Die hier gezeigte Zusammenfassung des Benchmarkings zeigt beispielsweise den für viele Szenarien massiven Rechenzeitanstieg für JRaS bei vergleichsweise geringen Vorteilen bezüglich der Schedulability. In [E12] waren die Vorteile bei der Schedulability dagegen wesentlich stärker ausgeprägt und vermittelten somit einen positiveren Gesamteindruck von JRaS. Beim hier durchgeführten Benchmarking finden sich vergleichbare Ergebnisse bei Betrachtung der einzelnen Testcases. So zeigt sich beispielsweise in TC-LL (Abb. 6.8), insbesondere für $ss^{lf} := 6.0$, eine Verschiebung zu Gunsten des JRaS-Ansatzes: Im Vergleich zur Zusammenfassung über alle Szenarien fällt hier der Rechenzeitnachteil kleiner und der Schedulabilityvorteil größer aus. Der vermittelte Eindruck entspricht hier den Ergebnissen aus [E12]. Dies verdeutlicht, dass die Ergebnisse in [E12] ohne das umfassende Benchmarking nur einen kleinen Ausschnitt des Performancevergleichs widerspiegeln. Eine weitere unerwartete Auffälligkeit im hier vorliegenden Benchmarking war zunächst die schlechtere Schedulability für das (nicht latenzsensitive) LB-Routing *r-lb-nwsp* im Vergleich zum SP-Routing, was im Widerspruch zu den Ergebnissen aus [E12] und auch der Zielsetzung dieses Routingverfahrens steht. Nachvollziehen lässt sich dieses Ergebnis ebenfalls anhand der Ergebnisse von TC-LL: Das LB-Routing führt bei niedrigen Latenzschranken der Streams ($ss^{lf} := 1.5$) zu nicht planbaren Routings, da in der Routingphase lange Pfade gewählt werden, für die anschließend kein Scheduling innerhalb der Latenzschranken möglich ist. Für höhere Latenzschranken erfüllt das LB-Routing dagegen die Erwartung einer besseren Schedulability gegenüber dem SP-Routing (*r-nwsp*/*r-nwsp-lu1*). Die Problematik bei niedrigen Latenzschranken war im Modell aus [E12] bereits vorhanden, hat sich dort aufgrund des geringeren Testumfangs jedoch nicht gezeigt. In den hier vorliegenden Modellen wurde das Problem durch das latenzsensitive Modell *r-lb-nwsp-lta* behoben. Der Einfluss der hier gezeigten Ergebnisse für JRaS- und SRaS-Verfahren auf die Weiterentwicklung der ILP-Modelle wird im folgenden Unterkapitel weiter diskutiert.

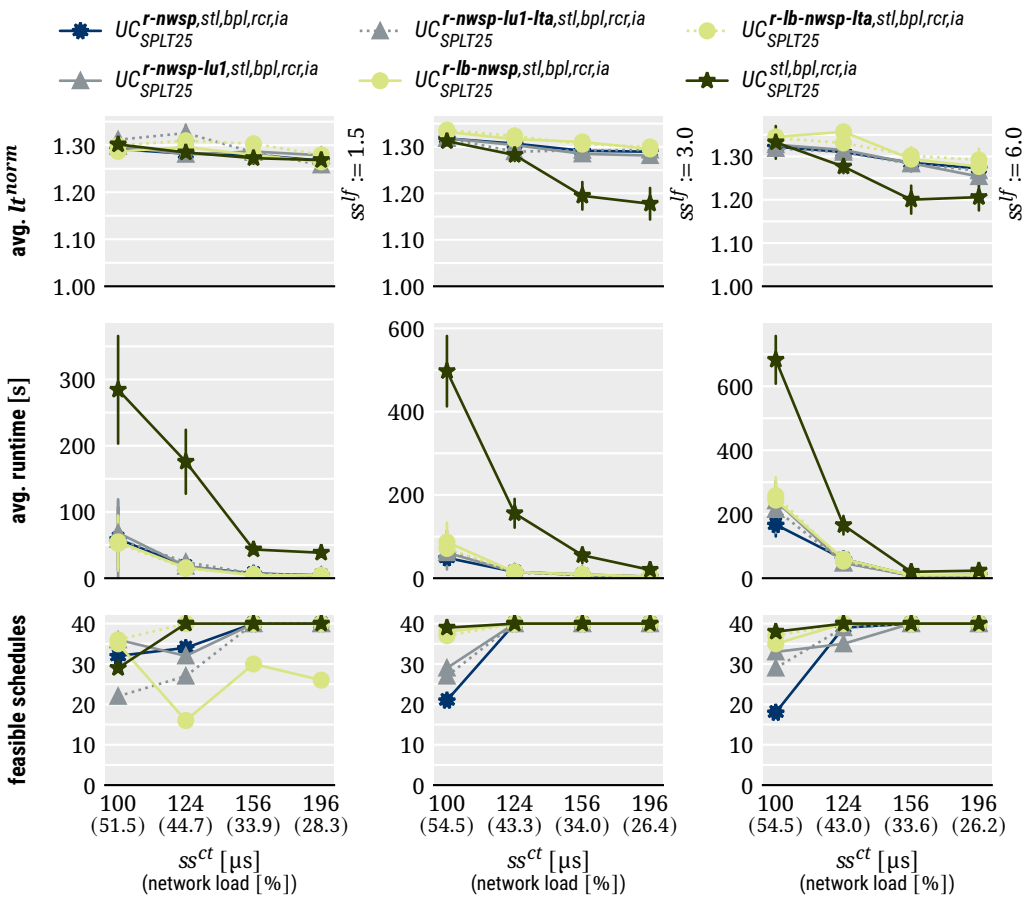


Abbildung 6.8: Ergebnisse für TC-LL auf Basis der Ringtopologie. Wiederholter Parametersweep von ss^{ct} für $ss^{lf} := 1.5$, $ss^{lf} := 3.0$ und $ss^{lf} := 6.0$ (v.l.n.r.). Konstante Szenarioparameter $ns := ring$, $|\mathcal{V}^{es}| := 8$, $|F| := 82$ und $ss^{fs} := 1500\text{B}$.

6.4.8 Zusammenfassung der Ergebnisse bezüglich *UC* und JRaS

In Kapitel 6.4 wurde das auf Unicast beschränkte ILP-Basismodell *UC* vorgestellt, welches das durch TT-RSP geforderte Routing und Scheduling in einem einzigen mathematischen Modell abbildet, sodass es durch einen ILP-Solver gemeinsam gelöst werden kann (JRaS, von Joint Routing and Scheduling). Das Modell wurde hier einschließlich der drei Modelloptionen *bpl*, *rcr* und *ia* und vier Optimierungsziele *NONE*, *1SP*, *SPLT25* und *LT25* analysiert. Alle drei Modelloptionen dienen dabei der Performancesteigerung des ILP-Modells, wobei dies bei *bpl* aufgrund der Routenselektion mit einer Einschränkung des Lösungsraumes einhergeht. Die Optionen *rcr* und *ia* erzielen dagegen bei gleichbleibendem Lösungsraum geringere Rechenzeiten. Die Rechenzeitreduktion erreicht dabei bis zu 69.8 % für die Benchmarkingszenarien mit hoher Netzwerkauslastung. Die Benchmarkingergebnisse zeigen außerdem für die beiden nicht-optimierenden Optimierungsziele (d.h. Abbruch bei erster Lösung) eine weitgehende Überlegenheit von *1SP* gegenüber *NONE*. Die beiden optimierenden Ziele *LT25* und *SPLT25* liefern dagegen beide gute Ergebnisse, wobei konkrete Vorteile für eines der Ziele von den jeweiligen Eingangsdaten abhängen. Insgesamt konnte somit entsprechend der

Zielsetzung aus Kapitel 6.1 eine Referenzlösung für TT-RSP erstellt und getestet werden. Die einzige Abweichung bezüglich der ursprünglich definierten Ziele besteht durch die Modelloption *bpl*, da es sich hierbei um eine Einschränkung des Lösungsraumes handelt. Prinzipiell wäre es jedoch möglich, das entwickelte Modell auf Kosten der Rechenzeit ohne diese Einschränkung zu betreiben.

Auf Basis des ILP-Modells *UC* wurde, wie schon in dessen ursprünglicher Veröffentlichung, ein Vergleich zu Verfahren durchgeführt, die Routing und Scheduling in zwei separaten Schritten lösen (SRaS, von Separate Routing and Scheduling). Für das Routing dieser SRaS-Verfahren wurde dabei entweder auf ein Shortest Path (SP)-Routing oder auf ein Load Balanced (LB)-Routing zurückgegriffen und das Scheduling weiterhin mit *UC* unter Nutzung der zuvor festgelegten Routen durchgeführt. Die Ergebnisse zeigen zwar erwartungsgemäß, dass das hier entworfene JRaS-Modell Vorteile bezüglich Schedulability und Ergebnisgüte (Streamlatenzen) erzielen kann, jedoch gehen diese insbesondere bei Szenarien mit geringer Netzwerklast mit einem unverhältnismäßigen Rechenzeitanstieg einher.

Die durchgeführten Vergleiche in diesem Kapitel bestätigen mehrfach die Bedeutung des großen Testumfangs des Benchmarkings. In vielen Fällen ändert sich die Bewertung verschiedener Verfahren in Abhängigkeit von den Eingangsdaten und erst die Analyse einzelner Testcases ermöglicht ein besseres Verständnis für das Verhalten bestimmter ILP-Modellvarianten.

Abschließend sollen an dieser Stelle die Entscheidungen bezüglich der weiteren Entwicklung der Lösungsverfahren für TT-RSP diskutiert werden. Grundsätzlich erforderlich ist die Weiterentwicklung des Lösungsverfahrens über die Fähigkeiten von *UC* hinaus, um den möglichen Einsatzbereich sowohl durch Performanceverbesserungen als auch die Ergänzung weiterer Features (Multicast, Queueingmodell von TSN) zu vergrößern. Die Weiternutzung der Modelloptionen *bpl*, *rcr* und *ia* steht dabei aufgrund der erzielten Ergebnisse ebenso wie die Weiternutzung der Benchmarkingszenarien außer Frage. Nicht eindeutig sind dagegen die Ergebnisse bezüglich JRaS und SRaS. Nach der Veröffentlichung von *UC* in 2017 wurde entschieden, den JRaS-Ansatz trotz der größeren Rechenzeiten weiterzuverfolgen, was zum damaligen Zeitpunkt folgendermaßen begründet wurde:

(a) Die teilweise signifikanten Vorteile bei der Schedulability und Vorteile bezüglich Ergebnisgüte überwiegen die Nachteile bei der Rechenzeit.

(b) Nachteile bei der Rechenzeit können zukünftig auch bei einem JRaS-Ansatz verringert werden, z.B. durch strengere Routenvorverarbeitung oder durch das inkrementelle Lösen des ILPs.

(c) Das ursprüngliche Ziel aus Kapitel 6.1, ein umfassendes Modell mit bestmöglicher Lösungsraumabdeckung und Optimalitätsinformation, sprach gegen die Reduktion auf einen SRaS-Ansatz.

Vor allem die positive Bewertung des Tradeoffs zwischen Schedulability und Rechenzeit (a) ist auf Basis der hier gezeigten Ergebnisse (Abb. 6.7) zunächst zweifelhaft. Nachvollziehbar wird die Entscheidung jedoch, wenn man berücksichtigt, dass die zum damaligen Zeitpunkt vorliegenden Ergebnisse aufgrund des geringeren Testumfangs den JRaS-Ansatz noch positiver dargestellt haben (vgl. Abb. 6.8, $ss^{lf} := 6.0$). Darüber hinaus ist die Argumentation hinsichtlich potentieller Performanceverbesserungen (b) von JRaS weiterhin zutreffend und wird insbesondere in Kapitel 7 im Detail ausgearbeitet. Das bedeutendste Entscheidungskriterium für den JRaS-Ansatz war jedoch (c), die Beibehaltung eines möglichst umfassenden Modells. Insbesondere die einfache Realisierung der SRaS-Verfahren durch eine einfache Reduktion des komplexeren JRaS-Modells hat gezeigt,

dass das komplexere Modell nicht nur zusätzliche Informationen liefern kann, sondern sich als Ausgangspunkt für die Entwicklung effizienter und bedarfsgerechter Heuristiken eignet. Die Möglichkeit, weitere Lösungsverfahren aus dem komplexen Modell ableiten zu können, macht dessen Weiterentwicklung entsprechend bedeutsam. Dementsprechend wird in der Weiterentwicklung der langsamere JRaS-Ansatz bevorzugt, wobei die Rechenzeitnachteile zukünftig durch eine strengere Routenvorverarbeitung oder inkrementelle Ansätze adressiert werden sollen.

6.5 Formulierung multicastfähiger ILP-Modelle für TT-RSP

Die im Kontext von *Smart Factories* und dem *Industrial Internet of Things (IIoT)* zunehmende Gerätekommunikation in Produktionsnetzwerken macht es erforderlich, Informationen an viele Geräte gleichzeitig durch Multicasts zu verteilen. Eine entsprechende Kommunikation durch mehrere Unicastverbindungen wäre in solchen Fällen zwar prinzipiell möglich, aber vergleichsweise ineffizient. Daher werden auch in den höheren Protokollschichten entsprechende Kommunikationsprofile definiert, wie z.B. OPC UA Publish/Subscribe [56]. In der Problemformulierung von TT-RSP in Kapitel 3 sind Multicaststreams dementsprechend ebenfalls bereits berücksichtigt. Vom bisher gezeigten ILP-Modell *UC* werden diese jedoch nicht unterstützt. Den Routing- und Schedulingbedingungen von *UC* lag die Annahme zugrunde, dass entlang der Route eines Streams keine Verzweigungen auftreten können und somit grundsätzlich an jeder Bridge maximal ein ausgehender Link aktiv ist. Da für Multicaststreams Verzweigungen entlang der Route benötigt werden, mussten die entsprechenden Bedingungen neu formuliert werden. Die entsprechenden ILP-Modelle wurden in [E14] veröffentlicht und werden hier zusammen mit zusätzlichen Modelloptionen vorgestellt und mit dem vergrößerten Testumfang des Benchmarkings analysiert.

6.5.1 Multicast-Basismodell *MCI*

Die entwickelten Multicastmodelle weisen einige grundlegende Überschneidungen mit dem bereits diskutierten Unicastmodell *UC* auf. Insbesondere die Vorverarbeitungsschritte und Entscheidungsvariablen entsprechen denen aus Kapitel 6.3 bzw. 6.4.1, sodass die dort eingeführten Symbole hier weiterhin gültig sind. Außerdem können die Anwendungsbedingungen (6.6), Ressourcenbedingungen (6.7)/(6.8), sowie alle bisher präsentierten Modelloptionen und Optimierungsziele weiterverwendet werden. Neu definiert werden müssen dagegen die Bedingungen für das Routing und Pfadscheduling, was hier im Kontext des ersten Multicast-Basismodells *MCI* erfolgen soll.

6.5.1.1 Routingbedingungen

Das Routing eines Streams wird in den multicastfähigen Modellen dadurch initiiert, dass laut (6.9) jeder Empfänger genau einen aktiven eingehenden Link benötigt.

$$\forall k \in F, \forall i \in f_k.dsts \qquad \sum_{m \in E_{ki}^{\leftarrow}} \mathfrak{p}_{km} = 1 \tag{6.9}$$

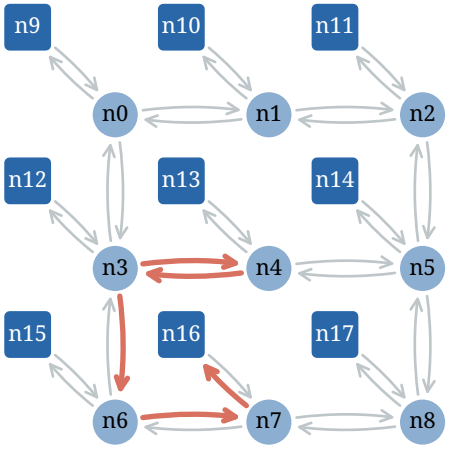


Abbildung 6.9: Ungültige, aber laut (6.10) zulässige Route für einen Stream von $\mathrm{n10}$ nach $\mathrm{n16}$.

Zur Fortsetzung der Route ausgehend von diesen aktiven Links an den Empfängern muss dann durch weitere Bedingungen gefordert werden, dass jeder aktive Link auch einen Vorgänger besitzt. Eine mögliche Gleichung hierfür ist (6.10), welche die Aktivierung von Ausgangslinks einer Bridge (u.a. hin zu einem Empfänger) nur bei mindestens einem aktiven Eingangslink erlaubt.

$$\forall k \in F, \forall i \in V_k^{br} \qquad \sum_{m \in E_{ki}^{\rightarrow}} \mathfrak{p}_{km} \;\leq\; M_{6.10} \cdot \sum_{m \in E_{ki}^{\leftarrow}} \mathfrak{p}_{km} \tag{6.10}$$

Die Konstante $M_{6.10}$ ist dabei so zu wählen, dass bereits bei einem aktiven Eingangslink keine Einschränkung bezüglich der Anzahl aktiver Ausgangslinks mehr existiert und somit beliebige Verzweigungen für Multicast möglich sind. Ein potentielles Problem von (6.10) liegt darin, dass zwei aktive Kanten, die ein Knotenpaar in entgegengesetzter Richtung verbinden, die Gleichung erfüllen, ohne zu einer sinnvollen Fortsetzung der Route zu führen (siehe Abb. 6.9). Eine alternative Modellierung wird daher in (6.11) vorgeschlagen.

$$\forall k \in F, \forall i \in V_k^{br}, \forall \vec{m} \in E_{ki}^{\rightarrow} \qquad \mathfrak{p}_{k\vec{m}} \;\leq \sum_{m \in E_{ki}^{\leftarrow} | m_0 \neq \vec{m}_1} \mathfrak{p}_{km} \tag{6.11}$$

Hier darf ein ausgehender Link $\vec{m}$ einer Bridge nur dann aktiv sein, wenn ein oder mehr eingehende Links aktiv sind, bei denen es sich nicht um die Gegenrichtung von $\vec{m}$ handelt. Auch hier können Verzweigungen an Bridgeknoten in beliebiger Weise auftreten, da ein aktiver eingehender Link an einer Bridge die Aktivierung von allen Ausgangslinks (außer der Gegenrichtung des eingehenden Links) ermöglicht. Da diese Gleichung jedoch nur separat für jeden Ausgangslink einer Bridge aufgestellt werden kann, ist die Gesamtanzahl benötigter Bedingungen hierbei höher als im Falle von (6.10). Grundsätzlich kann (6.10) oder (6.11) in einem ILP-Modell zum Einsatz kommen, allerdings müssen die potentiellen Schleifen zwischen benachbarten Knoten bei Nutzung von (6.10) durch weitere Bedingungen (beispielsweise Schedulingbedingungen) ausgeschlossen werden. Auf Kosten der höheren Anzahl von Routingbedingungen ist (6.11) dagegen alleine bereits ausreichend für ein korrektes Routing. Standardmäßig nutzen die in dieser Arbeit formulierten ILP-Modelle die Gleichung (6.10). Zusätzlich wird an dieser Stelle die Modelloption *rlp* eingeführt.

Modelloption *rlp* Falls die Option *rlp* (von *routing loop prevention*) bei der Erstellung eines ILP-Modells angegeben ist, wird (6.10) durch (6.11) ersetzt. Außerdem werden ggf. Maßnahmen zur Schleifenvermeidung zwischen Nachbarknoten deaktiviert, die nur für (6.10) notwendig sind. Dies wird im Rahmen der Schedulingbedingungen weiter erläutert.

Sowohl (6.10) als auch (6.11) werden nur für Bridgeknoten erstellt, sodass der Sender des Streams implizit gegeben ist, da dieser als einziger Knoten einen aktiven Ausgangslink besitzen kann, ohne dass ein Vorgänger gefordert ist.

6.5.1.2 Pfadscheduling

Entlang der gewählten Route eines Streams müssen die Sendezeitpunkte entsprechend der Link- und Bridgeverzögerungen ansteigen. Pro Stream konnte dies im Modell *UC* wie in (6.5) gezeigt durch eine einzige Bedingung pro Bridge erreicht werden. Dabei wurde ausgenutzt, dass an einer Bridge nur jeweils ein einziger Eingangs- bzw. Ausgangslink aktiv sein konnte und auch nur diese einen Sendezeitpunkt ungleich Null besitzen durfte. Daher ließen sich die Sendezeitpunkte von aktiven Eingangs- und Ausgangslinks unabhängig vom Routing durch Aufsummieren der Sendezeitpunkte aller Eingangs- bzw. Ausgangslink erfassen. Die möglichen Verzweigungen eines Pfades aufgrund von Multicast lassen diese Modellierung nicht mehr zu. Stattdessen werden hier nun eingehende und ausgehende Links einer Bridge paarweise betrachtet. Der erste, intuitive Ansatz (daher der Name *MCI*, von *multicast intuitive*) wurde in [E14] veröffentlicht und wird im Folgenden erläutert.

In *MCI* wird das korrekte Scheduling eines Streams k sichergestellt, indem für alle Paare direkt aufeinanderfolgender Links $\overleftarrow{m}, \overrightarrow{m}$ in E_k eine Schedulingbedingung zum Modell hinzugefügt wird, die den notwendigen Anstieg der Sendezeitpunkte genau dann fordert, wenn $\overleftarrow{m}$ in der aktuell gewählten Route enthalten (d.h. $\mathfrak{p}_{k\overleftarrow{m}} = 1$) ist. Die Erstellung entsprechender Bedingungen ist in Algorithmus 6.1 gezeigt. Wird der Link $\overleftarrow{m}$ nicht genutzt (d.h. $\mathfrak{p}_{k\overleftarrow{m}} = 0$), so lässt die aufgestellte Bedingung (für hinreichend großes M_{DPS}) beliebige Sendezeitpunkte zu. Dies ist in Topologien mit mehreren Routingmöglichkeiten erforderlich, damit nicht immer die Verzögerungszeiten von längeren Pfaden berücksichtigt werden, selbst wenn diese gar nicht genutzt werden. Die genutzte Schedulingbedingung wird hier aufgrund dieser Abhängigkeit von der Pfadvariable $\mathfrak{p}_{k\overleftarrow{m}}$ auch als *DPS*-Bedingung (von *dependent path scheduling*) bezeichnet. Darüber hinaus berücksichtigt Algorithmus 6.1 die Modelloption *rlp*. Standardmäßig (d.h. *rlp* nicht aktiv) werden die Schedulingbedingungen zwischen Kanten, die das gleiche Knotenpaar in entgegengesetzter Richtung verbinden, immer inkludiert, da

```
1 foreach k ∈ F do
2     foreach i ∈ V_k^br do
3         foreach (m←, m→) ∈ E_ki^← × E_ki^→ do
4             if m_0^← ≠ m_1^→ or not rlp then
5                 add (t^abs_{km→} − t^abs_{km←} ≥ d^{fwd'}_{km←} − M_DPS·(1 − p_{km←})) to ILP   // DPS constraint
```

Algorithmus 6.1: Erzeugung von Schedulingbedingungen des ILP-Modells *MCI*.

die zum Einsatz kommende Routingbedingung (6.10) eine Schleifenbildung zwischen Nachbarknoten als gültiges Routing zulässt (ohne Fortsetzung der Route zum Sender). Diese fehlerhaften Routen werden dann hier durch die zusätzlichen Schedulingbedingungen zwischen Kanten in entgegengesetzter Richtung vermieden: Die Aufnahme einer Schleife zwischen Nachbarknoten in das aktive Routing aktiviert hier die zugehörigen *DPS*-Bedingungen, welche einen Anstieg des Sendezeitpunkts in einer Schleife fordern, was offensichtlich nicht erfüllbar ist. Bei Aktivierung der Modelloption *rlp* wird dagegen die Routingbedingung (6.11) genutzt, welche solche fehlerhaften Routen vermeidet, sodass die zusätzlichen Schedulingbedingungen nicht erforderlich sind. Zusammengefasst bestehen Routing und Pfadscheduling des Modells *MCI* also aus (6.9), (6.10), Alg. 6.1 (*rlp* := `false`) und Schleifen zwischen Nachbarknoten werden durch Schedulingbedingungen verhindert, während *MCI*^{rlp} hierfür (6.9), (6.11), Alg. 6.1 (*rlp* := `true`) nutzt und bereits aufgrund der Routingbedingungen korrekte Routen bis hin zum Sender erzeugt.

Im Rahmen der Entwicklung wurde das Modell *MCI* zunächst mit dem Optimierungsziel *LT* (Streamlatenzen) getestet. Dabei zeigten sich grundsätzliche Probleme bezüglich Schedulability und Rechenzeit, welche auf die Abhängigkeit der Schedulingbedingungen von den Pfadvariablen zurückgeführt werden konnten: Vor dem Branching der Pfadvariablen ergeben die *DPS*-Bedingungen nicht den eigentlich notwendigen Anstieg der Sendezeitpunkte entlang eines Pfades, sodass in der LP-Relaxation des Modells Lösungen mit negativen Streamlatenzen enthalten sind. Eine Analyse des Suchverlaufs des Solvers zeigt, dass dieser nicht frühzeitig erkennt, dass diese Bereiche des LP-Lösungsraumes keine ganzzahligen Lösungen enthalten und somit später im Lösungsvorgang (nach Branching der Pfadvariablen) stets unlösbar werden. Daher wendet der Solver Zeit für das zwecklose Durchsuchen ebendieser Bereiche auf. Unabhängig von den Pfadvariablen lassen sich diese Bereiche mit negativen Streamlatenzen durch zusätzliche Bedingungen auch aus dem LP-Lösungsraum ausschließen [E14], was hier als Modelloption beschrieben wird.

Modelloption *isp* Die Option *isp* fordert mittels (6.12), dass der Sendezeitpunkt $\boldsymbol{t}_{km}^{\boldsymbol{abs}}$ auf einem Link m gegenüber dem Sendezeitpunkt $\boldsymbol{t}_{kn}^{\boldsymbol{abs}}$ auf dem ersten Link der Route n mindestens um die Verzögerungszeit $lt_{km}^{i,sp''}$ des kürzesten Pfades von n nach m verschoben sein muss. Dabei ist $lt_{km}^{i,sp''}$ die Latenz des kürzesten Pfades unter Idealbedingungen (*ideal shortest path*, *isp*), also ohne Queuing bzw. Zeitschlitzkonflikte (vgl. auch lt^{norm} in Kapitel 4.2.2.2) und ohne die Latenz von m selbst. Außerdem muss m über den bei der Latenzbestimmung genutzten Pfad auch tatsächlich *benutzbar* sein, d.h., m darf nicht die Gegenrichtung des direkten Vorgängerlinks sein (Vermeidung von Schleifen zwischen Nachbarknoten, siehe auch Abb. 6.9). Der vollständige Algorithmus zur Bestimmung von $lt_{km}^{i,sp''}$ ist in Anhang C.3 gezeigt.

$$\forall k \in F, \forall (n,m) \in E_k \times E_k \mid n_0 = f_k.src, m_0 \neq f_k.src \qquad \boldsymbol{t}_{km}^{\boldsymbol{abs}} - \boldsymbol{t}_{kn}^{\boldsymbol{abs}} \geq lt_{km}^{i,sp''} \tag{6.12}$$

6.5.2 Multicast-Basismodell *MCT*

Alternativ zum Modell *MCI* einschließlich Modelloption *isp* wurde in [E14] ein Modell vorgeschlagen, das die problematischen *DPS*-Bedingungen durch von den Pfadvariablen unabhängige *IPS*-Bedingungen (von *independent path scheduling*) ersetzt, immer wenn dies ohne Einfluss auf die Korrektheit des Modells möglich ist. Es basiert weiterhin auf den Anwendungs- und Ressourcenbedingungen (6.6) – (6.8) aus *UC* und nutzt außerdem das für *MCI* eingeführte Routing (6.9) – (6.11), unterscheidet sich von diesem jedoch durch den Algorithmus zur Erzeugung der Schedulingbedingungen.

Die Erzeugung der Schedulingbedingungen dieses Basismodells *MCT* ist in Algorithmus 6.2 gezeigt. Wie schon bisher erfolgt die Erstellung der Bedingungen separat für jeden Stream $k \in F$. Das Vorgehen für einen Stream ist dabei an Dijkstras bekannten Algorithmus zum Finden kürzester Pfade angelehnt. Im Gegensatz zum bekannten Dijkstra-Algorithmus wird hier allerdings nicht ausgehend vom Startknoten $f_k.src$ nach den kürzesten Pfaden zu allen anderen Knoten gesucht, sondern nach den kürzesten Pfaden zu allen Kanten in E_k. Jedes Mal, wenn eine bisher nicht besuchte Kante $\vec{m}$ erreicht wird, wird dem ILP-Modell eine *IPS*-Bedingung (Alg. 6.2, Z.11) zwischen dieser neuen Kante und der aktuellen Vorgängerkante $\overleftarrow{m}$ hinzugefügt, welche die notwendige Weiterleitungsverzögerung zwischen den beiden Kanten darstellt. Wird eine bereits besuchte Kante $\vec{m}$ nochmals über einen längeren Pfad erreicht, so wird zwischen dieser Kante und der Vorgängerkante $\overleftarrow{m}$ des längeren Pfades eine *DPS*-Bedingung (Alg. 6.2, Z.16) hinzugefügt. Als Metrik für die Pfadlänge kommt dabei immer die Verzögerungszeit zum Einsatz. Das Ergebnis von Algorithmus 6.2 ist in Abbildung 6.10 beispielhaft für einen Stream gezeigt. Darin ist ersichtlich, dass der Algorithmus ausgehend vom Sender eine Baumstruktur aus *IPS*-Bedingungen erzeugt, welche die kürzesten Pfade zu allen Kanten abbildet (daher der Name *MCT*, von *multicast tree*), während immer beim Zusammenlaufen eines längeren Pfades mit einem kürzeren Pfad eine *DPS*-Bedingungen eingesetzt wird. Insgesamt sind die Schedulingbedingungen somit bevorzugt unabhängig von den Pfadvariablen (*IPS*-Bedingung) und *DPS*-Bedingungen werden nur gezielt auf längeren Pfaden eingesetzt, um deren Verzögerung nur dann zu berücksichtigen, wenn diese auch tatsächlich im gewählten Routing genutzt werden. An dieser Stelle sei angemerkt, dass ungenutzte Links hier im Gegensatz zu *UC* keinen Sendezeitpunkt von Null erhalten, weshalb die Vereinfachung aus (6.5), den Sendezeitpunkt des eingehenden Links mittels

1 **foreach** $k \in F$ **do**
2 $i := f_k.src$
3 $visited := E_{ki}^{\rightarrow}$ // initialize set of visited links
4 $next := ((m, d_{km}^{fwd\prime}))_{m \in E_{ki}^{\rightarrow}}$ // a sequence of (link, delay) tuples for outgoing links of i
5 **while** $next \neq ()$ **do**
6 sort $next$ by delay (2nd field of each element)
7 $\overleftarrow{m}, delay :=$ pop_front($next$) // remove and return first element of $next$
8 $i := \overleftarrow{m}_1$
9 **foreach** $\vec{m} \in E_{ki}^{\rightarrow}$ **do** // iterate successor links of $\overleftarrow{m}$
10 **if not** ($\vec{m} \in visited$ **or** $\overleftarrow{m}_0 = \vec{m}_1$) **then**
11 add ($\boldsymbol{t}_{k\vec{m}}^{\boldsymbol{abs}} - \boldsymbol{t}_{k\overleftarrow{m}}^{\boldsymbol{abs}} \geq d_{k\overleftarrow{m}}^{fwd\prime}$) to ILP // *IPS* constraint
12 $visited := visited \cup \{\vec{m}\}$
13 append $(\vec{m}, delay + d_{k\vec{m}}^{fwd\prime})$ to $next$
14 **if** ($\vec{m} \in visited$ **and** $\overleftarrow{m}_0 \neq \vec{m}_1$) **or**
15 ($\overleftarrow{m}_0 = \vec{m}_1$ **and not** *rlp*) **then**
16 add ($\boldsymbol{t}_{k\vec{m}}^{\boldsymbol{abs}} - \boldsymbol{t}_{k\overleftarrow{m}}^{\boldsymbol{abs}} \geq d_{k\overleftarrow{m}}^{fwd\prime} - M_{DPS} \cdot (1 - \mathfrak{p}_{k\overleftarrow{m}})$) to ILP // *DPS* constraint

Algorithmus 6.2: Erzeugung von *IPS*- und *DPS*-Bedingungen des ILP-Modells *MCT*.

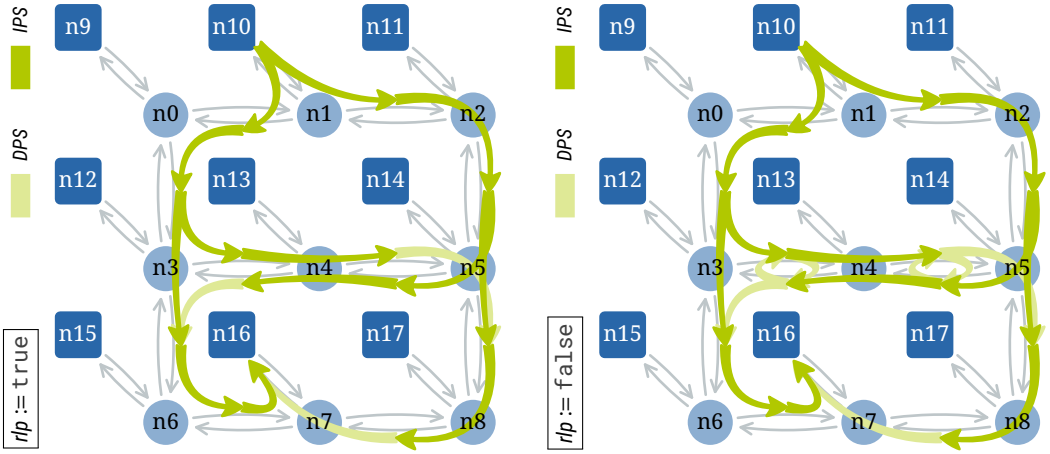


Abbildung 6.10: Von Algorithmus 6.2 erzeugte *IPS*- und *DPS*-Bedingungen für einen Stream von $n10$ nach $n16$ in einer homogenen Topologie (identische Verzögerung durch alle Bridges/Links). Jeder grüne Pfeil symbolisiert eine Schedulingbedingung zwischen den verbundenen Kanten.

Aufsummieren der Sendezeitpunkte aller eingehenden Links zu ermitteln, hier nicht angewendet werden kann. Äquivalent zu *MCI* unterstützt auch *MCT* zwei Varianten zur Vermeidung von Schleifen zwischen Nachbarknoten. Dementsprechend bestehen Routing und Pfadscheduling von *MCT* aus (6.9), (6.10), Alg. 6.2 (*rlp* := `false`), während MCT^{rlp} die Randbedingungen (6.9), (6.11), Alg. 6.2 (*rlp* := `true`) umfasst.

6.5.3 Modelloptionen gegen Redundanz (*ne*) und Linküberlastung (*lu1*)

Die Multicastmodelle sollen hier zusammen mit zwei bisher nicht veröffentlichten Modelloptionen getestet werden. Die erste Modelloption behandelt das Problem, dass die gezeigten Multicastmodelle auch Lösungen liefern, in denen mehr aktive Links enthalten sind, als tatsächlich für die jeweilige Route benötigt werden. Dies umfasst beispielsweise die Verbindung zweier Knoten auf mehreren

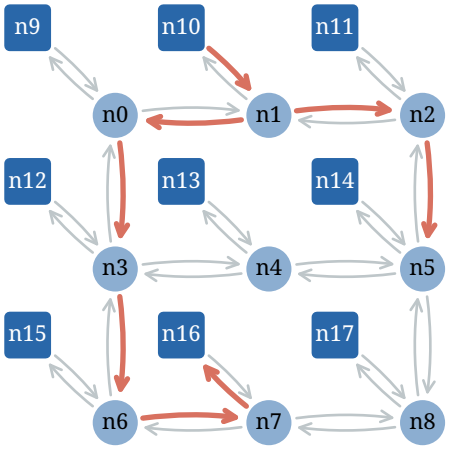


Abbildung 6.11: Überflüssige Zusatzlinks in einer Route von $n10$ nach $n16$.

redundanten Pfaden, sowie die Verzweigung in „Sackgassen“. Mit Sackgassen sind dabei Teile eines Pfades gemeint, die nach einer Verzweigung an einem beliebigen Knoten enden (d.h. nicht an einem Empfänger des Streams), was in Abbildung 6.11 gezeigt ist.

Modelloption *ne* Sackgassen und redundante Pfade werden durch die Option *ne* (von *no extra links*) mittels der Bedingungen (6.13) und (6.14) ausgeschlossen.

$$\forall k \in F, \forall i \in V_k^{br} \qquad \sum_{m \in E_{ki}^{\rightarrow}} \mathfrak{p}_{km} \;\geq\; \sum_{m \in E_{ki}^{\leftarrow}} \mathfrak{p}_{km} \tag{6.13}$$

$$\forall k \in F, \forall i \in V_k^{br} \qquad \sum_{m \in E_{ki}^{\leftarrow}} \mathfrak{p}_{km} \;\leq\; 1 \tag{6.14}$$

Sackgassen werden durch (6.13) vermieden, indem jeder Bridgeknoten mit aktiven Eingangslinks auch aktive Ausgangslinks benötigt. Redundante Pfade sind unzulässig, da (6.14) maximal einen aktiven Eingangslink an einer Bridge erlaubt und redundante Pfade daher nicht mehr zusammengeführt werden können.

Es sei angemerkt, dass die zusätzlichen Links in Form von Sackgassen und redundanten Pfaden grundsätzlich keine Fehler in der Lösung darstellen und neben der Modelloption *ne* auch durch einen Nachverarbeitungsschritt aus der Lösung entfernt werden könnten. Dennoch kann die Modelloption bei der weiteren Modellentwicklung hilfreich sein, da die überflüssigen Links beispielsweise beim inkrementellen Lösen des ILP problematisch sind und ohne die Option *ne* in jeder Iteration eine entsprechende Nachbearbeitung der Lösung notwendig wäre.

Ein weiteres Problem, das alle bisher gezeigten Modelle (*UC*, *MCI*, *MCT*) betrifft, liegt in der Erkennung überlasteter Links (Auslastung über $100\,\%$). Da in keiner der ILP-Bedingungen explizit die Linkauslastung unter Berücksichtigung der Pfadvariablen berechnet wird, kann eine durch das gewählte Routing verursachte Linküberlastung nur dadurch erkannt werden, dass keine konfliktfreien Zeitschlitzpositionen für dieses Routing gefunden werden können. Um dies festzustellen, ist unter Umständen ein Ausprobieren aller möglichen Zeitschlitzreihenfolgen auf dem betroffenen Link erforderlich. Hierbei wäre ein vielfaches Branching auf den zugehörigen ILP-Variablen $\mathfrak{a}_{klmxy}$ erforderlich, welche die Reihenfolge der Zeitschlitze modellieren. Diese zeitaufwändige Form der Erkennung von Linküberlastung lässt sich durch eine explizite Modellierung der Linkauslastung vermeiden.

Modelloption *lu1* Bei Aktivierung der Option *lu1* (von *link utilization < 1*) wird die Linkauslastung in (6.15) modelliert. Der Vorteil besteht darin, dass dies unabhängig vom Scheduling geschieht.

$$\forall m \in \mathcal{E} \qquad \sum_{k \in F \mid m \in E_k} luf \cdot \frac{sl'_{km}}{f_k.ct} \cdot \mathfrak{p}_{km} \;\leq\; luf \tag{6.15}$$

Die Bedingung stellt sicher, dass die Auslastung auf jedem Link weniger als $100\,\%$ beträgt. Die Multiplikation mit einem Faktor $luf := 5000$ wird dabei aus numerischen Gründen vorgenommen (vgl. [72, § 29]).

Die Modelloption *lu1* kann bei der schnelleren Erkennung nicht lösbarer Szenarien helfen und die Rechenzeiten von Szenarien mit hoher Netzwerkauslastung verbessern, da ungültige Routingentscheidungen vermieden werden. Bei geringer Netzwerkauslastung sollte die Optionen dagegen keinen relevanten Einfluss haben.

6.5.4 Evaluation der ILP-Modelle *MCI* und *MCT*

Die vorgestellten Basismodelle *MCI* und *MCT* werden unter Einbeziehung der neuen Modelloptionen *rlp*, *isp*, *ne* und *lu1* und in Kombination mit den drei Optimierungszielen *1SP*, *SPLT25* und *LT25* bewertet. Aus Rechenzeitgründen können dabei nicht alle möglichen Kombinationen evaluiert werden. Standardmäßig sind daher alle der genannten Optionen aktiv und der jeweilige Nutzen einer Modelloption wird durch das Ausschalten von alleine dieser Option bewertet. Da in Kombination mit den verschiedenen Basismodellen und Optimierungszielen weiterhin mehr als 30 ILP-Varianten verbleiben, erfolgt die Bewertung dennoch in mehreren Abschnitten.

6.5.4.1 Evaluation von *MCI* mit *SPLT25*

Die Ergebnisse des Modells *MCI* für den optimierenden Fall sind in Abbildung 6.12 gezeigt. Insgesamt sind dabei in Kombination mit *SPLT25* keine wesentlichen Performanceunterschiede zwischen den einzelnen Modellvarianten erkennbar. Geringfügige Unterschiede sollen hier mangels eines Signifi-

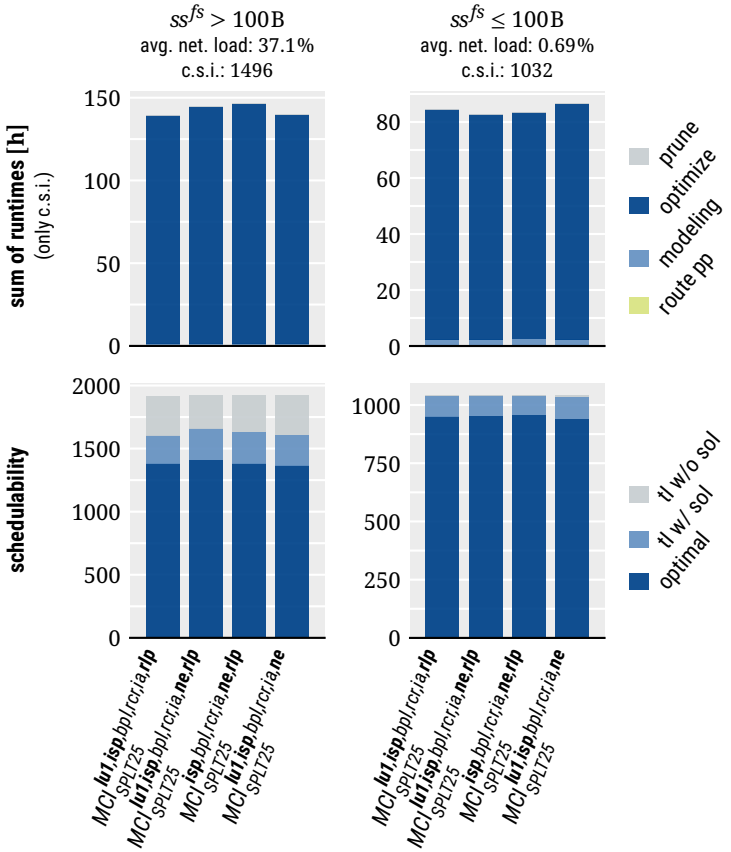


Abbildung 6.12: Schedulability und c.s.i.-Rechenzeiten für *MCI* bei Deaktivierung von *ne*, *lu1* und *rlp* für Optimierungsziel *SPLT25*. Die Rechenzeit *prune* beinhaltet eine Nachbearbeitung zum Entfernen überflüssiger Links aus Lösungen von Modellen ohne *ne*. Diese fällt i.d.R. sehr gering aus und ist daher nicht sichtbar.

kanztests für die zusammengefassten Daten aller Testcases nicht weiter betrachtet werden. In der Auswertung der einzelnen Testcases (siehe [E19]) lassen sich zwar ebenfalls vereinzelte Performanceunterschiede in Abhängigkeit vom jeweiligen Testcase auffinden, jedoch sind die Abweichungen auch dort gering und weisen nicht auf eine grundsätzlich überlegene Modellvariante hin. Für *LT25* zeigt sich das gleiche Bild bei insgesamt schlechterer Schedulability (siehe [E19]).

Für die Modelloption *ne* ist der geringe Performanceeinfluss erwartungsgemäß, da die Option der Bereinigung von unnötigerweise geplanten Zeitschlitzen dient und als Alternative zu einer Nachbearbeitung von Lösungen entworfen wurde. Der geringe Performanceeinfluss von *rlp* bedeutet hingegen, dass die Routingbedingung (6.11) und die *DPS*-Bedingung des Pfadschedulings gleichermaßen effektiv darin sind, ungültige Routen aus dem Lösungsraum auszuschließen. Bei Nutzung von *lu1* zeigt sich ebenfalls kein Performancevorteil bei Szenarien mit hoher Netzwerkauslastung, obwohl dieser durchaus erwartet wurde. Dies kann nur dadurch erklärt werden, dass die vom ILP-Solver getroffenen Branchingentscheidungen auch ohne *lu1* bereits solche Teilprobleme bevorzugt bearbeiten, für die ganzzahlige Lösungen ohne Linküberlastung existieren. Ohne *lu1* tragen dabei vor allem die Ressourcenbedingungen und das Optimierungsziel zu Branchingentscheidungen bei, Routings mit Linküberlastung zu vermeiden. Dementsprechend hat *lu1* in solchen Fällen keine Relevanz für das Lösen des ILP. Es sei angemerkt, dass *lu1* in Szenarien, für die überhaupt kein gültiges Routing (d.h. ohne Linküberlastung) existiert, zu einem stark beschleunigten Nachweis der Unlösbarkeit beitragen kann. Dies wird im Rahmen des Benchmarkings allerdings nicht sichtbar, da solche Szenarien bereits bei der Erstellung der Benchmarkingszenarien vermieden werden (vgl. Kapitel 4.2.1).

Abschließend soll außerdem der Einfluss der im gezeigten Ergebnisdiagramm stets aktivierten Modelloption *isp* beschrieben werden. Eine detaillierte quantitative Performanceanalyse von *MCI* ohne *isp* wird hier nicht gezeigt, da das umfangreiche Benchmarking aufgrund der vergleichsweise schlechten Performance dieses Modells nicht in angemessener Zeit durchgeführt werden konnte. Beispiele für die schlechte Performance des Modells (insbesondere bei Nutzung von *LT25*) sind in [E14] gezeigt, da dort aufgrund eines bedeutend geringeren Testumfangs noch ein direkter Vergleich zu anderen Modellvarianten möglich war. Wie schon in diesen früher durchgeführten Analysen war es dem ILP-Solver im Zuge des Benchmarkings bei deaktiviertem *isp* nicht möglich, effektiv festzustellen, dass Bereiche des LP-Lösungsraumes mit negativen Streamlatenzen keine ganzzahligen Lösungen (bzgl. der Pfadvariablen) enthalten und somit im Branching gemieden werden sollten. Je nach Optimierungsziel des ILP-Modells ergaben sich hierdurch unterschiedliche Effekte:

- *LT25*: Das Optimierungsziel „treibt“ die Suche in Bereiche des LP-Lösungsraumes mit negativen Streamlatenzen, da diese laut dem Optimierungsziel gute Lösungen darstellen. Da diese Bereiche jedoch keine ganzzahligen Lösungen beinhalten, führt dies zu hohen Rechenzeiten bzw. schlechter Schedulability unter dem gegebenen Zeitlimit (vgl. Ergebnisse aus [E14]).
- *SPLT25*: Unter Nutzung des primären Optimierungsziels bezüglich Anzahl genutzter Links werden (bei ggf. schlechter Performance) Lösungen gefunden, das sekundäre Optimierungsziel bezüglich Streamlatenzen leidet jedoch unter schlechter Best-Bound-Ermittlung und kann daher die geforderte Optimalität nicht effektiv nachweisen. Dies führt im Vergleich zu anderen Modellvarianten häufiger zu langen Rechenzeiten bis zum Abbruch durch das Zeitlimit.

- *1SP*: Das Fehlen der Modelloption *isp* hat kaum negative Auswirkungen, da diese nur Auswirkungen auf die Best Bound bei Latenzoptimierung hat, nicht jedoch bei Optimierung von Pfadlängen. Die Performance kann aufgrund der problematischen Schedulingbedingungen jedoch weiterhin schlechter ausfallen als bei anderen Modellvarianten.

6.5.4.2 Evaluation von *MCT* mit *SPLT25*

Die Ergebnisse von *MCT* mit Optimierung sind in Abbildung 6.13 gezeigt. Diese enthalten außerdem eine Modellvariante $MCT^{bpl,rcr,rlp}$, welche dem Modell *pdr_rlp* aus [E14] entspricht. Die gegenüber dieser bereits veröffentlichten Modellvariante erzielten Verbesserungen bezüglich Schedulability und Rechenzeiten sind dabei vor allem auf die bereits in Kapitel 6.4.5 für das Basismodell *UC* analysierte Modelloption *ia* zurückzuführen. Die Variation der weiteren, in diesem Unterkapitel vorgestellten Modelloptionen zeigt dagegen grundlegend das gleiche Verhalten wie mit dem Basismodell *MCI*. Maßgebliche Performanceeinflüsse sind demnach weder in den hier gezeigten zusammengefassten Ergebnissen noch in den Ergebnissen der einzelnen Testcases zu verzeichnen, wobei die Ursachen denen des Modells *MCI* entsprechen. In [E19] kann außerdem nachgeschlagen werden, dass sich das gleiche Bild auch für das Optimierungsziel *LT25* zeigt.

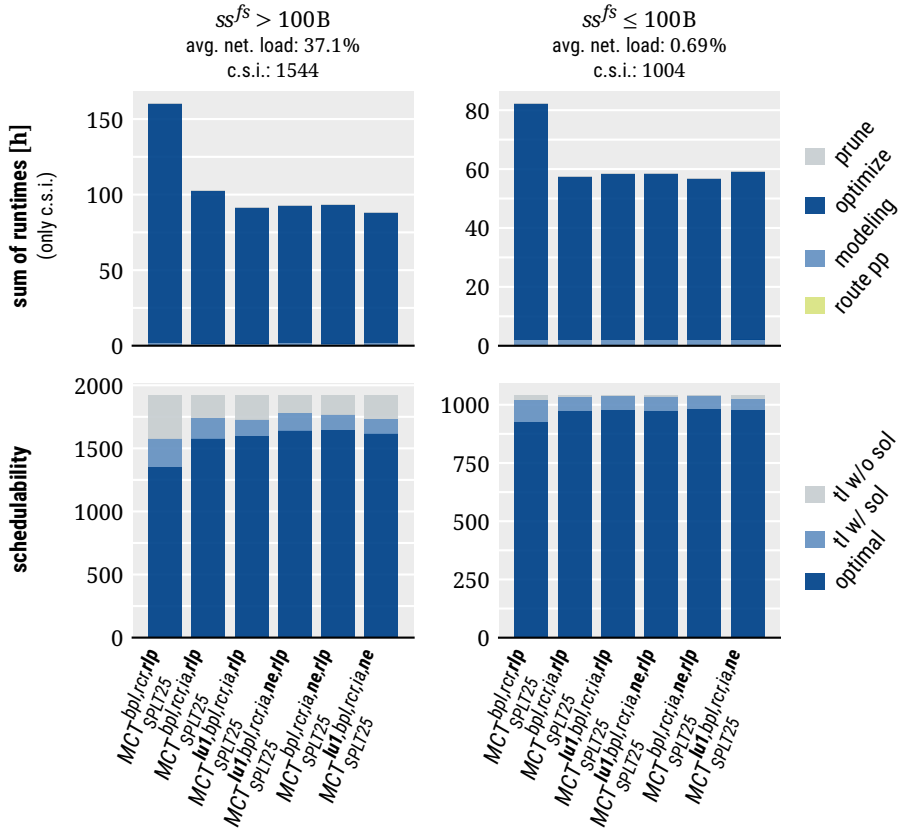


Abbildung 6.13: Schedulability und c.s.i.-Rechenzeiten für *MCT* bei Deaktivierung von *ne*, *lu1* und *rlp* für Optimierungsziel *SPLT25*.

6.5.4.3 Vergleich von *MCI* und *MCT* mit *SPLT25* und *LT25*

Unter Nutzung aller Modelloptionen dieses Unterkapitels (*lu1*, *ne*, *rlp*) werden die beiden Basismodelle *MCI* und *MCT* für den optimierenden Fall in Abbildung 6.14 verglichen. Dabei ist das Basismodell *MCT* deutlich überlegen, wobei die Performancevorteile für beide genutzten Optimierungsziele in der gleichen Größenordnung liegen. Die Analyse der einzelnen Testcases bestätigt darüber hinaus, dass *MCT* durchgängig bessere Ergebnisse liefert. Die grundsätzliche Überlegenheit des Modells ist dabei in Anbetracht des optimierten Pfadschedulings erwartungsgemäß. Der wesentliche Vorteil von *MCT* liegt dabei darin, dass bedeutend stärkere Einschränkungen hinsichtlich der Schedulingvariablen getroffen werden, sodass ungültige Belegungen dieser bereits ohne Branching auch aus dem LP-Lösungsraum ausgeschlossen werden. In *MCI* ist dagegen ein vielfaches Branching auf den Pfadvariablen notwendig, um eine vergleichbare Einschränkung des Lösungsraumes zu erzielen, was einen direkten Einfluss auf die Rechenzeiten hat. Während die Modelloption *isp* dieses Problem abschwächt, indem die Sendezeiten auf beliebigen Links in Bezug auf die Sendezeit auf dem ersten Link eingeschränkt werden, fehlt dem Modell im Vergleich zu *MCT* hierbei weiterhin eine von Pfadvariablen unabhängige Einschränkung der Sendezeiten zwischen beliebigen Links der Topologie untereinander (also nicht nur bezüglich des ersten Links).

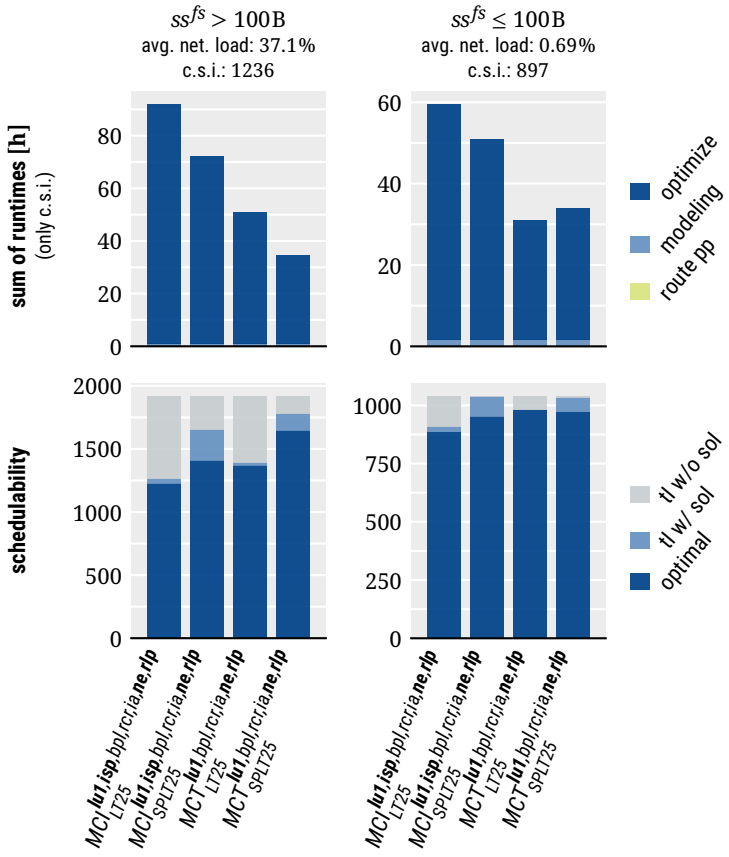


Abbildung 6.14: Vergleich von Schedulability und c.s.i.-Rechenzeiten für *MCI* und *MCT* für die Optimierungsziele *LT25* und *SPLT25*.

6.5.4.4 Vergleich von *MCI* und *MCT* mit *1SP*

Für den nicht-optimierenden Fall sind alle getesteten Modellvarianten der beiden Basismodelle *MCI* und *MCT* in Abbildung 6.15 gezeigt. Die zuvor schon für den optimierenden Fall verglichenen Modelloptionen werden hier für das Optimierungsziel *1SP* erneut getestet, um festzustellen, ob sich der Einfluss dieser Optionen signifikant unterscheidet, je nachdem ob initiale Lösungen gesucht werden oder darüber hinaus nach besseren Lösungen gesucht wird. Insbesondere bezüglich der Option *rlp* fällt auf, dass dies tatsächlich der Fall ist: Während die Modelloption für den optimierenden Fall kaum Einfluss auf die Rechenzeiten hatte (siehe Abb. 6.12, 6.13), zeigt sich hier für den nicht-optimierenden Fall eine erhöhte Rechenzeit für Modelle ohne *rlp*. Dies lässt sich auch durch statistisch signifikante Ergebnisse aus den einzelnen Testcases (vgl. [E19]) untermauern, soll hier jedoch aufgrund der weiterhin verhältnismäßig geringen Vorteile nicht im Detail diskutiert werden. Grundsätzlich zeigt dieser Fall allerdings, dass Modellierungsunterschiede sich unterschiedlich auf das Finden initialer Lösungen und den darauf folgenden Optimierungsvorgang auswirken können, sodass in der Evaluation stets beide Fälle berücksichtigt werden sollten. Dies bestätigt sich auch beim Vergleich von *MCI* und *MCT*. Während der Rechenzeitvorteil von *MCT* in Kombination mit *LT25* und *SPLT25* sowohl bei niedriger als auch bei hoher Netzwerkauslastung bei mehr als 30 % lag (Abb. 6.14), und in Kombination mit *1SP* bei hoher Netzwerkauslastung sogar bei mehr als 50 %

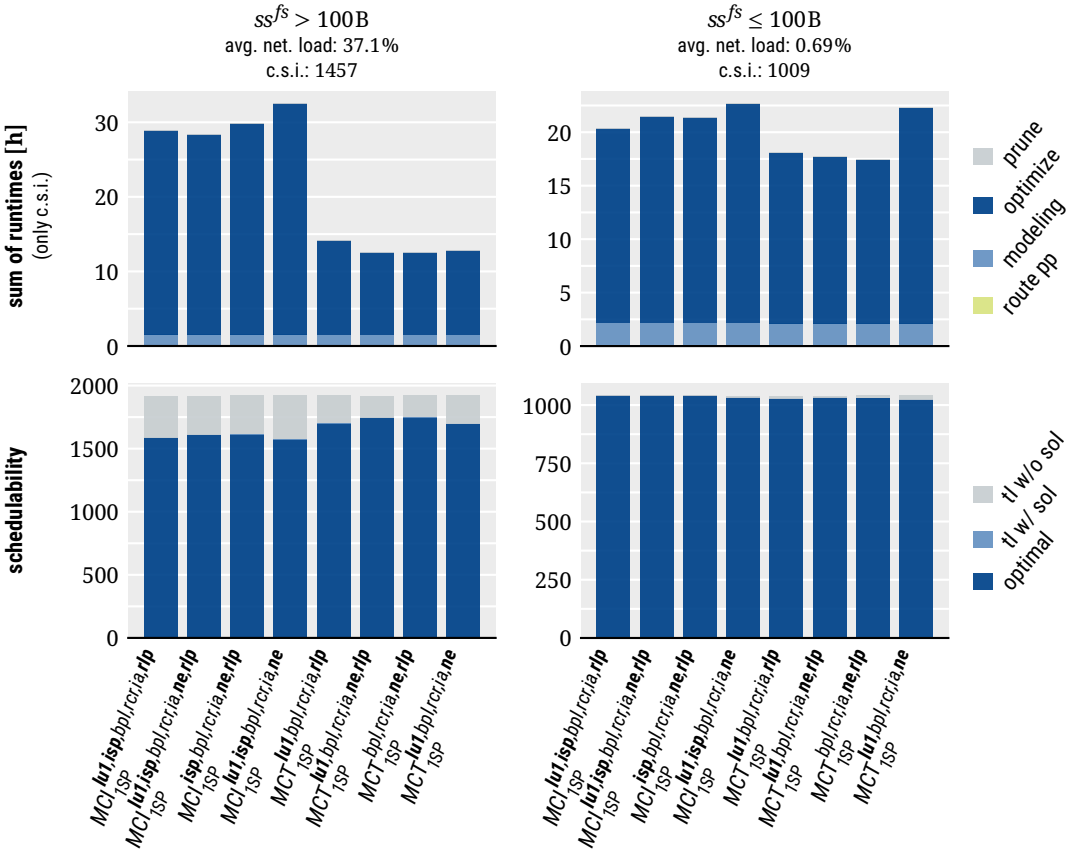


Abbildung 6.15: Vergleich von Schedulability und c.s.i.-Rechenzeiten für *MCI*/*MCT* mit Optimierungsziel *1SP*.

(Abb. 6.15), fällt der Vorteil für *1SP* und niedrige Netzwerkauslastung mit etwa 20 % deutlich geringer aus. Da derartige Unterschiede mit dem verwendeten ILP-Solver nicht hinsichtlich getroffener Branchingentscheidungen analysiert werden können, kann dies hier nur intuitiv bergründet werden. Während des Optimierungsvorgangs bezüglich Streamlatenzen (*LT25*/*SPLT25*) trägt das optimierte Pfadscheduling von *MCT* grundsätzlich zu besseren Branchingentscheidungen bezüglich der Schedulingvariablen bei. Ebenso ist dies ohne Optimierungsvorgang (*1SP*) der Fall, wenn aufgrund der hohen Netzwerkauslastung ein vielfaches Branching und Backtracking auf Schedulingvariablen notwendig ist, um überhaupt eine Lösung zu finden. Ohne Optimierungsvorgang und bei zugleich niedriger Netzwerkauslastung sinkt jedoch die Notwendigkeit eines wiederholten Branchings und Backtrackings auf Schedulingvariablen, sodass deren verbesserte Beschränkung in *MCT* weniger relevant ist. Insgesamt ist *MCT* dennoch auch für den nicht-optimierenden Fall grundsätzlich überlegen gegenüber *MCI*, wohingegen die Modelloptionen *lu1*, *ne* und *rlp* keinen maßgeblichen Einfluss auf die Performance haben.

Auf Basis der gezeigten Ergebnisse wurde entschieden, bei der Weiterentwicklung der ILP-Modelle ausschließlich das überlegene Modell *MCT* zu berücksichtigen. Die Modelloptionen *lu1*, *ne* und *rlp* werden darüber hinaus in folgenden Evaluationen standardmäßig aktiviert, auch wenn keine maßgeblichen Performanceunterschiede vorliegen. Dies wird vor allem mit von der Performance unabhängigen Eigenschaften der ILP-Modellierung begründet:

- *lu1* kann bei bestimmten Eingangsdaten zur Erkennung nicht lösbarer Szenarien beitragen.
- *ne* sorgt für Lösungen, die keine Nachbearbeitung bzgl. überflüssiger Zeitschlitze erfordern.
- *rlp* bietet seitens der Modellierung eine intuitivere Schleifenvermeidung zwischen Nachbarknoten (verglichen mit der Schleifenvermeidung durch zusätzliche Schedulingbedingungen).

6.5.5 Performancevergleich von Unicast- und Multicastmodellen

Das überlegene Multicastmodell *MCT* wird abschließend mit dem Unicastmodell *UC* verglichen. Die Modelle werden dabei unter Nutzung der drei bekannten Optimierungsziele verglichen, wobei alle relevanten, zuvor untersuchten Modelloptionen aktiviert sind und als *o1* zusammengefasst werden. Dies dient der besseren Übersicht bei der Entwicklung weiterer Modellvarianten und umfasst hier je nach Basismodell folgende Optionen:

- *UC*: Die Option *o1* fasst *bpl,rcr,ia,lu1* zusammen.
- *MCT*: Die Option *o1* fasst *bpl,rcr,ia,lu1,ne,rlp* zusammen.

Die in Abbildung 6.16 gezeigten Ergebnisse zeigen in Übereinstimmung mit den Ergebnissen aus [E14], dass *UC* grundsätzlich performanter arbeitet. Die quantitativen Unterschiede hängen dabei sowohl vom Optimierungsziel als auch von den Eingangsdaten ab und bewegen sich im Bereich der zweifachen bis dreifachen Rechenzeit für das Multicastmodell im Falle hoher Netzwerkauslastung, und bis zur neunfachen Rechenzeit bei niedriger Netzwerkauslastung. Bei Betrachtung der einzelnen Testcases (siehe z.B. Abb. 6.17) ist ersichtlich, dass mit Optimierungsziel *1SP* in vielen Fällen ähnliche Rechenzeiten für *UC* und *MCT* erzielt werden und insbesondere der Optimierungsvorgang von

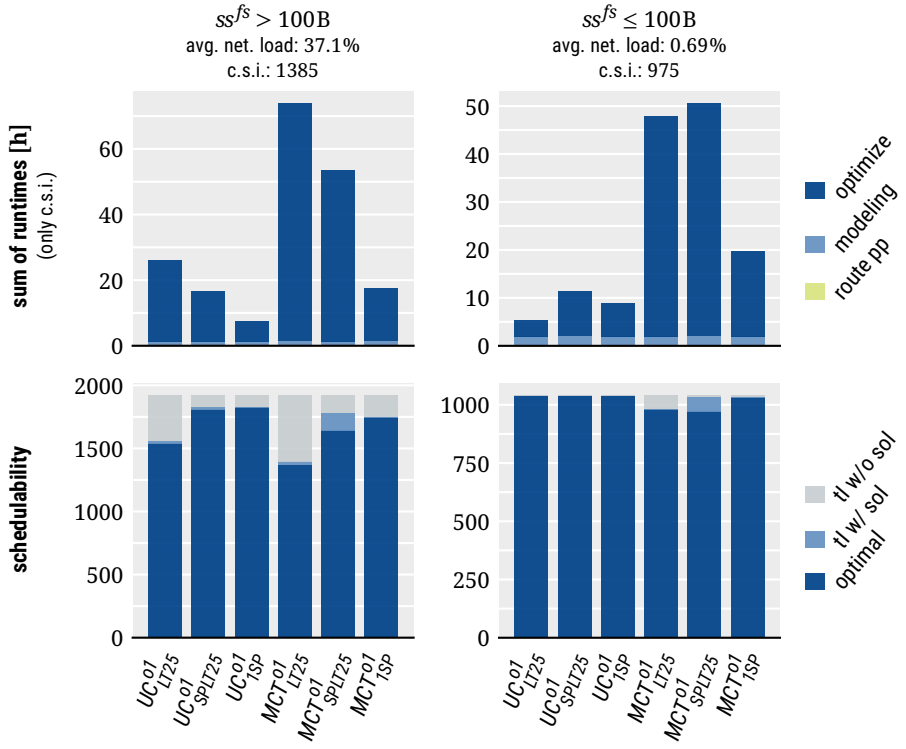


Abbildung 6.16: Schedulability und c.s.i.-Rechenzeiten von *UC* und *MCT* im Vergleich.

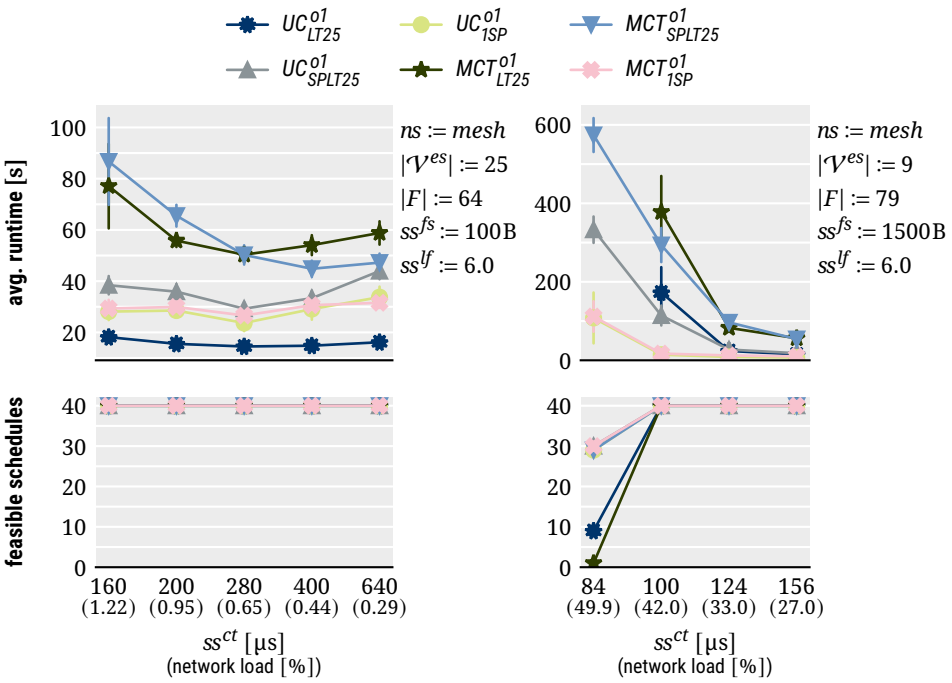


Abbildung 6.17: Ausschnitt der Ergebnisse für TC-DS (links) und TC-L (rechts) für *UC* und *MCT*.

SPLT25/*LT25* für *MCT* bedeutend zeitaufwändiger ausfällt. In der Zusammenfassung in Abbildung 6.16 äußert sich dies darin, dass der relative Unterschied der Modelle für *1SP* am kleinsten ausfällt. Im Gegensatz zu den zuvor analysierten Modelloptionen und der Gegenüberstellung von *MCI* und *MCT* lassen sich die Performanceunterschiede zwischen *UC* und *MCT* nicht auf einen einzelnen, konkreten Modellierungsunterschied zurückführen. Die Unterschiede bei den Bedingungen für Routing und Pfadscheduling führen zu wesentlichen Veränderungen in der Repräsentation der Probleminstanzen gegenüber dem ILP-Solver und damit einhergehend zu unterschiedlichen Branchingentscheidungen, deren detaillierte Analyse im Rahmen der Arbeit nicht möglich war bzw. nicht angestrebt wurde.

6.5.6 Zusammenfassung der Entwicklung von Multicastmodellen

Bei der Entwicklung von Multicastmodellen zum Lösen von TT-RSP wurde zuerst das Modell *MCI* entworfen. Da dieses insbesondere bei der Optimierung von Streamlatenzen eine schlechte Performance gezeigt hat, wurde das Pfadscheduling im Rahmen des Modells *MCT* überarbeitet. In diesem Modell werden die Schedulingvariablen bereits unabhängig von den Pfadvariablen stärker eingeschränkt, was auch zu stärker beschränkten LP-Relaxationen führt. Im Gegensatz zu *MCI* weist das *MCT* daher auch bei der Optimierung von Streamlatenzen keine Probleme im Suchverlauf und bei der Bestimmung der Best Bound auf, was sich auch an deutlich verbesserten Benchmarkingergebnissen zeigt. In der weiteren Entwicklung wird daher nur noch das Modell *MCT* berücksichtigt. Außerdem wurden im Zuge der Multicastmodelle auch die Modelloptionen *ne* und *lu1* eingeführt. Diese führen zwar zu keinen maßgeblichen Verbesserungen in der Performance, werden jedoch aufgrund funktionaler Vorteile im weiteren Verlauf standardmäßig aktiviert. So führt *ne* zu bereinigten Lösungen ohne redundante Zeitschlitzreservierungen, während *lu1* in bestimmten Fällen zu einer schnelleren Erkennung unlösbarer Szenarien führt. Die Performance des Unicastmodells *UC* kann jedoch auch von *MCT* nicht erreicht werden, was daher Gegenstand weiterer Untersuchungen in Kapitel 7.1 ist.

6.6 Zwischenfazit zu ILP-basierten Lösungsverfahren für TT-RSP

Die Entwicklungen in diesem Kapitel wurden u.a. durch die Problemstellung motiviert, die aus der frühen Arbeit [E11] hervorgegangen ist: Dort wurde ein heuristisches Schedulingverfahren für ein OpenFlow-basiertes Echtzeitnetzwerk entworfen, bei dem jedoch mangels Vergleichsmöglichkeit weitgehend offen blieb, wie Performance und Fähigkeiten dieses Schedulers zu bewerten sind. Konkret fehlte es zum damaligen Zeitpunkt also an einer *Referenzlösung* für TT-RSP, welche Vergleichswerte bezüglich der Schedulerperformance, Lösbarkeit von Szenarien und Qualität gefundener Schedules liefert. Zugleich stieg die Relevanz derartiger Schedulingverfahren durch die Standardisierung von TSN als Teil von IEEE 802.1Q. Da auch aus der Literatur keine entsprechenden Lösungsverfahren zur Verfügung standen, wurden diese im Rahmen von [E12, E14] erarbeitet und in diesem Kapitel in verbesserter Form mitsamt umfassenderer Evaluation vorgestellt.

Um die Zielsetzung einer Referenzlösung erfüllen zu können, sollte der Lösungsansatz performancekritische Komponenten der Implementierung mittels etablierter Bibliotheken realisieren, und die zugrunde liegende Methodik sollte eine bestmögliche Lösungsraumabdeckung sowie flexible

Randbedingungen und Optimierungsziele bieten. Entsprechend dieser Anforderungen wurde Integer Linear Programming (ILP) als Basis für die entworfenen Lösungsverfahren genutzt, wobei zunächst ein Unicastmodell und später ein Multicastmodell erstellt wurden, in denen *Routing und Scheduling* in einem *gemeinsamen mathematischen Modell* abgebildet werden (JRaS). Eine Besonderheit der in diesem Kapitel vorgestellten Arbeiten liegt außerdem darin, dass eine Vielzahl von ILP-Modellvarianten entworfen und evaluiert wurde. Zu diesem Zweck wurde die Beschreibung der ILP-Modelle unterteilt in Basismodelle, Modelloptionen und Optimierungsziele:

- Basismodelle stellen vollständige, zum Lösen von TT-RSP nutzbare ILP-Modelle dar.
- Modelloptionen beschreiben Modifikationen des Basismodells (z.B. Austausch bestimmter Randbedingungen) und dienen der Performanceverbesserung oder Funktionserweiterung.
- Optimierungsziele definieren bevorzugte Lösungen und beeinflussen die Suchrichtung des ILP-Solvers.

Eine Übersicht über alle eingeführten Modellbestandteile findet sich in Anhang B.

Signifikante Performanceverbesserungen konnten in diesem Kapitel sowohl für Modelloptionen nachgewiesen werden, die aufgrund vorgenommener Einschränkungen des Lösungsraumes als *heuristisch* einzustufen sind, als auch für Modelloptionen, die *exakte* Modellvariationen darstellen. Bezüglich heuristischer Einschränkungen war beispielsweise die Begrenzung von Pfadlängen (*bpl*) sehr effektiv. Besonders erfolgreiche exakte Modelloptimierungen waren die Entfernung redundanter Ressourcenbedingungen (*rcr*) und das Runden von Konstanten zwecks ganzzahliger Schedulingbedingungen (*ia*). Im Falle des Multicastmodells leistet außerdem die weitgehende Entkopplung des Pfadschedulings von den Routingvariablen (*MCT*) einen weiteren Beitrag zur Performance. Sowohl *MCT* als auch *ia* nutzen dabei aus, dass sich durch angepasste Randbedingungen die LP-Relaxationen des Modells verbessern. Seitens der Optimierungsziele wurden *SPLT* für den optimierenden Fall und *1SP* für den nicht-optimierenden Fall als vorteilhaft identifiziert. In diesen Optimierungszielen wird ausgenutzt, dass die bevorzugte Suche nach Lösungen mit weniger aktiven Links zu einem schnelleren Auffinden initialer Lösungen führt.

Nach den bisherigen Erkenntnissen dieses Kapitels erfordern jedoch folgende Schwächen der gezeigten Modelle weitere Analysen und Optimierungen:

- Bei niedriger Netzwerkauslastung liegt eine problematische Skalierbarkeit vor (siehe Ergebnisse für TC-SSS/TC-TS), obwohl Zeitschlitzkonflikte nur eine untergeordnete Rolle spielen und das Problem daher durch einfache Heuristiken schnell lösbar wäre.
- Der Performanceunterschied zwischen Unicast- und Multicastmodell weist darauf hin, dass weiterer Optimierungsbedarf seitens der Multicastmodelle besteht.
- Trotz nachweisbarer Vorteile des JRaS-Ansatzes, insbesondere bei der Schedulability bei hoher Last, überwiegen auf das gesamte Benchmarking bezogen die Rechenzeitnachteile gegenüber Ansätzen mit separatem Routing (SRaS).

Diese Probleme werden im folgenden Kapitel aufgegriffen.

Kapitel 7

Performanceoptimierung und Funktionserweiterung der ILP-Modelle

Die bisher erstellten ILP-Modelle lösen das allgemein formulierte TT-RSP, berücksichtigen jedoch noch nicht das Queuingmodell von TSN (siehe Kapitel 2.2.3). Da die so erstellten Schedules beliebiges Reordering von Frames innerhalb einer Bridge erlauben, bleibt somit offen, inwiefern sich diese auch mit den begrenzten Reorderingmöglichkeiten von nur acht Queues realisieren lassen. Wie in der studentischen Arbeit [51] gezeigt, kann diese Frage in einer Nachbearbeitung beantwortet werden und zugleich eine passende Queuezuweisung erstellt werden, sofern diese existiert. Hier soll allerdings das ILP-Modell um passende Randbedingungen erweitert werden, um grundsätzlich TSN-konforme Schedules zu erstellen. Ein entsprechendes Modell ermöglicht darüber hinaus, die bei der Planung von TT-Streams zu verwendende Queueanzahl frei zu wählen und diese gegebenenfalls auch als Optimierungsziel zu minimieren (vgl. auch Qualitätsindikator $|\mathrm{ttq}|$ in Kapitel 4.1.1).

Da die entsprechenden Randbedingungen die Modellkomplexität erhöhen, sind für entsprechende Modelle höhere Rechenzeiten zu erwarten. Um diesem Problem zu begegnen und Benchmarkingzeiten möglichst gering zu halten, sollen noch vor der Formulierung der Queuingbedingungen weitere Modellvarianten mit dem Ziel der Performancesteigerung entworfen werden. In diesem Kapitel werden daher zuerst entsprechende Performanceoptimierungen vorgestellt und anschließend TSN-konforme Modellerweiterungen entworfen.

7.1 Performanceoptimierung der ILP-Modelle

Die bisherigen Analysen der verschiedenen ILP-Modellvarianten haben gezeigt, dass die Performance nicht alleine durch die Nutzung von ILP als Lösungsverfahren definiert wird, sondern dass die konkrete Modellierung von TT-RSP von großer Bedeutung ist. Dabei konnten einerseits optimierte Modellierungen herausgearbeitet werden, andererseits aber auch der hohe Preis eines komplexeren Modells gezeigt werden: Sowohl die gemeinsame Modellierung von Routing und Scheduling (JRaS) gegenüber separaten Berechnungsschritten (SRaS) als auch die Modellierung von Multicaststreams erhöhen die Rechenzeiten maßgeblich. Ein weiteres Problem, das sich durchgängig für alle Modell-

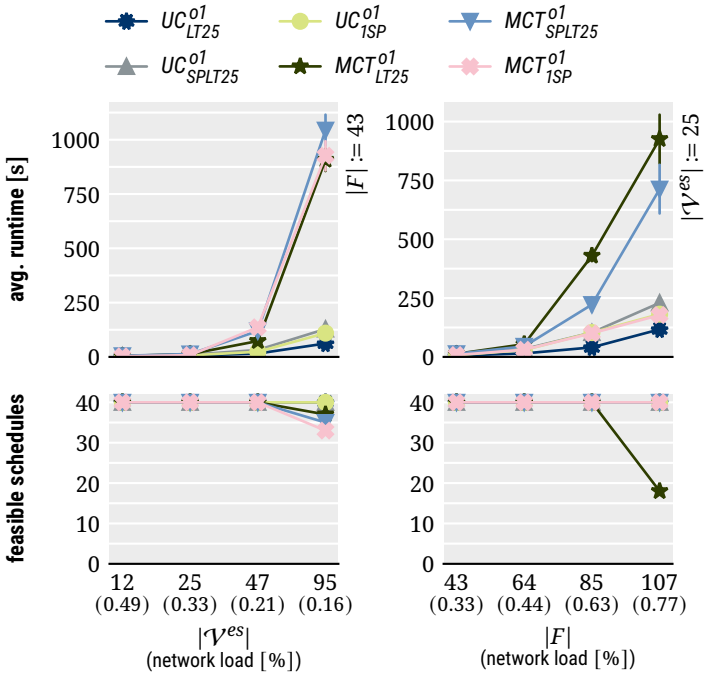


Abbildung 7.1: Ergebnisse für TC-TS (links) und TC-SSS (rechts) auf Basis der Meshtopologie für die optimierten Modellvarianten von *UC* und *MCT*. Parametersweep von $|\mathcal{V}^{es}|$ bzw. $|F|$. Konstante Szenarioparameter $ns := mesh$, $ss^{ct} := 400\,\mu s$, $ss^{fs} := 100\,B$ und $ss^{lf} := 6.0$.

varianten zeigt, ist eine schwache Skalierbarkeit bei niedriger Netzwerkauslastung. Dies zeigte sich beispielsweise in [E15] im Vergleich mit den ILP-Modellen einer anderen Arbeitsgruppe und sei hier noch einmal anhand der Testcases TC-SSS und TC-TS in Abbildung 7.1 verdeutlicht. Der Anstieg der Rechenzeiten ist hierbei offensichtlich wesentlich stärker ausgeprägt als proportional zur Problemgröße, obwohl Zeitschlitzkonflikte aufgrund der geringen Auslastung nur eine untergeordnete Rolle spielen und die Probleme daher grundsätzlich durch einfachste Heuristiken (z.B. SP-Routing gefolgt von ASAP-Scheduling) lösbar sind. In diesem Unterkapitel werden daher weitere Modellvarianten mit der folgenden Zielsetzung entwickelt:

- Verkleinerung des Rechenzeitunterschieds zwischen Multicast- und Unicastmodell
- Verbesserte Performance bei niedriger Netzwerkauslastung
- Verkleinerung des Rechenzeitunterschieds zwischen JRaS- und SRaS-Verfahren

Während die ersten beiden Ziele ohne Einschränkungen des Lösungsraumes verfolgt werden, wird im letzten Fall der Lösungsraum durch striktere Routenvorverarbeitung weiter eingeschränkt.

7.1.1 Multicast-Basismodell *MCS*

Um den Performanceunterschied zwischen Unicast- und Multicastmodell zu adressieren, wurde eine weitere Variante des Multicastmodells entworfen. Dieses nutzt die von *MCI*/*MCT* bereits bekannten Routingbedingungen (6.9)–(6.11), Anwendungsbedingungen (6.6), Ressourcenbedingun-

gen (6.7)/(6.8), und ist mit den bisher beschriebenen Modelloptionen und Optimierungszielen kompatibel. Der wesentliche Unterschied besteht, wie schon zwischen *MCI* und *MCT*, in den Bedingungen für das Scheduling entlang eines Pfades. Diese sind im Falle von *MCS* (von *multicast sum-based*) stärker an das Unicastmodell *UC* angelehnt, dessen Bedingungen für das Pfadscheduling hier daher noch einmal betrachtet werden.

$$\forall k \in F,\, \forall m \in E_k \qquad \boldsymbol{t}^{\boldsymbol{abs}}_{km} \;\leq\; M_{6.4} \cdot \mathfrak{p}_{km} \tag{6.4}$$

$$\forall k \in F,\, \forall i \in V^{br}_k \qquad \sum_{m \in E^{\rightarrow}_{ki}} \boldsymbol{t}^{\boldsymbol{abs}}_{km} \;-\; \sum_{m \in E^{\leftarrow}_{ki}} \boldsymbol{t}^{\boldsymbol{abs}}_{km} \;\geq\; \sum_{m \in E^{\leftarrow}_{ki}} d^{fwd\prime}_{km} \cdot \mathfrak{p}_{km} \tag{6.5}$$

Die Gleichung (6.5) modelliert den Anstieg der Sendezeitpunkte entlang der Route eines Streams mit nur einer Bedingung pro Bridgeknoten. Dabei wird ausgenutzt, dass im Falle von Unicastverbindungen an jedem Bridgeknoten nur je ein ausgehender und ein eingehender Link vom betrachteten Stream verwendet wird. Da für ungenutzte Links $\boldsymbol{t}^{\boldsymbol{abs}}_{km} = 0$ (nach (6.4)) und $\mathfrak{p}_{km} = 0$ gilt, ist somit in jeder Summenformel in (6.5) nur ein Element ungleich Null. Welches Element dies ist, hängt dabei vom aktuell gewählten Routing ab. Diese einfache Modellierung des Pfadschedulings ist im Falle von Multicasts nicht mehr ausreichend, da nun aufgrund von Verzweigungen für Multicast mehrere Ausgangslinks einer Bridge aktiv sein können. In (6.5) bedeutet dies, dass an einer solchen Verzweigung absteigende Sendezeitpunkte für die Ausgangslinks möglich wären, da in der ersten Summenformel zwei Elemente ungleich Null werden können und diese nur aufsummiert die eigentlich erforderliche Sendezeit ergeben müssen. Aufgrund dieses Fehlerfalls wurde (6.5) in den Multicastmodellen *MCI*/*MCT* durch eine paarweise Betrachtung ein- und ausgehender Links ersetzt (vgl. Alg. 6.1, 6.2). Im hier neu eingeführten Basismodell *MCS* werden (6.4) und (6.5) weiterhin genutzt und das Problem durch Ergänzung von (7.1) behoben.

$$\forall k \in F,\, \forall i \in V^{br}_k,\, \forall \vec{m} \in E^{\rightarrow}_{ki} \qquad \boldsymbol{t}^{\boldsymbol{abs}}_{k\vec{m}} \;-\; \sum_{m \in E^{\leftarrow}_{ki}} \boldsymbol{t}^{\boldsymbol{abs}}_{km} \;\geq\; \sum_{m \in E^{\leftarrow}_{ki}} d^{fwd\prime}_{km} \cdot \mathfrak{p}_{km} \;-\; M_{7.1} \cdot (1 - \mathfrak{p}_{k\vec{m}}) \tag{7.1}$$

Im Gegensatz zu (6.5) wird der Anstieg der Sendezeitpunkte hier für jeden ausgehenden Link in einer separaten Bedingung gefordert, sofern der jeweilige Ausgangslink aktiv ist. Für den Sendezeitpunkt des Eingangslinks wird dabei weiterhin auf die Summe aller Eingangslinks des betrachteten Knotens zurückgegriffen. Entsprechend dem Modell *MCI* führt diese Formulierung aufgrund der Abhängigkeit des Sendezeitanstiegs von den Pfadvariablen wieder zu potentiell negativen Streamlatenzen in der LP-Relaxation des Modells. Diese problematischen Bereiche des Lösungsraumes werden hier durch (7.2) ausgeschlossen.

$$\forall k \in F,\, \forall (n,m) \in E_k \times E_k \mid n_0 = f_k.src, m_1 \in f_k.dsts \qquad \boldsymbol{t}^{\boldsymbol{abs}}_{km} \;-\; \boldsymbol{t}^{\boldsymbol{abs}}_{kn} \;\geq\; lt^{i,sp\prime\prime}_{km} \tag{7.2}$$

Äquivalent zur Modelloption *isp* (Bedingung (6.12)) wird hierbei explizit und unabhängig von Pfadvariablen mindestens die Latenz des kürzesten Pfades gefordert. Einziger Unterschied zu (6.12) ist dabei, dass diese Forderung hier nur bezüglich der Links direkt an den Empfängern aufgestellt wird, während (6.12) dies für alle Kanten in E_k fordert. Diese Änderung ist hier erforderlich, da das Modell *MCS* auch Bedingung (6.4) nutzt, um den Sendezeitpunkt ungenutzter Links auf Null zu setzen, was

wiederum mit (6.12) in Konflikt stehen würde: Der Sendezeitpunkt auf ungenutzten Links kann nicht gleichzeitig Null und mindestens die Latenz des kürzesten Pfades betragen. Zusammengefasst besteht das Pfadscheduling von *MCS* aus (6.4), (6.5), (7.1) und (7.2).

7.1.2 Evaluation des Basismodells *MCS*

Das Modell *MCS* wurde unter Nutzung der bisherigen Performanceverbesserungen mit *MCT* und *UC* verglichen. Die Ergebnisse für den optimierenden Fall (Abb. 7.2) zeigen leichte Verbesserungen bei der Schedulability für *MCS* gegenüber *MCT*, sowie bedeutende Fortschritte bei den c.s.i.-Rechenzeiten. Für die Benchmarkingszenarien mit hoher Netzwerkauslastung konnte die c.s.i.-Rechenzeit für das im Allgemeinen besser funktionierende Optimierungsziel *SPLT25* um 30.73 % reduziert werden, für niedrige Netzwerkauslastung sogar um 57.86 %. Der verbleibende Rechenzeitnachteil gegenüber dem Unicastmodell liegt für dieses Optimierungsziel etwa bei einem Faktor von 2. Im nichtoptimierenden Fall (Abb. 7.3) sind die Unterschiede zwischen den Basismodellen grundsätzlich weniger ausgeprägt. Für Szenarien mit hoher Last kann *MCS* zwar eine geringfügig bessere Schedulability erzielen als *MCT*, allerdings fallen in diesen Fall die Rechenzeiten sogar etwas höher aus. Bei niedriger Last kann *MCS* dagegen deutliche Rechenzeitverbesserungen gegenüber *MCT* erzielen und befindet sich auf einem Niveau mit *UC*.

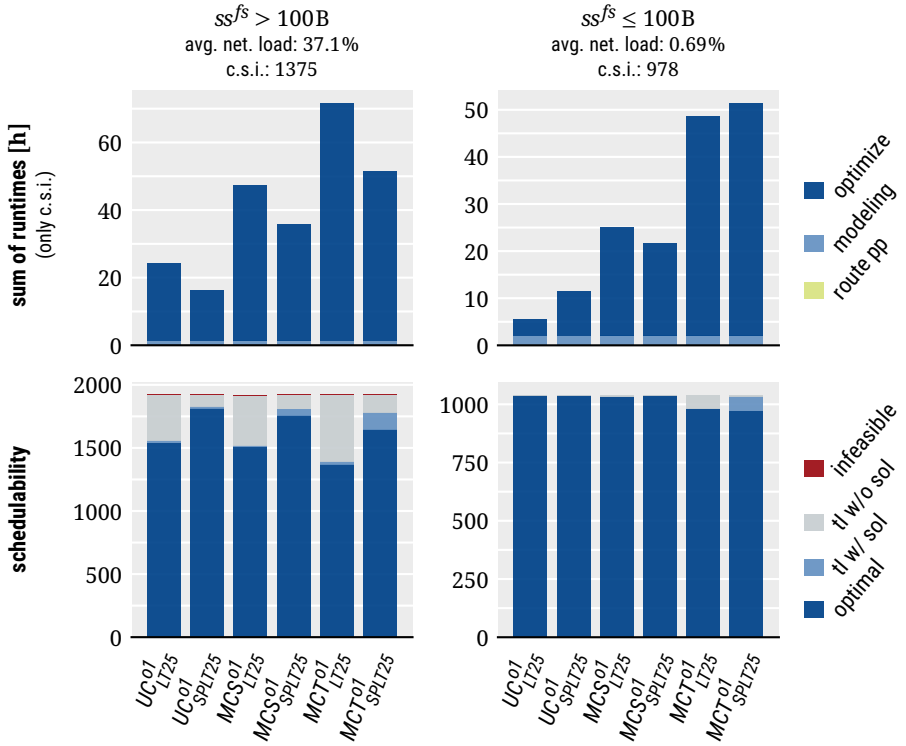


Abbildung 7.2: Vergleich von Schedulability und c.s.i.-Rechenzeiten zwischen den Basismodellen für den optimierenden Fall.

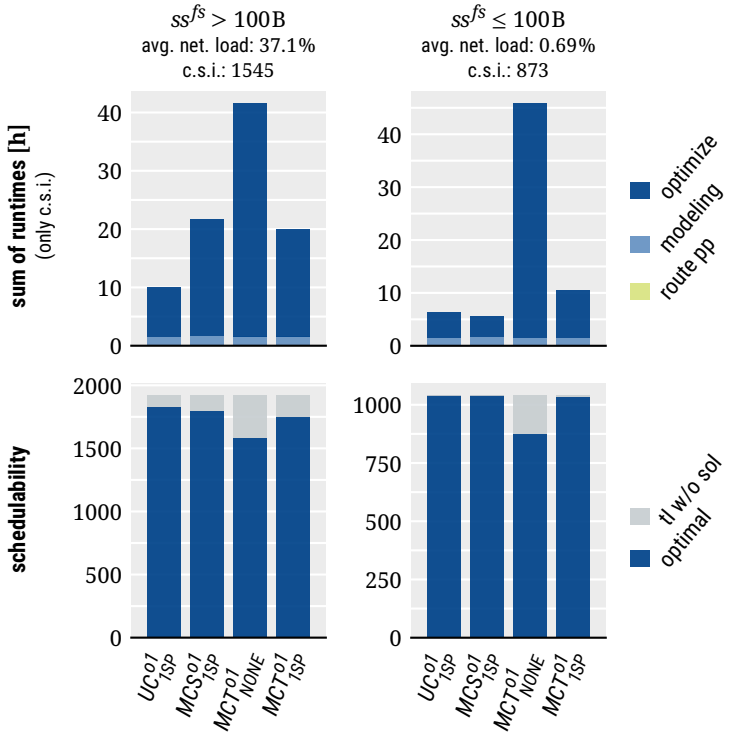


Abbildung 7.3: Vergleich von Schedulability und c.s.i.-Rechenzeiten zwischen den Basismodellen für den nichtoptimierenden Fall.

Insgesamt zeigen die Ergebnisse, dass die größere Übereinstimmung bezüglich der Randbedingungen das Multicastmodell *MCS* auch bei der Performance an das Modell *UC* angenähert hat. Konkrete, quantitative Unterschiede zwischen den drei Basismodellen sind jedoch sowohl vom gewählten Optimierungsziel als auch den Eingangsdaten abhängig. Während sich dies bei den hier gezeigten Ergebnissen bereits in den Unterschieden zwischen hoher und niedriger Netzwerkauslastung zeigt, sei auch noch einmal auf die Ergebnisse der einzelnen Testcases in [E19] verwiesen. Diese zeigen, dass die Abhängigkeit von den Eingangsdaten so weitreichend ist, dass sich je nach Testcase auch unterschiedliche Rankings der Scheduler ergeben können. Im Gegensatz zu vielen der bisher untersuchten Modelloptionen (z.B. die in *o1* zusammengefassten *bpl*, *rcr*, *ia*, *lu1*) gibt es also keine eindeutig dominierende Modellvariante. Dieses Ergebnis liegt dabei innerhalb der Erwartungen und ist mit den jeweiligen Veränderungen des Modells durch die verschiedenen Basismodelle und Modelloptionen erklärbar. Modelloptionen wie *rcr*, *ia* und *lu1* entfernen Redundanzen und reduzieren das im Lösungsverfahren erforderliche Branching eindeutig und lassen bereits aufgrund theoretischer Betrachtungen keine Nachteile im ILP-Solving erwarten. Ebenso trifft dies auf die Weiterentwicklung von *MCI* zu *MCT* zu. Die Modellunterschiede zwischen *MCT*, *MCS* und *UC* sind dagegen jeweils eine Umformulierung von Randbedingungen eines vieldimensionalen Lösungsraumes, bei denen eine Abschätzung der Auswirkungen auf ILP-Solving dagegen nicht trivial ist. Da die Ergebnisse keinen über alle Testcases hinweg dominierenden Scheduler zeigen, werden dementsprechend auch alle drei Modelle in den folgenden Analysen weiterverwendet.

7.1.3 Modellierung der Sendezeitpunkte (Modelloptionen *mod*, *modv*)

Als nächste mögliche Modellverbesserung soll die schwache Performance bei niedriger Netzwerkauslastung diskutiert werden. Ein grundsätzliches Problem beim ILP-Solving besteht darin, zu entscheiden, in welcher Reihenfolge verschiedene Teilbereiche des LP-Lösungsraumes nach ganzzahligen Lösungen durchsucht werden. Diese Reihenfolge ergibt sich aus den Branchingentscheidungen des Solvers, d.h. der Auswahl einer Variable für das Branching und der bevorzugten Suchrichtung nach dem Branching. Um zu verstehen, warum bestimmte Probleminstanzen zu einer unerwartet schlechten Performance führen, wäre es hilfreich, das Branching des Solvers und somit die (problematische) Suchrichtung analysieren zu können. Bei dem aus Performance- und Implementierungsgründen genutzten ILP-Solver Gurobi sind derartige Informationen nicht einsehbar. Daher lässt sich auch keine Aussage darüber treffen, welche Teilbereiche des LP-Lösungsraumes bei den bisher getesteten Modellen zuerst betrachtet werden, und warum diese Suchrichtung für das schnelle Auffinden von ganzzahligen Lösungen für Probleminstanzen mit niedriger Last ungünstig ist.

Daher wurde entschieden, als einen möglichen Einflussfaktor die Modellierung der Sendezeitpunkte durch ILP-Variablen zu untersuchen. Bisher wurde der Sendezeitpunkt von Stream k auf Link m durch den linearen Ausdruck $\boldsymbol{t}_{km}^{\boldsymbol{abs}} := \mathfrak{o}_{km} \cdot f_k.ct + \mathfrak{t}_{km}$ unter Verwendung der ganzzahligen Entscheidungsvariablen $0 \leq \mathfrak{t}_{km} \leq f_k.ct - 1$ und $0 \leq \mathfrak{o}_{km}$ beschrieben. In den Ressourcenbedingungen wird wiederum die Projektion des Sendezeitpunkts in den Wertebereich $[0, f_k.ct - 1]$ eingesetzt, welche nach der genannten Variablendefinition durch $\boldsymbol{t}_{km}^{\boldsymbol{mod}} := \mathfrak{t}_{km}$ erfasst werden kann. Die folgenden Modelloptionen beschreiben alternative Modellierungen der Sendezeitpunkte.

Modelloption *mod* Bei Aktivierung dieser Option wird *anstelle* von $\mathfrak{t}_{km}$ eine neue ganzzahlige Variable $\mathfrak{t}_{km}^{\mathfrak{a}}$ eingeführt, sowie die betroffenen linearen Ausdrücke $\boldsymbol{t}_{km}^{\boldsymbol{abs}}$, $\boldsymbol{t}_{km}^{\boldsymbol{mod}}$ neu definiert:

$$0 \leq \mathfrak{t}_{km}^{\mathfrak{a}} (\leq f_k.ct - 1 \text{ falls } m_0 = f_k.src) \qquad \boldsymbol{t}_{km}^{\boldsymbol{abs}} := \mathfrak{t}_{km}^{\mathfrak{a}} \qquad \boldsymbol{t}_{km}^{\boldsymbol{mod}} := \mathfrak{t}_{km}^{\mathfrak{a}} - \mathfrak{o}_{km} \cdot f_k.ct$$

Diese Formulierung erfordert zusätzlich die Bedingungen (7.3) und (7.4) zur korrekten Abbildung der *modulo*-Operation in den Wertebereich $[0, f_k.ct - 1]$.

$$\forall k \in F, \forall m \in E_k \qquad \boldsymbol{t}_{km}^{\boldsymbol{mod}} \geq 0 \tag{7.3}$$

$$\forall k \in F, \forall m \in E_k \qquad \boldsymbol{t}_{km}^{\boldsymbol{mod}} \leq f_k.ct - 1 \tag{7.4}$$

Die hier gezeigte neue Definition der linearen Ausdrücke wirkt sich bei der Modellbildung in allen Randbedingungen aus, in denen diese Ausdrücke eingesetzt werden (z.B. (6.5), (6.7), (6.8), usw.). Abgesehen von dieser Substitution kommen bei der Aktivierung von *mod* jedoch weiterhin die gleichen Randbedingungen des jeweiligen Modells zum Einsatz. Dementsprechend kann die Option auf jedes der bisher vorgestellten Basismodelle angewendet werden.

Der Modellierungsunterschied ohne bzw. mit der Option *mod* kann auch so beschrieben werden, dass sich die Komplexität bestimmter Randbedingungen verschiebt. Ohne die Option enthalten beispielsweise die Bedingungen für das Scheduling entlang eines Pfades durch $\boldsymbol{t}_{km}^{\boldsymbol{abs}}$ einen aus zwei Entscheidungsvariablen zusammengesetzten linearen Ausdruck, während die Projektion der Sendezeitpunkte $\boldsymbol{t}_{km}^{\boldsymbol{mod}}$ in den Ressourcenbedingungen durch nur eine Variable ausgedrückt werden kann. Mit der Option *mod* enthalten dagegen die Ressourcenbedingungen einen zusammengesetzten

Ausdruck, während sich das Pfadscheduling vereinfacht. Eine weitere Möglichkeit besteht darin, für beide Ausdrücke eine dedizierte Variable vorzusehen und diese über weitere Bedingungen aneinander zu koppeln.

Modelloption *modv* Bei dieser Option wird *zusätzlich* zu $\mathfrak{t}_{km}$ eine neue ganzzahlige Variable $\mathfrak{t}^{\mathfrak{a}}_{km}$ eingeführt, und ausschließlich $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$ neu definiert:

$$0 \leq \mathfrak{t}^{\mathfrak{a}}_{km} (\leq f_k.ct - 1 \text{ falls } m_0 = f_k.src) \qquad \boldsymbol{t}^{\boldsymbol{abs}}_{km} := \mathfrak{t}^{\mathfrak{a}}_{km}$$

Die Kopplung der neuen Variable für den absoluten Sendezeitpunkt an das aus dem ursprünglichen Modell vorhandene $\mathfrak{t}_{km}$ erfolgt durch (7.5).

$$\forall k \in F, \forall m \in E_k \qquad \mathfrak{t}_{km} = \mathfrak{t}^{\mathfrak{a}}_{km} - \mathfrak{v}_{km} \cdot f_k.ct \tag{7.5}$$

Eine konkrete Beschränkung auf den Wertebereich $[0, f_k.ct - 1]$ ist hier nicht erforderlich, da $\mathfrak{t}_{km}$ bereits laut Variablendefinition auf genau diesen Bereich beschränkt ist.

7.1.4 Evaluation der Modelloptionen *mod* und *modv*

Die Ergebnisse für die alternative Modellierung der Sendezeitpunkte mittels der Optionen *mod* und *modv* sind in Abbildung 7.4 für den nichtoptimierenden Fall dargestellt (Optimierungsziel *1SP*). Diese zeigen, dass insbesondere die angestrebte Verbesserung der Rechenzeiten für Szenarien mit niedriger Netzwerkauslastung erreicht wurde. Die konkrete Verbesserung unterscheidet sich dabei je nach Basismodell, wobei *UC* am stärksten profitiert und *MCS* am wenigsten. Die Fortschritte bei niedriger Last sind jedoch gegenläufig zum Verhalten bei hoher Netzwerkauslastung: Bei Aktivierung von *mod*/*modv* steigen die Rechenzeiten bei hoher Last für *UC* und *MCT*. Das bei niedriger Last weniger stark profitierende *MCS* zeigt dagegen auch bei hoher Last eine Verbesserung der Rechenzeiten. Stark ausgeprägte Unterschiede zwischen *mod* und *modv* zeigen sich nur im Falle von *MCT*, welches für *mod* eine wesentlich größere Verschlechterung der Rechenzeiten bei hoher Last zeigt. Interessant ist in dieser Auswertung auch die Verteilung der Streamlatenzen. Unter Nutzung der Optionen *mod*/*modv* weist ein Großteil der gefundenen Lösungen auch für den hier vorliegenden, nichtoptimierenden Fall deutliche bessere Streamlatenzen auf als die Lösungen der ursprünglichen Modelle. Dies ist darauf zurückzuführen, dass sich durch die alternative Modellierung der Sendezeitpunkte die Suchrichtung des Solvers verändert und dementsprechend andere Lösungen zuerst gefunden werden. Dies zeigt, dass die neu eingeführten Modelloptionen genau die zuvor diskutierte Problematik der Suchrichtung adressieren, welche als potentielles Problem für die Performance bei niedriger Last identifiziert wurde. Da die Modellveränderungen bezüglich der Sendezeitpunkte vorgenommen wurden, ist es auch plausibel, dass sich die Auswirkungen bei den Streamlatenzen zeigen.

In Kombination mit dem Optimierungsziel *SPLT25* (Abb. 7.5) wiederholen sich nicht nur die großen Vorteile bei niedriger Last, sondern auch die Ergebnisse bei hoher Last fallen besser aus. Hier zeigt neben *MCS* auch *MCT* eine Verbesserung der Rechenzeiten bei hoher Last, wohingegen die Aktivierung der Modelloptionen in Kombination mit *1SP* noch zu einer Verschlechterung führte. Im Falle von *UC* hat sich der zuvor beobachtete Nachteil bei hoher Last deutlich reduziert, sodass die Modellvarianten mit *mod*/*modv* hier nahezu gleichauf mit dem Ursprungsmodell liegen. Dieses bessere Abschneiden von

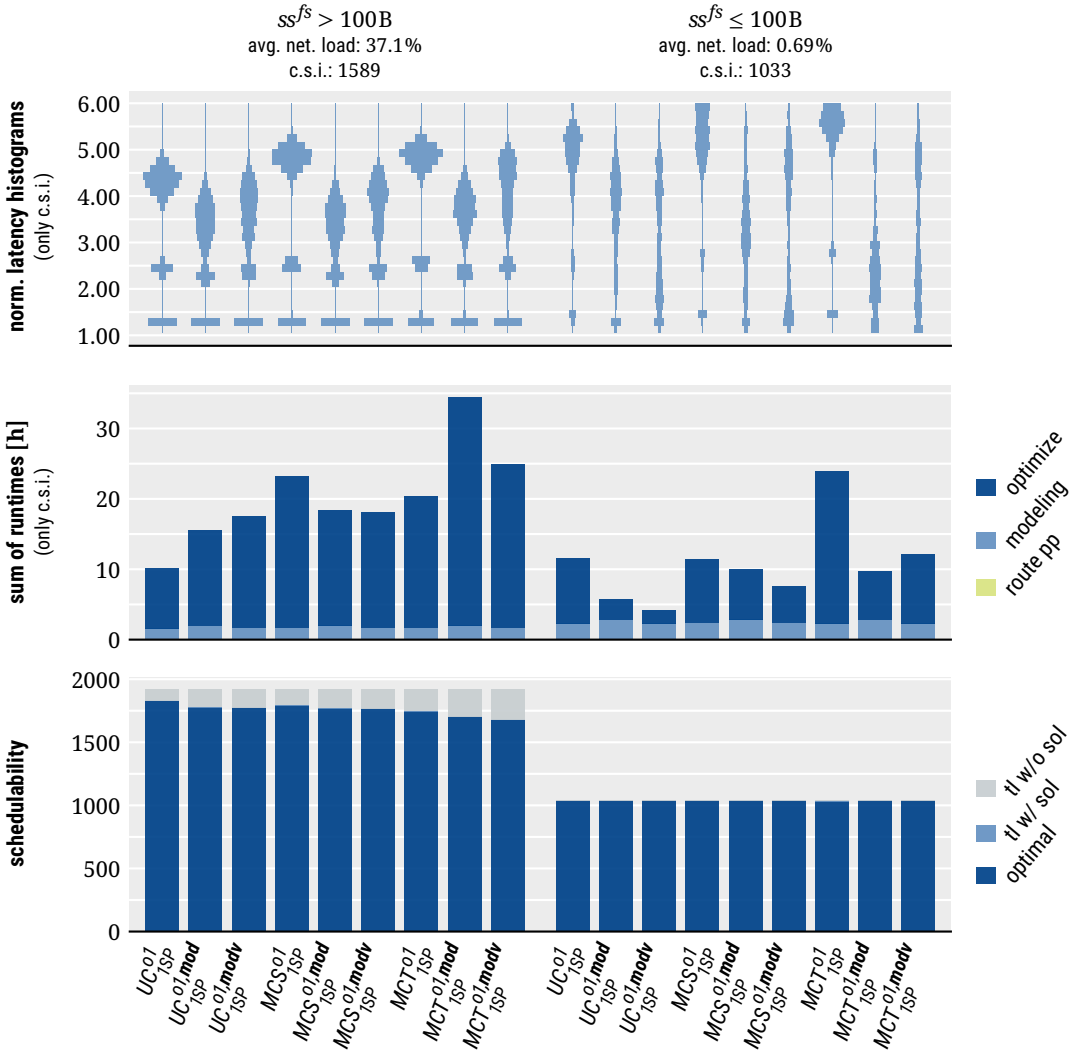


Abbildung 7.4: Schedulability, c.s.i.-Rechenzeiten und Latenzverteilung bei Aktivierung von *mod* bzw. *modv* für nichtoptimierende ILP-Modelle.

mod/*modv* bei Szenarien mit hoher Last im Vergleich zum nichtoptimierenden Fall ist intuitiv erklärbar: Bereits die Ergebnisse des nichtoptimierenden Falls haben gezeigt, dass bei Aktivierung der Optionen initiale Lösungen für Szenarien mit hoher Last zwar langsamer gefunden werden, diese jedoch in vielen Fällen eine wesentlich bessere Streamlatenz aufweisen. Die Optionen *mod*/*modv* beeinflussen demnach die Suchrichtung des Solvers zugunsten von Lösungen mit besseren Streamlatenzen. Sind diese nun durch das Optimierungsziel explizit gefordert, so sind diese Modellvarianten auch bezüglich der Rechenzeiten gegenüber der Modellierung ohne *mod*/*modv* im Vorteil, welche zuerst Lösungen mit schlechteren Streamlatenzen findet und daher einen längeren Optimierungsvorgang bis zum Erreichen der gewünschten Optimalität erfordert. Die gleichen Ergebnisse zeigen sich auch bei Verwendung des Optimierungsziels *LT25* (siehe [E19]).

Zusammengefasst überzeugen die Ergebnisse der Modelloptionen *mod*/*modv* bei niedriger Netzwerkauslastung grundsätzlich, während es bei hoher Netzwerkauslastung vom Basismodell und

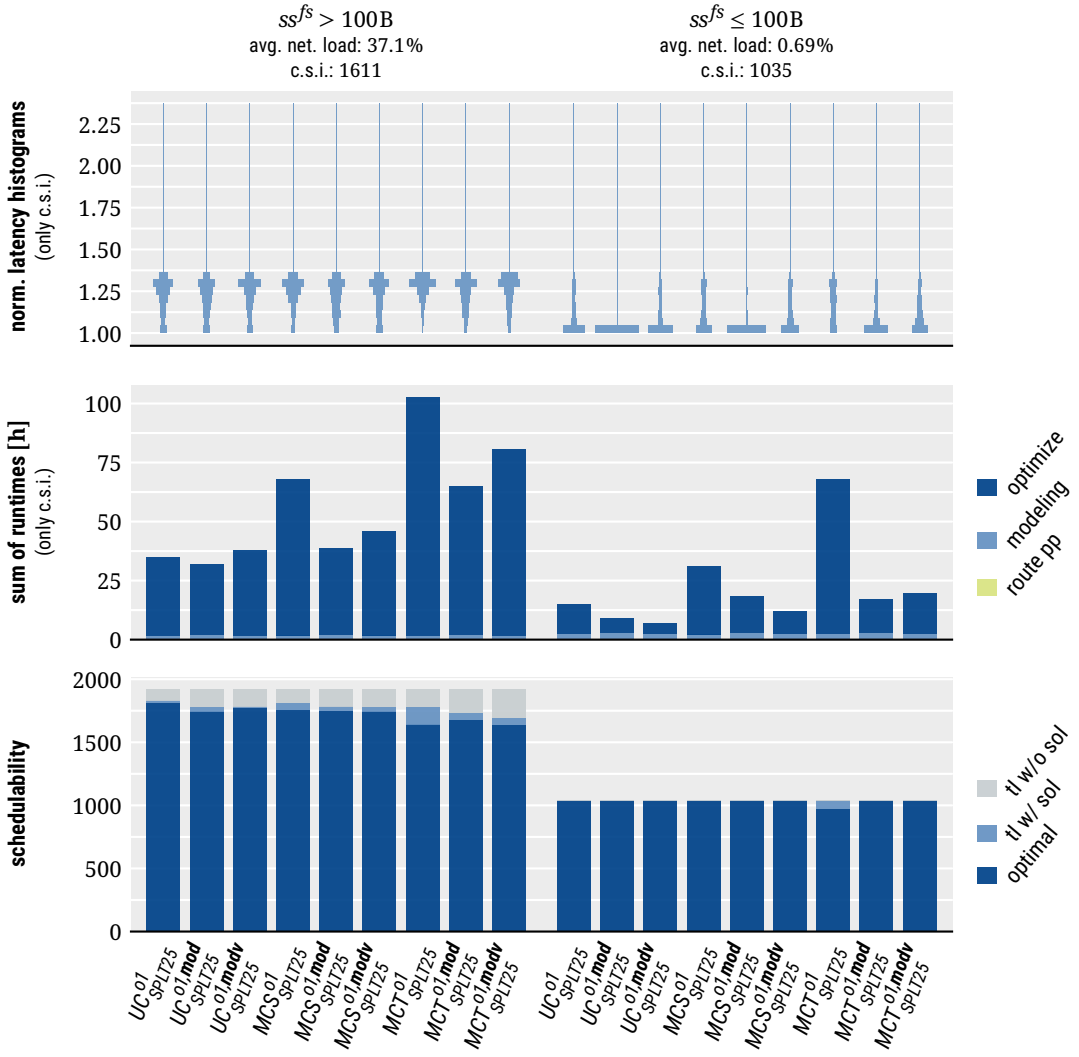


Abbildung 7.5: Schedulability, c.s.i.-Rechenzeiten und Latenzverteilung bei Aktivierung von *mod* bzw. *modv* bei Optimierungsziel *SPLT25*.

Optimierungsziel abhängt, ob die Optionen genutzt werden sollten. Die Unterschiede zwischen den beiden Optionen fallen mit Ausnahme des Basismodells *MCT* gering aus, wobei in der Regel *modv* zu bevorzugen ist. Für *UC* und *MCT* ist außerdem insbesondere im nichtoptimierenden Fall und bei hoher Last das Ursprungsmodell weiterhin führend. Für folgende Analysen werden auf Basis dieses Ergebnisses daher in der Regel die sechs Modellvarianten UC^{o1}, $UC^{o1,modv}$, MCS^{o1}, $MCS^{o1,modv}$, MCT^{o1} und $MCT^{o1,modv}$ berücksichtigt.

7.1.5 Striktere Routenvorverarbeitung (Modelloptionen *rpp-lbsp*, *rpp-ksp*)

Der Vergleich von JRaS- und SRaS-Verfahren in Kapitel 6.4 hat gezeigt, dass die gemeinsame Modellierung von Routing- und Schedulingentscheidungen zwar in wenigen Fällen zu einer besseren Schedulability und Lösungsgüte führen kann, dass dies jedoch im Allgemeinen zu einem massiven

Rechenzeitanstieg führt. Um diesen Nachteil zu adressieren, ist es daher naheliegend, eine strengere Routenvorverarbeitung anzuwenden. Die Zielsetzung ist dabei, zu Gunsten der Rechenzeiten wesentlich weniger Routingmöglichkeiten im ILP zu modellieren, während in den wenigen verbleibenden Routingoptionen insbesondere die relevanten Pfade enthalten sein sollen, die für die Vorteile bezüglich Schedulability und Lösungsgüte verantwortlich sind.

7.1.5.1 Kombiniertes Load Balanced/Shortest Path Routing (*rpp-lbsp*)

Die erste Variante der strikteren Routenvorverarbeitung kombiniert die bekannten Routingmethoden der SRaS-Verfahren: Load Balanced (LB)- und Shortest Path (SP)-Routing.

Modelloption *rpp-lbsp* Bei dieser Routenvorverarbeitung werden die beiden ILP-basierten Routingmethoden *r-nwsp-lu1-lta* (SP-Routing) und *r-lb-nwsp-lta* (LB-Routing) nacheinander ausgeführt, sodass zwei vollständige Routings für das Verkehrsmuster zur Verfügung stehen. Die im Scheduling-ILP genutzten, streamspezifischen Mengen V_k und E_k eines Streams k werden anschließend mit den Knoten bzw. Kanten befüllt, die sich aus der Vereinigung der beiden ermittelten Pfade ergeben. Dem resultierenden Schedulingmodell stehen somit für jeden Stream maximal zwei Pfade je Empfänger zur Auswahl.

Im Falle von Multicaststreams kann der Lösungsraum des Schedulingmodells dabei neue Pfadkombinationen für die unterschiedlichen Empfänger enthalten, da beispielsweise für einen Empfänger der mittels *r-nwsp-lu1-lta* ermittelte Pfad und für einen anderen Empfänger gleichzeitig der Pfad aus *r-lb-nwsp-lta* genutzt werden kann. Es sei außerdem angemerkt, dass sich bei dieser Form der Routenvorverarbeitung für einige Streams auch bereits feste Routen ergeben können, sofern die Ergebnisse von *r-nwsp-lu1-lta* und *r-lb-nwsp-lta* identisch sind.

Wie schon die zugrundeliegenden SRaS-Routings berücksichtigt *rpp-lbsp* bei aktiver Modelloption *bpl* nur die nach dieser ersten Vorverarbeitung verbleibenden Routen. Die genutzten ILP-Modelle der wiederverwendeten SRaS-Routings finden sich in Anhang C.2.

Die Grundidee der Modelloption *rpp-lbsp* ist es, dass die Kombination der beiden Routings eine ausreichende Routenauswahl bietet, da das SP-Routing bei hinreichend niedriger Netzwerkauslastung optimale Ergebnisse liefert, während das LB-Routing eine bestmöglich planbare Alternativroute für stark ausgelastete Bereiche einer Netzwerktopologie bietet.

$\mathcal{P}_{ijs}^{\mathrm{kSP}\prime<} := ()$ // an empty sequence
for $n := 0$ **to** $\min\left(p^{max}, \left|\mathcal{P}_{ijs}^{\mathrm{kSP}\prime}\right|\right) - 1$ **do** // sequence indexes start at 0
  $P := \mathcal{P}_{ijs_n}^{\mathrm{kSP}\prime}$ // select the n^{th} path of $\mathcal{P}_{ijs}^{\mathrm{kSP}\prime}$
  if $n < p^{min}$ **or** $\left(\mathrm{LT}(P,s) \leq cut^{lt,rel} \cdot lt_{ijs}^{i,sp\prime}\right.$ **and** $\left.\mathrm{PL}(P) \leq cut^{pl,abs}\right)$ **then**
    append P to $\mathcal{P}_{ijs}^{\mathrm{kSP}\prime<}$
  if $\mathrm{LT}(P,s) \geq cut^{lt,rel} \cdot lt_{ijs}^{i,sp\prime}$ **and** $n+1 \geq p^{min}$ **then**
    break // stop loop iteration

Algorithmus 7.1: Erzeugung von $\mathcal{P}_{ijs}^{\mathrm{kSP}\prime<}$ für Sender i, Empfänger j und Framegröße s. Die Operationen $\mathrm{LT}(P,s)$ und $\mathrm{PL}(P)$ ermitteln die ideale Latenz bzw. die Pfadlänge (Kantenanzahl) des Pfades P für einen Frame der Größe s.

7.1.5.2 k Shortest Path Routing (*rpp-ksp*)

Eine weitere Variante der Routenvorverarbeitung vertieft das Konzept der Modelloption *bpl*, in der für jeden Stream alle einfachen Pfade gebildet wurden, wobei die maximale Pfadlänge (bezüglich Hopanzahl) sowohl absolut als auch relativ zum kürzesten Pfad begrenzt wurde.

Modelloption *rpp-ksp* Bei dieser Modelloption werden die Knoten- und Kantenmengen V_k bzw. E_k eines Streams erstellt, indem für jeden Empfänger des Streams k zunächst ein *k Shortest Path* (kSP)-Algorithmus ausgeführt wird.[1] Aus diesen kürzesten Pfaden wird nach weiteren Selektionsparametern (z.B. Cutoffs) eine Untermenge an Pfaden für jeden Empfänger erstellt, bevor die Mengen V_k bzw. E_k als Vereinigung der Knoten bzw. Kanten aus allen vorselektierten Pfaden aller Empfänger des Streams k gebildet werden. Das konkrete Vorgehen wird im Folgenden beschrieben.

Entsprechend dem Ergebnis eines kSP-Algorithmus sei $\mathcal{P}_{ijs}^{\text{kSP}\prime}$ eine Sequenz der möglichen Pfade von Sender i zu Empfänger j für einen Frame der Größe s, geordnet nach aufsteigender idealer Streamlatenz (d.h. mit Bridge- und Leitungsverzögerung, aber ohne Queueing; vgl. auch lt^{norm} in Kapitel 4.2.2.2). Außerdem ist durch p^{min} und p^{max} die Mindestanzahl bzw. Höchstanzahl an Pfaden gegeben, die bei der Modellbildung für einen Empfänger berücksichtigt werden sollen. Des Weiteren sind die Cutoffs $cut^{pl,abs}$ und $cut^{lt,rel}$ gegeben, welche die maximale Kantenanzahl bzw. die höchste Latenz relativ zur idealen Latenz des kürzesten Pfades $lt_{ijs}^{i,sp\prime}$ spezifizieren, damit ein Pfad als zulässig gilt. Auf Basis dieser Parameter wird entsprechend Algorithmus 7.1 eine Sequenz $\mathcal{P}_{ijs}^{\text{kSP}\prime<}$ erzeugt, welche die bei der Modellbildung zu berücksichtigenden Pfade für (i, j, s) enthält.

Die Definition von $\mathcal{P}_{ijs}^{\text{kSP}\prime<}$ besagt, dass höchstens die p^{max} kürzesten Pfade aus $\mathcal{P}_{ijs}^{\text{kSP}\prime}$ betrachtet werden. Hiervon werden die p^{min} kürzesten Pfade ohne weitere Bedingungen in $\mathcal{P}_{ijs}^{\text{kSP}\prime<}$ aufgenommen. Von allen weiteren Pfade werden nur solche zu $\mathcal{P}_{ijs}^{\text{kSP}\prime<}$ hinzugefügt, die die gegebenen Cutoffs bezüglich Latenz und Kantenanzahl erfüllen. Sofern eine Topologie weniger als p^{min} Pfade von i nach j bietet, kann $\mathcal{P}_{ijs}^{\text{kSP}\prime<}$ dabei auch weniger als p^{min} Pfade enthalten. Nach der Definition der zulässigen Pfade für ein gegebenes (i, j, s) kann darauf basierend eine Sequenz $\mathcal{P}_{kj}$ definiert werden, welche die zulässigen Pfade für einen Empfänger $j \in f_k.dsts$ des Streams k enthält. Nach Algorithmus 7.2 enthält $\mathcal{P}_{kj}$ alle Pfade aus $\mathcal{P}_{ijs}^{\text{kSP}\prime<}$ (für $i := f_k.src, s := f_k.fs$), welche darüber hinaus die Latenzanforderung $f_k.ml$ erfüllen. Aus den Pfadsequenzen $\mathcal{P}_{kj}$ können abschließend

$$E_k := \bigcup_{j \in f_k.dsts} \bigcup_{P \in \mathcal{P}_{kj}} P \qquad \text{und} \qquad V_k := \bigcup_{m \in E_k} \{m_0, m_1\}$$

abgeleitet werden.[2] Es handelt sich hierbei um die Vereinigung aller Kanten bzw. Knoten, die in den vorselektierten Pfaden des jeweiligen Streams enthalten sind. Wie bekannt werden diese Mengen in der Vorverarbeitung ermittelt und beeinflussen die gesamte darauf aufbauende Modellbildung des Scheduling-ILPs und sind daher gleichermaßen für alle Basismodelle anwendbar.

Die Parameter p^{min} und p^{max} sind wählbar und werden im Folgenden als Suffix der Modelloption angegeben. Dementsprechend bedeutet *rpp-ksp-1-5* beispielsweise, dass *rpp-ksp* mit $p^{min} := 1$ und $p^{max} := 5$ genutzt wurde. Die Werte von $cut^{pl,abs}$ und $cut^{lt,rel}$ werden wie schon für *bpl* aus den

[1] Leider konnte an dieser Stelle der Namenskonflikt für k bzw. *k* nicht vermieden werden. In mathematischen Formulierungen bezieht sich k im Folgenden wie auch bisher immer auf Stream-IDs und hat keinen Bezug zur Pfadanzahl des kSP-Algorithmus.

[2] Obwohl $\mathcal{P}_{kj}$ hier als *Sequenz* definiert wurde, wird hier die für *Mengen* übliche Notation $P \in \mathcal{P}_{kj}$ zur Iteration über die Elemente der Sequenz verwendet.

$\mathcal{P}_{kj} := ()$ // an empty sequence
$i := f_k.src$
$s := f_k.fs$
foreach $P \in \mathcal{P}_{ijs}^{\text{kSP}'<}$ **do** // assume that iteration happens in order of $\mathcal{P}_{ijs}^{\text{kSP}'<}$
  if $\text{LT}(P, s) \leq f_k.ml$ **then**
    append P to $\mathcal{P}_{kj}$

Algorithmus 7.2: Erzeugung von $\mathcal{P}_{kj}$ für Stream k und einen Empfänger $j \in f_k.dsts$.

Topologiedaten entnommen. Während $cut^{pl,abs}$ daher auch hier von der Topologiegröße abhängig ist, gilt für die genutzten Benchmarkingszenarien für $cut^{lt,rel}$ ein Wert von 1 für $fattree$-Topologien und ein Wert von 3 für $ring$ und $mesh$. Die im folgenden Unterkapitel evaluierte Implementierung von *rpp-ksp* nutzt einen kSP-Algorithmus der Pythonbibliothek `NetworkX` zur Bestimmung von $\mathcal{P}_{ijs}^{\text{kSP}'}$ für ein gegebenes (i, j, s). Dabei wird $\mathcal{P}_{ijs}^{\text{kSP}'}$ ausschließlich für Kombinationen von (i, j, s) erstellt, die im vorliegenden Verkehrsmuster auch tatsächlich vorkommen. Außerdem werden die nach Algorithmus 7.1 gefilterten Pfadsequenzen $\mathcal{P}_{ijs}^{\text{kSP}'<}$ stets zwischengespeichert, sodass der kSP-Algorithmus für ein bestimmtes (i, j, s) immer nur einmal ausgeführt wird, auch wenn im Verkehrsmuster mehrere Streams mit der entsprechenden Kombination aus Sender, Empfänger und Framegröße existieren. Darüber hinaus wird der kSP-Algorithmus grundsätzlich nicht über die ersten p^{max} Pfade hinaus ausgeführt. Die kSP-Implementierung von `NetworkX` basiert auf Yens Algorithmus [52], welcher für die Bestimmung von p^{max} Pfaden in einer Topologie mit $|\mathcal{V}|$ Knoten eine Komplexität von $O(p^{max} \cdot |\mathcal{V}|^3)$ aufweist. Alternative oder optimierte kSP-Implementierungen wurden nicht weiter betrachtet, da die Routenvorverarbeitung in der Regel wenig Rechenzeit im Vergleich zum anschließenden Lösen des Schedulingmodells in Anspruch nimmt. Es sei außerdem angemerkt, dass *rpp-ksp* die Modelloption *bpl* im Gegensatz zu den ILP-basierten Routenvorverarbeitungen ignoriert, sodass im Falle beider aktiven Optionen (z.B. $UC^{o1,\text{rpp-ksp}}$, da *o1* auch *bpl* enthält) die Option *bpl* keinen Effekt hat.

7.1.5.3 Explizite Pfadselektion auf Basis von *rpp-ksp*

In der Beschreibung von *rpp-ksp* ist ersichtlich, dass in der Routenvorverarbeitung zunächst konkrete Pfade für jeden Empfänger eines jeden Streams gebildet werden. Die Zusammenführung der Pfade in der Kantenmenge E_k und die darauf aufbauende ILP-Modellierung haben jedoch zur Folge, dass die bereits bekannten Pfade niemals explizit im ILP-Modell formuliert werden. Stattdessen müssen gültige Pfade im ILP-Solving anhand der Routingbedingungen wieder aus den gegebenen Kantenmengen zusammengesetzt werden. Alternativ ist es möglich, die bereits aus der Routenvorverarbeitung bekannten Pfade (gegeben in $\mathcal{P}_{kj}$ nach Alg. 7.2) explizit zu modellieren und den ILP-Solver mittels entsprechender Entscheidungsvariablen nur noch zwischen diesen bekannten Pfaden wählen zu lassen. Diesbezüglich wurden zwei mögliche Formulierungen entworfen.

Modelloption *psel-pa* Sei $p_{kj}^{no} := |\mathcal{P}_{kj}|$ die Anzahl der zur Verfügung stehenden Pfade für Empfänger j des Streams k. Des Weiteren sei $P_{km}^{sel} := \{(j,n) \in f_k.dsts \times \mathbb{N}^0 \mid n < p_{kj}^{no} \wedge m \in \mathcal{P}_{kj_n}\}$ eine Menge von Tupeln (j,n), die alle zum Stream k gehörigen Pfade identifiziert, die den Link m enthalten. Bei Nutzung der Modelloption *psel-pa* (von *path selection, path activation*) werden dem ILP-Modell binäre Entscheidungsvariablen $\mathfrak{P}_{kjn}$ hinzugefügt, wobei $\mathfrak{P}_{kjn} = 1$ anzeigt, dass der Empfänger j des Streams k über den Pfad $\mathcal{P}_{kj_n}$ geroutet wird.[3] Mit Hilfe der neu eingeführten Variablen und den in der Routenvorverarbeitung ermittelten Pfaden $\mathcal{P}_{kj}$ lässt sich ein gültiges Routing durch (7.6) und (7.7) ausdrücken.

$$\forall k \in F, \forall j \in f_k.dsts \qquad \sum_{n \in \mathbb{N}^0 \mid n < p_{kj}^{no}} \mathfrak{P}_{kjn} = 1 \tag{7.6}$$

$$\forall k \in F, \forall j \in f_k.dsts, n \in \mathbb{N}^0 \mid n < p_{kj}^{no} \qquad \sum_{m \in \mathcal{P}_{kj_n}} \mathfrak{p}_{km} \geq |\mathcal{P}_{kj_n}| \cdot \mathfrak{P}_{kjn} \tag{7.7}$$

$$\forall k \in F, \forall m \in E_k \qquad \mathfrak{p}_{km} \leq \sum_{(j,n) \in P_{km}^{sel}} \mathfrak{P}_{kjn} \tag{7.8}$$

Dabei besagt (7.6), dass von den p_{kj}^{no} bekannten Pfaden für Empfänger j des Streams k genau einer aktiv sein muss. Die Gleichung (7.7) sorgt dafür, dass die zugehörigen Pfadvariablen $\mathfrak{p}_{km}$ der einzelnen Links des aktiven Pfades ebenfalls aktiviert werden. Darüber hinaus wird (7.8) eingesetzt, um alle Pfadvariablen, die mit keiner aktiven Route assoziiert sind, auf Null zu setzen. Bei Nutzung der Modelloption *psel-pa* werden die genannten Bedingungen dem ILP-Modell *anstelle* der regulären Routingbedingungen ((6.1), (6.2), (6.9), (6.10), (6.11)) hinzugefügt.

Modelloption *psel-la* Die Modelloption *psel-la* (von *path selection, link activation*) ist äquivalent zu *psel-pa*, mit Ausnahme der expliziten Routenmodellierung durch (7.9) anstelle von (7.7).

$$\forall k \in F, \forall m \in E_k \qquad |f_k.dsts| \cdot \mathfrak{p}_{km} \geq \sum_{(j,n) \in P_{km}^{sel}} \mathfrak{P}_{kjn} \tag{7.9}$$

Nach (7.9) muss eine Pfadvariable $\mathfrak{p}_{km}$ immer dann den Wert Eins annehmen, wenn eine beliebige Route des Streams k aktiviert wird, die den Link m nutzt. Die Multiplikation mit $|f_k.dsts|$ ist erforderlich, da die explizite Routenmodellierung durch $\mathfrak{P}_{kjn}$ separat für jeden Empfänger $j \in f_k.dsts$ erfolgt und dementsprechend viele der durch $\mathfrak{P}_{kjn}$ referenzierten Pfade gleichzeitig aktiv sein können. Wie schon für *psel-pa* werden die hier genannten Bedingungen anstelle der regulären Routingbedingungen genutzt.

Die Modelloptionen *psel-pa*/*psel-la* realisieren eine explizite Routenmodellierung auf Basis der in $\mathcal{P}_{kj}$ gegebenen Pfade, wobei $\mathcal{P}_{kj}$ hier im Rahmen von *rpp-ksp* definiert wurde. Auch wenn *psel-pa*/*psel-la* im Folgenden nur in genau dieser Form (d.h. Verwendung von *rpp-ksp*) evaluiert wurden, sei abschließend darauf hingewiesen, dass die gezeigte explizite Routenmodellierung grundsätzlich mit beliebigen anderen Methoden zur Routenvorverarbeitung einsetzbar ist, sofern diese die entsprechenden Pfadsequenzen $\mathcal{P}_{kj}$ zur Verfügung stellen.

[3] Die doppelte Indizierung in $\mathcal{P}_{kj_n}$ ist beabsichtigt und referenziert den n-ten Pfad der Sequenz $\mathcal{P}_{kj}$.

7.1.5.4 Zusammenfassung der Routenvorverarbeitungsschritte

Aufgrund der Vielzahl verschiedener Modelloptionen zwecks Routenvorverarbeitung und Routing soll vor der Evaluation der neu eingeführten Methoden ein kurzer Gesamtüberblick über alle derartigen Modelloptionen sowie der möglichen Interaktion von diesen gegeben werden. Der folgende Überblick ist entsprechend dem Programmablauf beim Lösen einer Probleminstanz dargestellt.

1. *Routenvorverarbeitung nach bpl*
   - `NetworkX`-basierte Selektion aller schleifenfreien Pfade unter Einhaltung von Cutoffs
   - Erstellt Kantenmenge E_k und Knotenmenge V_k für alle Streams
   - Wird bei aktiviertem *rpp-ksp* übersprungen
2. *Routenvorverarbeitung nach rpp-* oder Routing nach r-**

   rpp-lbsp
   Nutzt E_k/V_k aus *bpl* und führt einen zweiten Schritt der Routenvorverarbeitung durch: Zweifache Routenermittlung mittels ILP, gefolgt von erneuter Erstellung von E_k/V_k

   rpp-ksp
   Routenvorverarbeitung mittels eines kSP-Algorithmus: Stellt für jedem Stream k und Empfänger j mehrere explizite Pfade $\mathcal{P}_{kj}$ zur Verfügung, sowie direkt daraus abgeleitete Mengen E_k/V_k

   *r-**
   Nutzt E_k/V_k aus *bpl* und führt eine konkrete Routenberechnung mittels ILP durch (SRaS): Erneute Erstellung von E_k/V_k, wobei diese je Stream auf eine einzige konkrete Route beschränkt sind
3. *Modellbildung des Scheduling-ILPs*
   - Modellbildung des regulären Scheduling-ILPs in Abhängigkeit vom gewählten Basismodell und aktiven Modelloptionen
   - Nutzt das zuletzt erstellte E_k/V_k, sodass nur die darin enthaltenen Routen bei der Planung gebildet werden können
   - In Kombination mit *rpp-ksp* können die Modelloptionen *psel-pa*/*psel-la* genutzt werden, um Randbedingungen zur expliziten Routenmodellierung anstelle der regulären Routingbedingungen zu verwenden

In der Evaluation wird die Rechenzeit von *bpl*, *rpp-lbsp* und *rpp-ksp* vollständig dem Schritt *route pp* zugeschrieben, während SRaS-Optionen (*r-**) als *routing* deklariert werden. Die Schritte *modeling* und *optimize* enthalten unabhängig von den Routenvorverarbeitungsschritten nur die Modellbildung bzw. das Solving des Scheduling-ILPs.

7.1.6 Evaluation der strikteren Routenvorverarbeitung

Die neuen Varianten der Routenvorverarbeitung sollen durch einen Vergleich mit der einfachen Vorverarbeitung (*bpl*, hier enthalten in *o1*) und den beiden SRaS-Verfahren *r-nwsp-lu1-lta*/*r-lb-nwsp-lta* bewertet werden. Die Evaluation wurde für alle drei Basismodelle und die beiden Optimierungsziele *SPLT25*/*1SP* durchgeführt, wobei Abbildung 7.6 die Ergebnisse für $MCS_{SPLT25}^{o1,modv,stl}$ zusammenfasst. Insbesondere *rpp-lbsp* zeigt dabei herausragende Ergebnisse, welche mit den Entwicklungszielen dieser Vorverarbeitung übereinstimmen: Die Rechenzeiten entsprechen nahezu den SRaS-Verfahren,

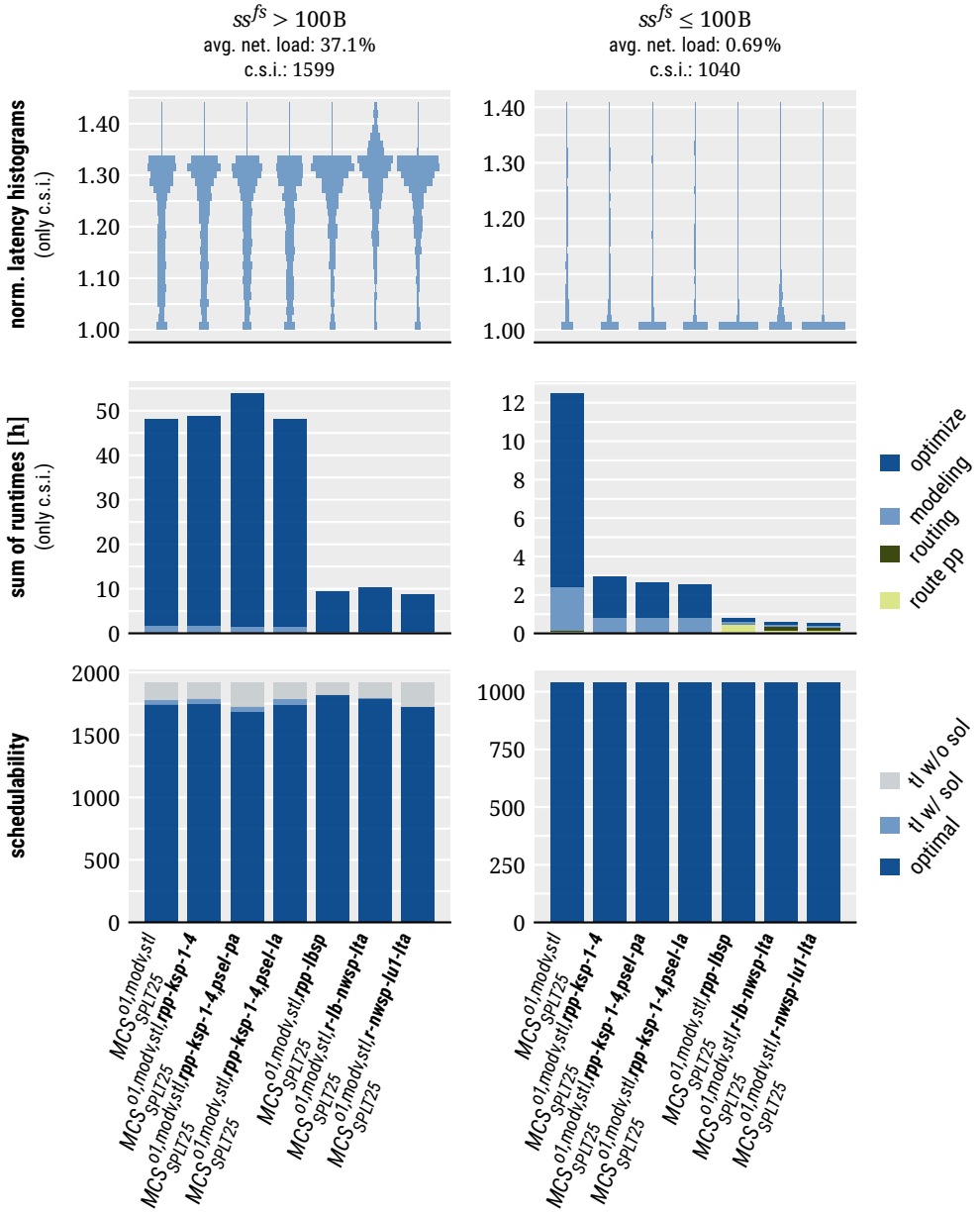


Abbildung 7.6: Schedulability, c.s.i.-Rechenzeiten und Latenzverteilung verschiedener Routenvorverarbeitungen für $MCS^{o1,modv,stl}_{SPLT25}$.

während die Schedulability führend unter allen getesteten Verfahren ist und die Streamlatenzen im Gegensatz zum festen LB-Routing (*r-lb-nwsp-lta*) kaum Nachteile gegenüber dem komplexeren Modell $MCS^{o1,modv,stl}_{SPLT25}$ aufweisen. Die gegenüber $MCS^{o1,modv,stl}_{SPLT25}$ gesteigerte Schedulability bei hoher Netzwerkauslastung liegt dabei an den wesentlich geringeren Rechenzeiten, welche zu mehr gelösten Szenarien innerhalb des Zeitlimits führen. Im Gegensatz zu den SRaS-Verfahren bieten die wenigen modellierten Routen (maximal zwei) dabei weiterhin ausreichend Optionen, um die Schedulability nicht negativ zu beeinträchtigen.

Die drei getesteten Konfigurationen von *rpp-ksp* sind im Vergleich zu *rpp-lbsp* weniger restriktiv bezüglich der Routen, was dazu führt, dass diese Modellvarianten bei hoher Netzwerkauslastung keine Vorteile gegenüber dem Referenzmodell $MCS_{SPLT25}^{o1,modv,stl}$ zeigen. Dies ist darauf zurückzuführen, dass die genutzten Routen und resultierenden Schedulingmodelle sich in diesem Fall nicht maßgeblich vom Referenzmodell unterscheiden. Eine detailliertere Analyse ist auf Basis der Daten aus [E19] möglich, da die gespeicherten Ergebnisse auch die Kantenmengen E_k der Routenvorverarbeitung enthalten und somit ein Vergleich der zur Verfügung stehenden Routen für *bpl* und *rpp-ksp* möglich wäre. Aufgrund des begrenzten Umfangs dieser Arbeit wurde eine solche Analyse jedoch nicht durchgeführt. Bei niedriger Netzwerkauslastung zeigen auch die Modellvarianten von *rpp-ksp* deutliche Vorteile gegenüber der Referenz, wobei aufgrund der weniger restriktiven Vorverarbeitung nicht die Ergebnisse von *rpp-lbsp* erreicht werden. Das bessere Abschneiden im Vergleich zur hohen Netzwerkauslastung ist hier insbesondere darauf zurückzuführen, dass die Benchmarkingszenarien für niedrige Last überwiegend größere Netzwerktopologien nutzen, in denen die Vorverarbeitung mit den gewählten Parametern von *rpp-ksp* zu größeren Unterschieden gegenüber $MCS_{SPLT25}^{o1,modv,stl}$ führt, als in den kleineren Topologien der Szenarien mit hoher Last.

Im Allgemeinen sind die erzielten Ergebnisse verschiedener Routenvorverarbeitungen bedeutend von den genutzten Netzwerktopologien abhängig, wobei die in den Benchmarkingszenarien genutzten Ring- und Meshtopologien eine strikte Routenvorverarbeitung begünstigen. In Topologien mit einer höheren Anzahl von Pfaden ähnlicher Länge (z.B. $fattree$) können sich dagegen Vorteile einer weniger restriktiven Vorverarbeitung zeigen, wobei hier vor allem *rpp-ksp* durch die topologiespezifische Anpassung seiner Parameter vorteilhaft sein kann. Dies wird in Kapitel 8.1 im Rahmen des Multicast-Benchmarkings aufgegriffen, da die dort genutzten Szenarien auch $fattree$-Topologien nutzen. Im Verlauf der folgenden Tests wird jedoch zunächst vorwiegend *rpp-lbsp* eingesetzt, da diese Modellvariante im Unicast-Benchmarking die besten Ergebnisse erzielt und somit insbesondere zu einer reduzierten Benchmarkingdauer beiträgt. Die herausragenden Ergebnisse von *rpp-lbsp* wurden auch für andere Basismodelle und das Optimierungsziel *1SP* bestätigt (siehe [E19]). In diesen weiteren Ergebnissen fällt außerdem auf, dass sich die Performanceunterschiede zwischen den verschiedenen Basismodellen reduzieren und bei Nutzung einer strengen Routenvorverarbeitung kaum noch vorhanden sind. Dieser Effekt ist anhand theoretischer Überlegungen nachvollziehbar: Die Basismodelle unterscheiden sich in den Routingbedingungen und dem Pfadscheduling, wobei sich die unterschiedlichen Gleichungen bei Vorhandensein von nur einem ausgehenden und eingehenden Link an einer Bridge (was bei strenger Routenvorverarbeitung häufig vorliegt) in vielen Fällen auf die gleiche Gleichung reduzieren.

7.2 Funktionserweiterungen bezüglich TSN

Die in Kapitel 3 unter der Bezeichnung TT-RSP eingeführte Problemstellung fordert das Auffinden von Routen und konfliktfreien Zeitschlitzen für alle Streams eines gegebenen Verkehrsmusters. Diese Problemstellung sowie die bisher diskutierten Lösungsverfahren sind neben TSN auch für weitere Netzwerktechnologien mit zeitgesteuerten Sendemechanismen (z.B. Profinet, TTEthernet)

relevant. Dementsprechend wurde bei den bisherigen ILP-Modellen davon abgesehen, den Lösungsraum aufgrund von TSN-spezifischen Mechanismen weiter einzuschränken. In den bisher gezeigten ILP-Modellen wird daher das in Kapitel 2.2 vorgestellte Queuing-Modell von TSN nicht berücksichtigt. Dieses schreibt vor, dass an den Ausgangsports von TSN-Bridges maximal acht FIFO-Queues zur Verfügung stehen. Aufgrund dieser hardwareseitigen Einschränkung können Frames innerhalb einer Bridge nur begrenzt in der Sendereihenfolge angepasst werden (d.h. abweichende Empfangs- und Sendereihenfolge von Frames). Lösungen, die von den bisher präsentierten ILP-Modellen erzeugt wurden, können somit in einigen Fällen nicht zur Konfiguration eines TSN-Netzwerkes eingesetzt werden, da mehr als acht FIFO-Queues notwendig wären, um die geplante Sendereihenfolge zu realisieren. In diesem Kapitel wird daher eine Erweiterung der ILP-Modelle um TSN-konforme Queuingbedingungen vorgeschlagen. Neben der Beschränkung auf acht FIFO-Queues pro Ausgangsport adressieren diese auch das ebenfalls TSN-spezifische Problem der *Frameisolation*. Die sogenannte Frameisolation stellt sicher, dass eine TSN-Bridge beim Fehlen eines im Schedule eingeplanten Frames (z.B. aufgrund eines Geräteausfalls) einen darauffolgenden Frame nicht zu früh sendet.

Die notwendigen Bedingungen wurden erstmals in [27] und [34] als SMT bzw. ILP formuliert und finden später auch in [14] und [42] Anwendung. Im Vergleich zu diesen Arbeiten wird hier nicht nur eine umfassendere Herleitung der Bedingungen gegeben, sondern es werden auch bisher nicht betrachtete Optimierungsmöglichkeiten aufgezeigt. Dies umfasst die Übertragung des Prinzips der Reduktion von Ressourcenbedingungen (Modelloption *rcr*) auf die Frameisolation, die Wiederverwendung der Variablen α_{klmxy} aus den Ressourcenbedingungen zur drastischen Reduktion benötigter Variablen, sowie den Vergleich zweier Modellvarianten.

7.2.1 Problemstellung: FIFO-Konformität und Frameisolation

An dieser Stelle soll zunächst konkretisiert werden, welche Anforderungen bei der Weiterleitung von Streams über denselben Ausgangsport einer TSN-Bridge zu berücksichtigen sind. Die Einschränkungen beim Scheduling aufgrund der Nutzung von maximal acht FIFO-Queues an einem Ausgangsport lassen sich folgendermaßen zusammenfassen:

- Sofern zwei Streams derselben Queue zugewiesen sind, muss die Sendereihenfolge der Frames der Empfangsreihenfolge entsprechen.
- Bei Zuweisung zweier Streams zu unterschiedlichen Queues ist eine Umsortierung zwischen Empfang und Senden zulässig.
- Für zwei Streams müssen die genannten Einschränkungen für alle zyklischen Wiederholungen ihrer Frames innerhalb der gemeinsamen Hyperperiode eingehalten werden.
- Zur Umsortierung von Frames stehen maximal acht Queues zur Verfügung, wobei nach den Empfehlungen von [61, § I] nicht mehr als fünf davon für zeitkritischen Verkehr genutzt werden sollten.

Ein weiteres TSN-spezifisches Problem liegt in der Isolation von aufeinanderfolgendes Frames bei Nutzung der gleichen Queue. Dieses Problem soll anhand eines einfachen Beispiels erläutert werden und zugleich die notwendigen Bedingungen zur Frameisolation hergeleitet werden.

7.2.1.1 Herleitung der Bedingungen zur Frameisolation

Das Problem der Isolation von aufeinanderfolgenden Frames ergibt sich in TSN vor allem daraus, dass für zeitgesteuerte Übertragungen nicht gezielt ein im Sendezeitplan vorgesehener Frame aus einem Speicher entnommen und gesendet wird, sondern ein Gate-Open-Event eine Queue zur Übertragung freigibt und einfach der an der Spitze der Queue stehende Frame ohne weitere Identifikation (z.B. durch Headerfelder) gesendet wird. Es ist also Aufgabe des Schedulers, Sendezeitpläne so zu erstellen, dass stets der richtige Frame im richtigen Moment an der Spitze der Queue steht. Dabei muss berücksichtigt werden, dass Anwendungen ggf. Frames mit abweichender Größe (im Vergleich zur ursprünglichen Streamspezifikation) senden. Während zu große Frames als Fehlverhalten der Anwendung gewertet werden können und beispielsweise durch Mechanismen wie PSFP verworfen werden können, um keine Störungen für anderen Verkehr zu verursachen, sollte das Netzwerk unerwartet kleine Frames korrekt übertragen und fehlende Frames ohne Folgefehler tolerieren können. Im Folgenden werden daher Zusatzbedingungen für Schedules beschrieben, um genau diese Robustheit zu gewährleisten.

Als Beispiel dient hier die Übertragung der Frames zweier Streams k und l über einen gemeinsam genutzten Link $(\mathrm{n0}, \mathrm{n3})$, wobei beide Streams der Queue $\mathrm{q0}$ zugewiesen sind. Zur Veranschaulichung zeigt Abbildung 7.7a die Queuenutzung, Gate-Events und die vom Scheduler geplanten Sendezeitschlitze auf dem gemeinsam genutzten Link und den direkten Vorgängerlinks der beiden Streams. Dabei wird zur Vereinfachung von Store-and-Forward-Bridges ohne Verarbeitungsverzögerung und von Links ohne Übertragungsverzögerung ausgegangen, sodass die Frames exakt dann in der Queue $\mathrm{q0}$ von Link $(\mathrm{n0}, \mathrm{n3})$ eingereiht werden, wenn die Übertragung auf dem jeweiligen Vorgängerlink endet. Ab dem Moment, in dem ein Frame in der Queue eingereiht ist, ist dieser *sendebereit* und kann am nächsten Gate-Open-Event übertragen werden. Wie in der Abbildung zu sehen, endet an diesem Gate-Open-Event außerdem die Queuenutzung durch den Frame, da dieser zwecks Übertragung aus der Queue entnommen wird. Das zugehörige Gate-Close-Event darf laut TSN-Standard [61, §8.6.8.4, §Q.2] dagegen erst nach dem Übertragungsende folgen. Solange die zugehörigen Anwendungen der Streams k und l ihre Frames in der spezifizierten Größe senden, kann der Sendezeitplan nach Abbildung 7.7a von TSN-Bridges eingehalten werden. So wird der Frame von k zwar in $\mathrm{q0}$ eingereiht, bevor die Übertragung von l abgeschlossen und das Gate wieder geschlossen ist, allerdings wird dieser Frame dennoch korrekterweise nicht im Zeitschlitz des Frames von l übertragen. Stattdessen erkennt eine TSN-konform arbeitende Bridge, dass der Frame von k nicht vor dem nächsten Gate-Close-Event vollständig übertragen werden kann und wartet daher mit der Übertragung bis zum nächsten, tatsächlich für k vorgesehenen Zeitschlitz.

Der erste zu betrachtende Problemfall ist die Übertragung kleinerer Frames als geplant. Liegt dieser Fall vor, so werden die jeweiligen Frames früher als erwartet in der Queue $\mathrm{q0}$ von Link $(\mathrm{n0}, \mathrm{n3})$ eingereiht. Dies ist in Abbildung 7.7b gezeigt, wobei hier für beide Streams davon ausgegangen wird, dass Frames in Minimalgröße $(64\,\mathrm{B})$ übertragen werden. Das Queuing in $\mathrm{q0}$ von $(\mathrm{n0}, \mathrm{n3})$ erfolgt demnach bereits kurz nach Beginn des jeweiligen Zeitschlitzes auf dem Vorgängerlink, da nur $64\,\mathrm{B}$ übertragen werden müssen. Daraus resultiert, dass sich die Queuenutzung von $\mathrm{q0}$ durch die Streams k und l auf $(\mathrm{n0}, \mathrm{n3})$ überlappt. Obwohl eine gleichzeitige Queuenutzung technisch möglich ist (die

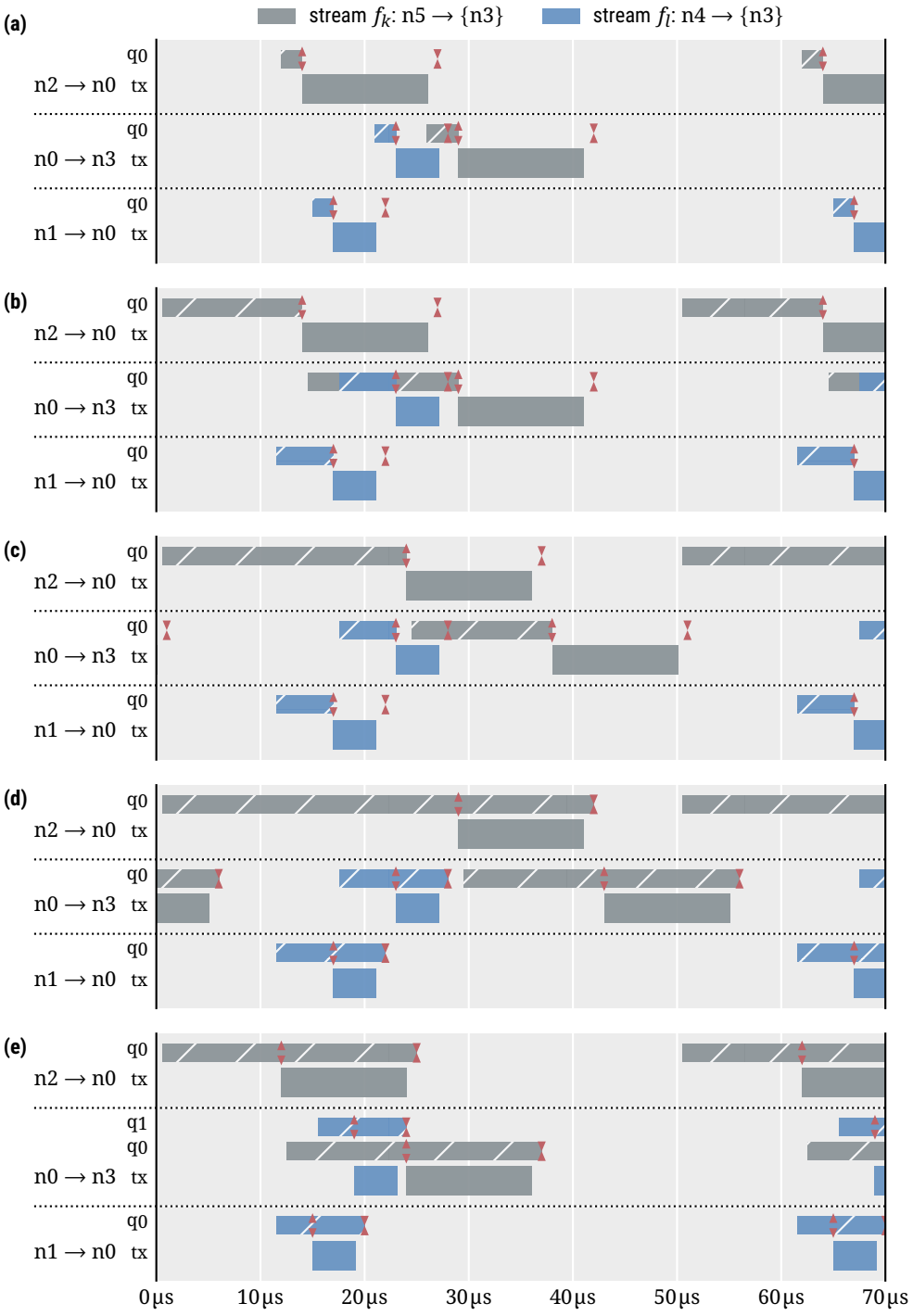


Abbildung 7.7: Herleitung der Bedingungen zur Frameisolation anhand von zwei Streams k und l mit Sendeintervall 50 µs. (a) Queuenutzung bei exakter Einhaltung der geplanten Framegrößen. (b) Framereordering bei unerwartet kleinen Frames. (c) Vermeidung von Reordering durch Einbeziehung kleinerer Frames bei der Planung. (d) Vermeidung von verfrühtem Senden im Falle zu kleiner Frames. (e) Frameisolation durch Nutzung verschiedener Queues.

Frames werden hintereinander in der Queue gespeichert), kann der geplante Schedule in diesem Fall nicht mehr eingehalten werden, da die Framereihenfolge nicht korrekt ist. Da der Frame von k zuerst eingereiht wurde, befindet sich dieser an der Spitze der Queue und würde am nächsten Gate-Open-Event zuerst übertragen werden, obwohl der Zeitschlitz für l vorgesehen war (Abbildung 7.7b zeigt nur die ursprünglich geplante Übertragung). Um diesen Fehlerfall zu vermeiden, muss ein Scheduler bei der Planung die korrekte Queuing-Reihenfolge auch für kleinere Framegrößen sicherstellen, als ursprünglich spezifiziert wurden. Wie in Abbildung 7.7c gezeigt, kann dies im diskutierten Beispiel durch ein späteres Senden von k auf dem Vorgängerlink erreicht werden.

Ein weiterer Fehlerfall kann allerdings auch bei einer Gerätekonfiguration nach Abbildung 7.7c auftreten: Ist hier sowohl der Frame von l als auch der von k ausreichend klein, kann die Übertragung von k noch im Zeitschlitz von l abgeschlossen werden. Dieses verfrühte Senden des Frames von k kann wiederum zu Fehlern auf Nachfolgelinks (z.B. veränderte Queuing-Reihenfolge) führen. Um diesen Fehler zu verhindern, darf der Frame von k erst nach dem Gate-Close-Event des Zeitschlitzes von l in der gleichen Queue eingereiht werden. Dieser korrigierte Fall ist in Abbildung 7.7d gezeigt. Zur Veranschaulichung der notwendigen Bedingungen wurde dabei außerdem die Queuenutzung der Frames bis zum zugehörigen Gate-Close-Event fortgeführt. Auch wenn die Frames aus technischer Sicht bereits am Gate-Open-Event aus der Queue entfernt werden, symbolisiert diese verlängerte Queuenutzung, dass bis zum Gate-Close-Event kein Frame eines anderen Streams die Queue nutzen darf, da sonst die Gefahr einer verfrühten Übertragung besteht.

Um ein vollständig deterministisches Netzwerkverhalten zu erzielen, muss ein Scheduler den Beginn der Queuenutzung durch einen Frame also auf Basis von Minimalframes ermitteln und das Ende der Queuenutzung entsprechend des Gate-Close-Events des geplanten Zeitschlitzes setzen, und außerdem überlappungsfreie Queuenutzungen für verschiedene Streams sicherstellen (entspricht Abb. 7.7d). Es sei angemerkt, dass theoretisch eine geringfügige Überlappung (max. $63\,\mathrm{B}$) dieser weit gefassten Queuenutzungen zulässig wäre: Da in einem solchen Fall auch ein Minimalframe nicht mehr vor dem Gate-Close-Event übertragen werden könnte, bestünde bei einer *standardkonformen* Bridge keine Gefahr einer verfrühten Übertragung des kurz vor dem Gate-Close-Event eingetroffenen Frames. In der Praxis konnte jedoch im Rahmen des Projekts *nicht-standardkonformes* Verhalten beobachtet werden, bei dem Bridges auch dann noch Frames übertragen haben, wenn dies nicht vor dem nächsten Gate-Close-Event abgeschlossen werden konnte. Aus diesem Grund wird im Folgenden davon ausgegangen, dass die Queuenutzung eines Folgeframes immer erst nach dem Gate-Close-Event eines vorangegangenen Frames beginnen darf. Diese Modellierung ist gleichermaßen für standardkonforme und nicht-standardkonforme Bridges nutzbar. Die erläuterten Bedingungen machen allerdings dicht aufeinanderfolgende Übertragungen zweier Frames bei Nutzung der gleichen Queue unmöglich (siehe Abb. 7.7d). Um diese zu ermöglichen muss ein Scheduler Zuweisungen zu unterschiedlichen Queues vornehmen. Ein entsprechender Schedule ist in Abbildung 7.7e gezeigt.

Insgesamt stellt die Frameisolation in TSN also sicher, dass Schedules auch bei fehlenden und zu kleinen Frames eingehalten werden können. Dabei kann die Isolation entweder zeitlich (überlappungsfreie Nutzung der gleichen Queue) oder räumlich (Nutzung unterschiedlicher Queues erfolgen). Des Weiteren sei angemerkt, dass die beschriebenen Bedingungen zur Frameisolation implizit auch FIFO-Konformität sicherstellen.

7.2.2 Modellerweiterung bezüglich FIFO-Queuing und Frameisolation

Die zuvor am Beispiel beschriebenen Bedingungen sollen als Teil des ILP-Modells formalisiert werden. Hierfür werden zunächst Hilfsausdrücke definiert, die die anschließende Beschreibung von Randbedingungen zur Frameisolation erleichtern.

7.2.2.1 Hilfsausdrücke zur Bildung der Randbedingungen

Wie im einführenden Beispiel ersichtlich, wird bei der Formulierung von Bedingungen zur (zeitlichen) Frameisolation der Zeitpunkt benötigt, ab dem ein Frame nach einer Übertragung auf Link m frühestens in eine Queue auf dem Nachfolgelink eingereiht wird. Hierfür muss davon ausgegangen werden, dass ein Minimalframe anstelle des ursprünglich spezifizierten Frames gesendet wird. Somit ergibt sich die folgende kleinstmögliche Verzögerung d_m^{min} eines Frames beim Durchlaufen von Link $m := (i, j)$ und Bridge $j := m_1$:

$$d_m^{min} := \frac{\min(v_j.hb, 64\text{B} + pre + sfd)}{e_m.ls} + e_m.prp + v_j.prc$$

Beginnt der Sendezeitschlitz eines Streams k auf Link m zum Zeitpunkt t^s, so wird ein Frame von k auf dem Nachfolgelink von m demnach frühestens zum Zeitpunkt $t^s + d_m^{min}$ in eine Queue eingereiht. Entsprechend der vorigen Erläuterungen ist dieser Zeitpunkt zur Bestimmung der Queuenutzung im Falle unerwartet kleiner Frames notwendig, und ist im Gegensatz zur regulären Bridgeverzögerung d_{km}^{fwd} nicht streamspezifisch, da die angesetzte Framegröße stets ein Minimalframe ist.

Entsprechend der Ressourcenbedingung (Vermeidung von Zeitschlitzkonflikten) muss die Frameisolation außerdem paarweise für alle Streamkombinationen (k, l), für jeden von (k, l) potentiell gemeinsam genutzten Link m und für alle zyklischen Wiederholungen (x, y) der zugehörigen Frames bis zur gemeinsamen Hyperperiode von k und l formuliert werden. Zur Formulierung der Ressourcenbedingungen wurden daher ursprünglich die Mengen $RC^{k..l}$ und $RC^{l..k}$ aufgestellt und in der Modelloption *rcr* optimiert, welche alle relevanten Kombinationen als Tupel (k, l, m, x, y) enthalten (vgl. Kapitel 6.4.2.4 und 6.4.4.3). Die nach *rcr* optimierten Mengen sind jedoch bei der Bildung

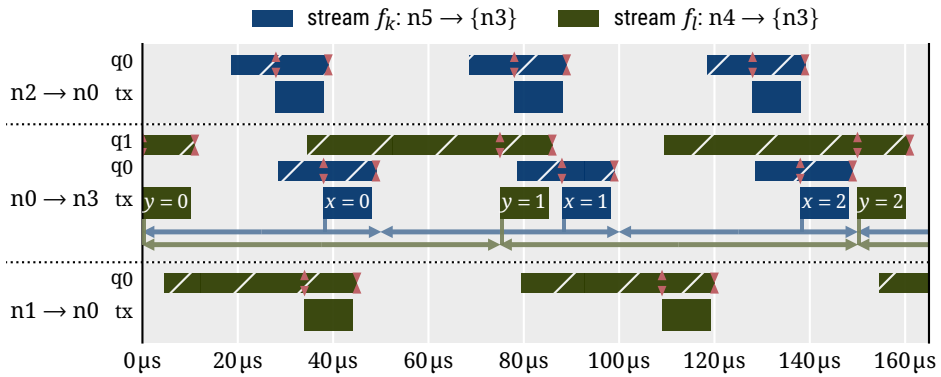


Abbildung 7.8: Zeitschlitzwiederholungen zweier Streams mit $f_k.ct := 50\,\mu s$ und $f_l.ct := 75\,\mu s$. Pfeile zeigen den laut ILP-Variablen möglichen Wertebereich des jeweiligen Zeitschlitzanfangs.

der Bedingungen zur Frameisolation nicht mehr ausreichend, da nicht alle relevanten Zeitschlitzwiederholungen enthalten sind. Dies ist bei der Betrachtung der Zeitschlitzwiederholungen (x, y) der Streams k und l in Abbildung 7.8 ersichtlich: Für das Zeitschlitzpaar $(0, 1)$ ist keine Überlappung der Sendezeiten (tx) auf Link $(\mathrm{n0}, \mathrm{n3})$ möglich, da die 0-te Wiederholung von k spätestens zum Zeitpunkt $49\,\mu\mathrm{s}$ gesendet werden kann und mit einer Länge von $10\,\mu\mathrm{s}$ maximal bis $59\,\mu\mathrm{s}$ reicht, während die 1-te Wiederholung von l frühestens ab $75\,\mu\mathrm{s}$ gesendet werden kann. In $RC^{k..l}$ nach *rcr* ist diese (x, y)-Kombination daher nicht enthalten und es existiert somit keine entsprechende Konfliktprüfung im ILP-Modell. Die Abbildung zeigt jedoch, dass hier dennoch Bedingungen zur Frameisolation notwendig sind, da die Queuenutzung dieser beiden Zeitschlitzwiederholungen trotzdem überlappen kann. Daher werden hier äquivalent zu $RC^{k..l}$ und $RC^{l..k}$ die Mengen $FI^{k..l}$ und $FI^{l..k}$ eingeführt, welche die größeren Wertebereiche der Queuenutzung (im Vergleich zum Wertebereich der Zeitschlitze) berücksichtigen und somit alle relevanten Kombinationen von Links, Streams und Zeitschlitzwiederholungen enthalten, für die Bedingungen zur Frameisolation erforderlich sind.

Die grundlegende Idee bei der Formulierung dieser Mengen ist, dass ein Zeitschlitzpaar immer dann Bedingungen zur Frameisolation benötigt, wenn es aufeinanderfolgend (ohne weitere Zeitschlitzwiederholungen dazwischen) geplant werden könnte, so wie im Beispiel für die Wiederholungen $(x, y) = (0, 1)$ gegeben. Für die Wiederholungen $(1, 2)$ ist dagegen keine Bedingung zur Frameisolation notwendig, da die Bedingung bereits für $(2, 2)$ gefordert werden muss, und die Wiederholungen $(1, 2)$ nie direkt aufeinander folgen können, da der 2-te Zeitschlitz von k aufgrund der Wertebereiche nicht hinter dem 2-ten Zeitschlitz von l liegen kann. Wie für *rcr* wird hier zunächst ein boolescher Ausdruck definiert, der *wahr* wird, wenn für ein Zeitschlitzpaar *keine* Bedingung zur Frameisolation notwendig ist:

$$\begin{aligned} fi^{ex}_{klmxy} \;:=\; & \left(\min(x \cdot f_k.ct/hp_{kl}, y \cdot f_l.ct/hp_{kl}) > 0\right) \vee \left((x+2) \cdot f_k.ct - 1 \leq y \cdot f_l.ct + sl'_{lm}\right) \\ & \vee \left((y+2) \cdot f_l.ct - 1 \leq x \cdot f_k.ct + sl'_{km}\right) \end{aligned}$$

Die Mengen relevanter Zeitschlitzpaare lassen sich dann folgendermaßen definieren:

$$FI^{k..l} \;:=\; \left\{(k, l, m, x, y) \in RC \mid \neg fi^{ex}_{klmxy},\, \mathfrak{a}_{klmxy} \neq 0\right\}$$

$$FI^{l..k} \;:=\; \left\{(k, l, m, x, y) \in RC \mid \neg fi^{ex}_{klmxy},\, \mathfrak{a}_{klmxy} \neq 1\right\}$$

Dabei entspricht RC weiterhin der Definition aus Kapitel 6.4.2.4 und umfasst alle potentiellen Zeitschlitzkonflikte, während $\mathfrak{a}_{klmxy}$ entsprechend der Modelloption *rcr* (vgl. Kapitel 6.4.4.3) entweder eine ILP-Entscheidungsvariable ist, oder feste Werte annimmt, sofern die Zeitschlitzreihenfolge zum Zeitpunkt der Modellbildung aufgrund der Wertebereiche (vgl. Abb. 7.8) bereits bekannt ist.

7.2.2.2 Modelloptionen zur Frameisolation (*bq*/*iq*)

Die hier formulierten Modellerweiterungen beschreiben die zuvor eingeführte Frameisolation (d.h. zeitliche oder räumliche Isolation), die Fehlerfälle durch fehlende oder zu kleine Frames vermeidet und zugleich für FIFO-Konformität sorgt. Hierbei wird angenommen, dass die Queuezuweisung eines Streams für alle zyklischen Wiederholungen des zugehörigen Frames konstant bleibt, obwohl die

zeitgesteuerten PSFP-Gates (vgl. Kapitel 2.2.3.5) auch unterschiedliche Queuezuweisungen im Laufe einer Hyperperiode erlauben würden. Die Einschränkung wird hier zur Vereinfachung der Modellierung vorgenommen, kann jedoch in der Praxis ebenfalls von Vorteil sein, um die Länge der Stream-GCLs gering zu halten. Zwecks Performancevergleich wurden zwei Modellvarianten entworfen.

Modelloption *bq* Zur Formulierung von Bedingungen zur Frameisolation muss die Zuweisung von Streams zu den Queues eines Netzwerklinks als Entscheidungsvariable abgebildet werden. Bei der Modelloption *bq* (von *binary queue modeling*) werden dem ILP-Modell hierfür die Binärvariablen $\mathfrak{qb}_{kmn}$ hinzugefügt. Dabei bedeutet $\mathfrak{qb}_{kmn} = 1$, dass der Stream k auf Link m der n-ten Queue zugewiesen ist. Durch (7.10) wird dann ausgedrückt, dass auf den laut Pfadvariablen genutzten Links eine Queuezuweisung erfolgen muss, wobei einem Stream in diesem Fall genau eine Queue zugewiesen wird. Die Anzahl der zur Verfügung stehenden Queues beim Senden über Link m ist hierbei vorab durch q_m^{no} gegeben.

$$\forall k \in F,\ \forall m \in E_k \qquad \mathfrak{p}_{km} = \sum_{n \in \mathbb{N}^0 \mid n < q_m^{no}} \mathfrak{qb}_{kmn} \tag{7.10}$$

Weitere Bedingungen müssen sicherstellen, dass die Nutzung der gleichen Queue durch zwei Streams stets überlappungsfrei bleibt. Diese Bedingungen zur zeitlichen Frameisolation, (7.11) und (7.12), werden für alle relevanten Kombinationen von (k, l, m, x, y) benötigt, welche zuvor in $FI^{k..l}$ und $FI^{l..k}$ erfasst wurden. Des Weiteren müssen bei der zeitlichen Frameisolation die Ankunftszeiten der weitergeleiteten Frames betrachtet werden, welche im ILP-Modell nur auf Basis der Sendezeitpunkte auf den jeweiligen Vorgängerlinks bestimmt werden können. Da zum Zeitpunkt der Modellbildung noch unbekannt ist, über welchen Eingangslink die jeweiligen Frames empfangen werden, muss die Bedingung zur zeitlichen Frameisolation für jeden Eingangslink der aktuell betrachteten Bridge m_0 formuliert werden (d.h. $\forall \overleftarrow{m} \in E^{\leftarrow}_{lm_0}$ bzw. $\forall \overleftarrow{m} \in E^{\leftarrow}_{km_0}$). Außerdem müssen die entsprechenden Bedingungen separat für jede Queue eines Links m aufgestellt werden (d.h. $\forall n \in \mathbb{N}^0 \mid n < q_m^{no}$). Für alle beschriebenen Fälle erfolgt die Modellierung der zeitlichen Frameisolation in (7.11) und (7.12) äquivalent zur Ressourcenbedingung als Entweder-oder-Bedingung, um die beiden möglichen Sendereihenfolgen der Frames von k und l abzubilden.

$$\begin{aligned} &\forall (k, l, m, x, y) \in FI^{k..l},\\ &\forall \overleftarrow{m} \in E^{\leftarrow}_{lm_0},\ \forall n \in \mathbb{N}^0 \mid n < q_m^{no} \end{aligned} \qquad \begin{aligned} &x \cdot f_k.ct + \boldsymbol{t}_{km}^{\boldsymbol{mod}} + \lceil sl_{km} \rceil\\ \leq\ &(y + \mathfrak{v}_{l\overleftarrow{m}} - \mathfrak{v}_{lm}) \cdot f_l.ct + \boldsymbol{t}_{l\overleftarrow{m}}^{\boldsymbol{mod}} + \lfloor d_{\overleftarrow{m}}^{min} \rfloor\\ &+ M_{7.11} \cdot (4 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{l\overleftarrow{m}} - \mathfrak{qb}_{kmn} - \mathfrak{qb}_{lmn}) \end{aligned} \tag{7.11}$$

$$\begin{aligned} &\forall (k, l, m, x, y) \in FI^{l..k},\\ &\forall \overleftarrow{m} \in E^{\leftarrow}_{km_0},\ \forall n \in \mathbb{N}^0 \mid n < q_m^{no} \end{aligned} \qquad \begin{aligned} &y \cdot f_l.ct + \boldsymbol{t}_{lm}^{\boldsymbol{mod}} + \lceil sl_{lm} \rceil\\ \leq\ &(x + \mathfrak{v}_{k\overleftarrow{m}} - \mathfrak{v}_{km}) \cdot f_k.ct + \boldsymbol{t}_{k\overleftarrow{m}}^{\boldsymbol{mod}} + \lfloor d_{\overleftarrow{m}}^{min} \rfloor\\ &+ M_{7.12} \cdot (3 + \mathfrak{a}_{klmxy} - \mathfrak{p}_{k\overleftarrow{m}} - \mathfrak{qb}_{kmn} - \mathfrak{qb}_{lmn}) \end{aligned} \tag{7.12}$$

Die Erläuterung der Gleichungen soll beispielhaft für den Fall erfolgen, dass auf m der x-te Frame von k vor dem y-ten Frame von l übertragen wird. Für diese Sendereihenfolge wird die notwendige Bedingung zur zeitlichen Frameisolation in (7.11) modelliert. Darüber hinaus sei zunächst angenommen, dass $M_{7.11}$ mit Null multipliziert wird und der entsprechende Term somit wegfällt. In diesem Fall beschreibt die linke Seite der Ungleichung (7.11) den Zeitpunkt des Zeitschlitzendes der x-ten Framewiederholung des Streams k. Das Aufrunden von sl_{km} ist hierbei erforderlich, da das zum Zeitschlitzende gehörende Gate-Close-Event laut Annahmen über die Bridgefähigkeiten auf einen ganzzahligen Zeitpunkt fallen muss. Rechtsseitig verbleibt nach dem Wegfall des M-Terms der Zeitpunkt, an dem die y-te Framewiederholung des Streams l für die Weiterleitung auf m *sendebereit* wird. Dieser Zeitpunkt wird dabei anhand des Sendezeitpunkts auf dem Vorgängerlink $\overleftarrow{m}$ und der minimalen Verzögerung des Links und der durchlaufenen Bridge gebildet. Hierbei ist zu berücksichtigen, dass der zur y-ten Framewiederholung auf m gehörige Frame auf dem

Eingangslinks $m^{\leftarrow}$ nicht ebenfalls die y-te Wiederholung sein muss (siehe Abb. 7.8, Zeitschlitze von l auf $(\mathrm{n0}, \mathrm{n3})$ und Vorgängerlink $(\mathrm{n1}, \mathrm{n0})$), da das Pfadscheduling eine Zeitplanung über die Grenzen der Zykluszeit hinweg erlaubt. Die relevante Startzeit auf dem Eingangslink $m^{\leftarrow}$ ergibt sich daher aus $(y + \mathfrak{o}_{lm^{\leftarrow}} - \mathfrak{o}_{lm}) \cdot f_l.ct + \boldsymbol{t}^{\boldsymbol{mod}}_{lm^{\leftarrow}}$.

Abschließend soll der bisher als Null angenommene M-Term erläutert werden. Wie üblich dient dieser dazu, die Gleichung durch Addition einer großen Konstante trivial zu erfüllen, wenn diese nicht aktiv sein soll. Dies ist hier genau dann notwendig, wenn der betrachtete Eingangslink $m^{\leftarrow}$ von l gar nicht genutzt wird ($\mathfrak{p}_{lm^{\leftarrow}} = 0$) oder einer der Streams auf m nicht der gerade betrachteten Queue n zugewiesen ist ($\mathfrak{qb}_{kmn} = 0$ oder $\mathfrak{qb}_{lmn} = 0$). Letzteres bildet die räumliche Frameisolation ab: Die Streams sind unterschiedlichen Queues zugewiesen und zeitliche Frameisolation ist daher nicht notwendig. Durch $\mathfrak{a}_{klmxy}$ wird darüber hinaus die Zeitschlitzreihenfolge entschieden und dementsprechend entweder (7.11) oder (7.12) aktiviert. Wichtig ist hierbei, dass es sich bei $\mathfrak{a}_{klmxy}$ um die gleiche Variable handelt, die schon in den Ressourcenbedingungen eingesetzt wird: So aktiviert $\mathfrak{a}_{klmxy} = 1$ sowohl (6.7) als auch (7.11) für die x-te Zeitschlitzwiederholung des Streams k und die y-te Zeitschlitzwiederholung des Streams l auf dem Link m. Gleichung (6.7) sorgt dann dafür, dass der referenzierte Zeitschlitz von l überlappungsfrei nach dem von k geplant wird, während (7.11) für genau diese Zeitschlitze sicherstellt, dass die Ankunftszeit der zugehörigen Frames von l die Bedingung zur zeitlichen Frameisolation einhält (sofern diese nicht wie beschrieben aufgrund anderer Variablen des Aktivierungsterms inaktiv ist).

Modelloption *iq* Alternativ zur Modellierung der Queuezuweisung eines Streams k auf Link m durch mehrere binäre Entscheidungsvariablen kann stattdessen eine einzige ganzzahlige Variable eingesetzt werden. Bei der Modelloption *iq* (von *integer queue modeling*) wird dem ILP-Modell dementsprechend eine Ganzzahlvariable $0 \leq \mathfrak{qi}_{km} \leq q^{no}_m - 1$ hinzugefügt, die die zugewiesene Queue von k auf m explizit als Wert widerspiegelt. Ob für zwei Streams (k, l) auf Link m Bedingungen zur zeitlichen Frameisolation aktiv sein sollen, wird außerdem durch eine zusätzliche Binärvariable $\mathfrak{fi}_{klm}$ abgebildet. Nimmt diese den Wert Eins an, so werden (in Abhängigkeit weiterer Variablen) die in (7.13) und (7.14) gezeigten Bedingungen zur zeitlichen Frameisolation aktiviert.

$$\begin{aligned} \forall (k,l,m,x,y) \in FI^{k..l},\ \forall m^{\leftarrow} \in E^{\leftarrow}_{lm_0}: \quad & x \cdot f_k.ct + \boldsymbol{t}^{\boldsymbol{mod}}_{km} + \lceil sl_{km} \rceil \\ & \leq (y + \mathfrak{o}_{lm^{\leftarrow}} - \mathfrak{o}_{lm}) \cdot f_l.ct + \boldsymbol{t}^{\boldsymbol{mod}}_{lm^{\leftarrow}} + \lfloor d^{min}_{m^{\leftarrow}} \rfloor \\ & + M_{7.13} \cdot (5 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{lm^{\leftarrow}} - \mathfrak{p}_{km} - \mathfrak{p}_{lm} - \mathfrak{fi}_{klm}) \end{aligned} \tag{7.13}$$

$$\begin{aligned} \forall (k,l,m,x,y) \in FI^{l..k},\ \forall m^{\leftarrow} \in E^{\leftarrow}_{km_0}: \quad & y \cdot f_l.ct + \boldsymbol{t}^{\boldsymbol{mod}}_{lm} + \lceil sl_{lm} \rceil \\ & \leq (x + \mathfrak{o}_{km^{\leftarrow}} - \mathfrak{o}_{km}) \cdot f_k.ct + \boldsymbol{t}^{\boldsymbol{mod}}_{km^{\leftarrow}} + \lfloor d^{min}_{m^{\leftarrow}} \rfloor \\ & + M_{7.14} \cdot (4 + \mathfrak{a}_{klmxy} - \mathfrak{p}_{km^{\leftarrow}} - \mathfrak{p}_{km} - \mathfrak{p}_{lm} - \mathfrak{fi}_{klm}) \end{aligned} \tag{7.14}$$

Diese unterscheiden sich gegenüber Modelloption *bq* nur durch den Aktivierungsterm, sodass sie hier nicht erneut erläutert werden sollen. Neben $\mathfrak{fi}_{klm}$ hängt die Aktivierung der Bedingungen vom aktuellen Routing (gegeben in den Pfadvariablen) und der durch $\mathfrak{a}_{klmxy}$ ausgedrückten Zeitschlitzreihenfolge ab. Für $\mathfrak{fi}_{klm} = 0$ sind die gezeigten Bedingungen zur zeitlichen Frameisolation inaktiv und k und l müssen in diesem Fall auf Link m unterschiedlichen Queues zugewiesen werden (räumliche Frameisolation). Dieser Zusammenhang zwischen $\mathfrak{fi}_{klm}$ und den Queuezuweisungen in $\mathfrak{qi}_{km}/\mathfrak{qi}_{lm}$ wird durch die Bedingungen (7.15) und (7.16) ausgedrückt.

$$\forall (k,l) \in F \times F \mid k < l,\ \forall m \in E^{\cap}_{kl}: \quad \mathfrak{qi}_{km} - \mathfrak{qi}_{lm} \geq 1 - q^{no}_m \cdot (3 - \mathfrak{c}_{klm} - \mathfrak{p}_{km} - \mathfrak{p}_{lm} + \mathfrak{fi}_{klm}) \tag{7.15}$$

$$\forall (k,l) \in F \times F \mid k < l,\ \forall m \in E^{\cap}_{kl}: \quad \mathfrak{qi}_{lm} - \mathfrak{qi}_{km} \geq 1 - q^{no}_m \cdot (2 + \mathfrak{c}_{klm} - \mathfrak{p}_{km} - \mathfrak{p}_{lm} + \mathfrak{fi}_{klm}) \tag{7.16}$$

Die Gleichungen modellieren, dass entweder nach (7.15) Stream k einer höheren Queue zugewiesen sein muss als l oder umgekehrt (entsprechend (7.16)). Welcher der beiden Fälle zutrifft, wird durch $\mathfrak{c}_{klm}$ entschieden ($\mathfrak{c}_{klm} = 1$: (7.15) ist aktiv), während die jeweils andere Bedingung durch Subtraktion von q_m^{no} trivial erfüllt wird. Außerdem werden beide Gleichungen deaktiviert, wenn einer der Streams den Link m laut Pfadvariablen gar nicht nutzt oder wenn laut $\mathfrak{fi}_{klm}$ Bedingungen zur zeitlichen Frameisolation aktiv sind, da in diesen Fällen beliebige Queuezuweisungen (einschließlich gleicher Queue für k und l) zulässig sind. Die Funktionsweise der Variable $\mathfrak{fi}_{klm}$ kann auch folgendermaßen beschrieben werden: Entweder muss zeitliche Frameisolation gelten und es sind daher beliebige Queuezuweisungen erlaubt ($\mathfrak{fi}_{klm} = 1$, (7.13)/(7.14) aktiv, (7.15)/(7.16) inaktiv), oder es wird räumliche Frameisolation gefordert und die Zuweisung zur selben Queue ist daher nicht zulässig ($\mathfrak{fi}_{klm} = 0$, (7.13)/(7.14) inaktiv, (7.15)/(7.16) aktiv).

Die wesentlichen Unterschiede der Modelloption *iq* gegenüber *bq* liegen darin, dass die ganzzahlige Modellierung der Queuezuweisung weniger Variablen benötigt und die Bedingungen zur Frameisolation ebenfalls nicht mehr separat für alle Queues wiederholt werden müssen. Damit ist die Anzahl der benötigten Variablen und Randbedingungen unabhängig von der Anzahl zur Verfügung stehender Queues. Im Gegenzug wird jedoch mit $\mathfrak{c}_{klm}$ und $\mathfrak{fi}_{klm}$ eine größere Anzahl an Hilfsvariablen benötigt. Eine klare Erwartung bezüglich der Ergebnisse für die beiden Modellvarianten kann daher nicht formuliert werden. Bei hoher Queueanzahl (q_m^{no}) wären Nachteile für *bq* zu erwarten, da die Modellgröße hier von Queueanzahl abhängt. Dieser Fall liegt jedoch aufgrund der sehr begrenzten Queueanzahl von TSN in der Regel nicht vor.

7.2.3 Evaluation von FIFO-Queuing und Frameisolation

Die wesentlichen Fragen, die sich bezüglich der Bedingungen für Queuezuweisungen und Frameisolation ergeben, sind (a) inwiefern gegebene Szenarien bei limitierter Queueanzahl noch lösbar sind, (b) wie stark die Performanceeinbußen durch die zusätzlichen Randbedingungen sind, und (c) welche der beiden Modelloptionen *bq*/*iq* performanter ist. Zur Beantwortung dieser Fragen wurden *bq*/*iq* für alle drei Basismodelle, mit und ohne die Modelloption *modv*, und für die Optimierungsziele *1SP*/*SPLT25* miteinander verglichen, wobei global $q_m^{no} := 4$ gesetzt wurde.

Die Ergebnisse für $MCS_{SPLT25}^{o1,stl}$ und $MCS_{SPLT25}^{o1,modv,stl}$ sind in Abbildung 7.9 dargestellt. Darin ist ersichtlich, dass die Schedulability durch die zusätzlichen Bedingungen nur geringfügig sinkt, wobei in keinem Fall Unlösbarkeit nachgewiesen werden konnte. Bei Betrachtung der einzelnen Testcases (siehe [E19]) bestätigt sich ebenfalls das Bild, dass die Schedulability-Nachteile im Wesentlichen durch die höheren Rechenzeiten und das Zeitlimit entstehen. Dies bedeutet, dass die konfigurierte Queueanzahl von 4 zumindest für die gegebenen Benchmarkingszenarien ausreichend ist und hier in keinem Fall nachgewiesen werden konnte, dass zusätzliche Queues zwecks Reordering oder Isolation notwendig wären. Demnach stellen die technischen Möglichkeiten von TSN-Bridges keine relevante Einschränkung bei der Lösbarkeit von Szenarien dar.

Der Rechenzeitnachteil liegt jedoch bereits im günstigsten Fall (Hinzuschalten der Option *iq* ohne *modv*) etwa bei einem Faktor von 2. Für die Modelloption *bq* fällt dieser in der Regel höher aus, wobei dieser Nachteil seitens *bq* bei niedriger Last stärker ausgeprägt ist. Auch die Ergebnisse mit Modelloption *1SP* fallen nochmals schlechter aus (siehe [E19]) und liegen in ungünstigen Fällen etwa bei der fünffachen c.s.i.-Rechenzeit. Insgesamt liegen die Rechenzeitnachteile durch die zusätzlichen

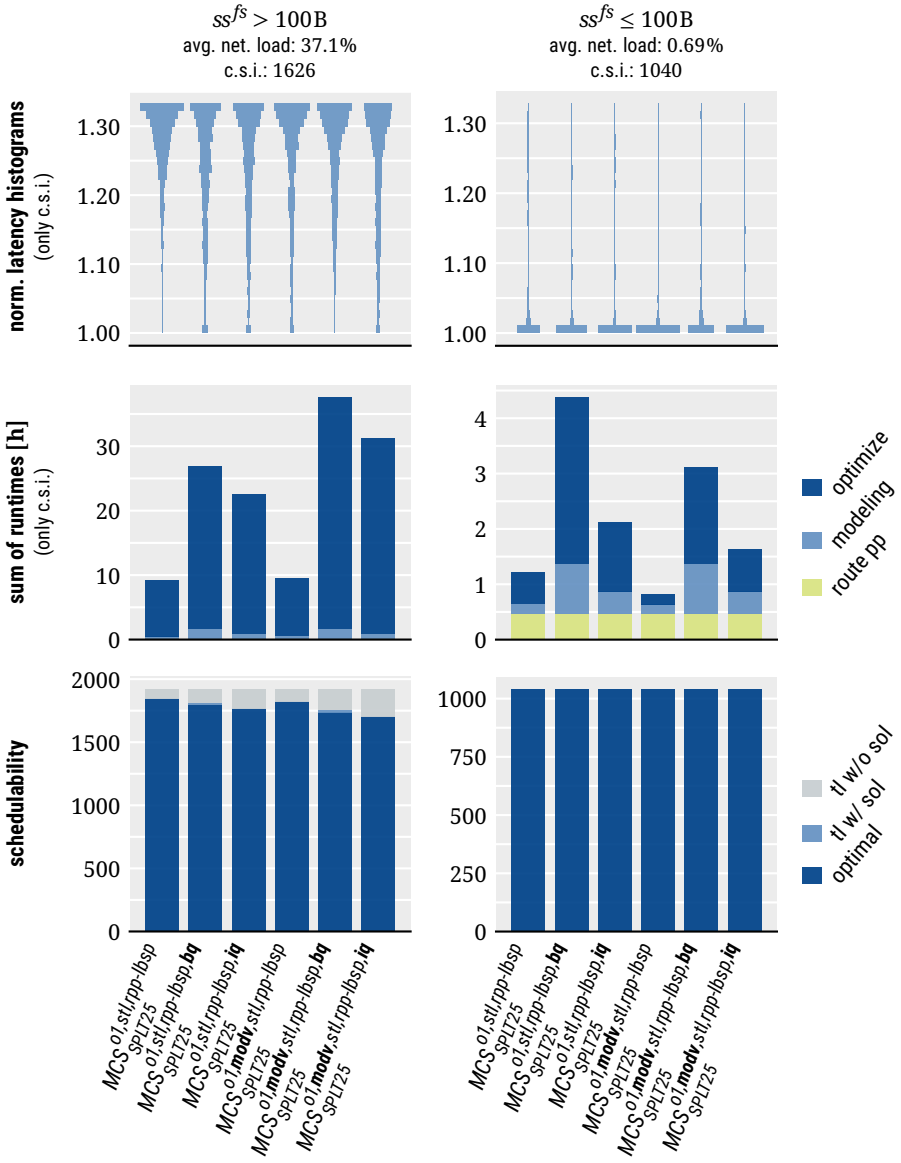


Abbildung 7.9: Schedulability, c.s.i.-Rechenzeiten und Latenzverteilung der Modelle zur Frameisolation für $MCS^{o1,stl}_{SPLT25}$ und $MCS^{o1,modv,stl}_{SPLT25}$.

Queuezuweisungen und die Frameisolation jedoch in einer Größenordnung, bei der die praktische Anwendbarkeit der Modelle weiterhin gegeben bleibt. Eine präzisere Einordnung lässt sich ohne weitere Vergleichsmodelle oder eine bestimmte Zielanwendung nicht vornehmen.

Beim Vergleich von *bq* und *iq* zeigt sich, dass *iq* aufgrund der Rechenzeiten insbesondere bei großen Szenarien mit niedriger Last zu bevorzugen ist, wohingegen der Nachteil von *bq* bei hoher Last massiv sinkt und *bq* außerdem eine geringfügig bessere Schedulability bietet. Ein dominierendes Modell lässt sich somit nicht identifizieren. Im weiteren Verlauf der Arbeit wird jedoch aus Rechenzeitgründen nur noch das Modell *iq* weiter betrachtet. Die hier gezeigten Ergebnisse treffen gleichermaßen auf die Basismodelle *UC* und *MCT* zu (siehe [E19]).

7.3 Zusammenfassung entworfener Optimierungen und Erweiterungen

In diesem Kapitel wurden zunächst die Schwächen der ILP-Modelle aus Kapitel 6 adressiert. Hierzu wurde ein Multicastmodell *MCS* entworfen, das die Performancelücke zwischen Unicast- und Multicastmodellen verringert. Die Performance und Skalierbarkeit bei niedriger Netzwerkauslastung konnte wiederum deutlich verbessert werden, indem die Repräsentation der Sendezeitpunkte im ILP angepasst wurde (Modelloptionen *mod*/*modv*). Neben diesen Umformulierungen des Modells, welche weiterhin denselben Lösungsraum beschreiben, wurden außerdem weitere Varianten der heuristischen Routenvorverarbeitung entworfen. Hierbei konnte für die sehr restriktive Modelloption *rpp-lbsp* nachgewiesen werden, dass diese den Rechenzeitnachteil von Ansätzen mit gemeinsamem Routing und Scheduling (JRaS) gegenüber zweischrittigen Lösungen (SRaS) fast vollständig eliminieren kann, ohne dabei die Vorteile von JRaS bezüglich Schedulability und Lösungsgüte zu verlieren. Dabei wird ausgenutzt, dass im JRaS-Ansatz tatsächlich nur wenige zusätzliche Routingoptionen modelliert werden müssen, um die Unzulänglichkeiten des festen Routings (SRaS) zu beheben.

Die erzielte Performanceverbesserung wurde wiederum genutzt, um komplexitätssteigernde TSN-konforme Queuezuweisungen in das ILP-Modell zu integrieren. Dabei wurden mögliche Abweichungen vom geplanten Schedule durch fehlende oder unerwartet kleine Frames berücksichtigt und zwecks Fehlervermeidung sowohl die zeitliche als auch die räumliche Frameisolation (d.h. Queuezuweisungen) modelliert. Während sich die Rechenzeit durch das komplexere Modell um einen Faktor von $2-3$ verschlechtert, fielen die Nachteile seitens der Schedulability nur gering aus. Konkret stellt insbesondere die im TSN-Standard verankerte Beschränkung auf 8 FIFO-Queues pro Ausgangsport einer Bridge in den getesteten Benchmarkingszenarien kein relevantes Problem dar. Eine umfassendere Einordnung der vielen entworfenen ILP-Modellvarianten erfolgt in Kapitel 8.

Kapitel 8

Multicast-Benchmarking und Zusammenfassung

Nachdem in den vorangegangenen Kapiteln eine Vielzahl verschiedener ILP-Modellvarianten zum Lösen von TT-RSP vorgestellt wurden, fokussiert sich dieses Kapitel auf eine abschließende Analyse und Bewertung der Verfahren. Dies umfasst einen Vergleich der Multicast-Modelle *MCS* und *MCT* mit Hilfe des Benchmarking-Datensatzes mit Multicastszenarien, welcher in den bisherigen Analysen zwecks Vergleich mit dem sehr performanten Unicast-Modell *UC* nicht zum Einsatz kam. Anschließend sollen die im Laufe der Entwicklung erzielten Fortschritte zusammengefasst werden und bewertet werden, inwiefern die ursprünglichen Ziele erreicht wurden, bevor ein Ausblick auf weitere Entwicklungsmöglichkeiten gegeben wird.

8.1 Benchmarking mit Multicast-Verkehrsmustern

Zur abschließenden Bewertung von *MCS* und *MCT* wurde der Multicast-Datensatz des Benchmarkings genutzt. Neben dem Einsatz von Multicaststreams unterscheidet sich dieser vom Unicast-Benchmarking auch durch den zusätzlichen Einsatz der $fattree$-Topologie sowie einen grundsätzlich höheren Schwierigkeitsgrad. In den Szenarien mit hoher Last liegt dieser vor allem an der steigenden Linkauslastung in Richtung der Empfänger, während die Szenarien mit niedriger Last außerdem auch eine höhere Streamanzahl nutzen als im Unicast-Fall. Die Ergebnisse werden in Abbildung 8.1 wieder beispielhaft für den optimierenden Fall dargestellt. Die wesentliche Erkenntnis liegt an dieser Stelle darin, dass *MCT* in der Regel gleichwertig zu *MCS* oder gar überlegen ist, wobei die Unterschiede zu *MCS* je nach Modelloption variieren. Damit unterscheiden sich die Ergebnisse vom Unicast-Benchmarking, in dem mit *MCS* deutliche Performanceverbesserungen erzielt werden konnten. Die Betrachtung einzelner Testcases (siehe Abb. 8.2) zeigt darüber hinaus, dass dieser Unterschied nicht etwa durch die Aufnahme einer neuen Topologie verursacht wurde, sondern dass *MCS* auch in zuvor schon getesteten Topologien schlechter abschneidet, in denen die Bewertung von *MCS* und *MCT* im Unicast-Fall noch umgekehrt ausfiel. Da diese Szenarien sich hier gegenüber dem Unicast-Fall nur durch die Empfängeranzahl der Streams und eine damit einhergehende Anpassung der Streamanzahl

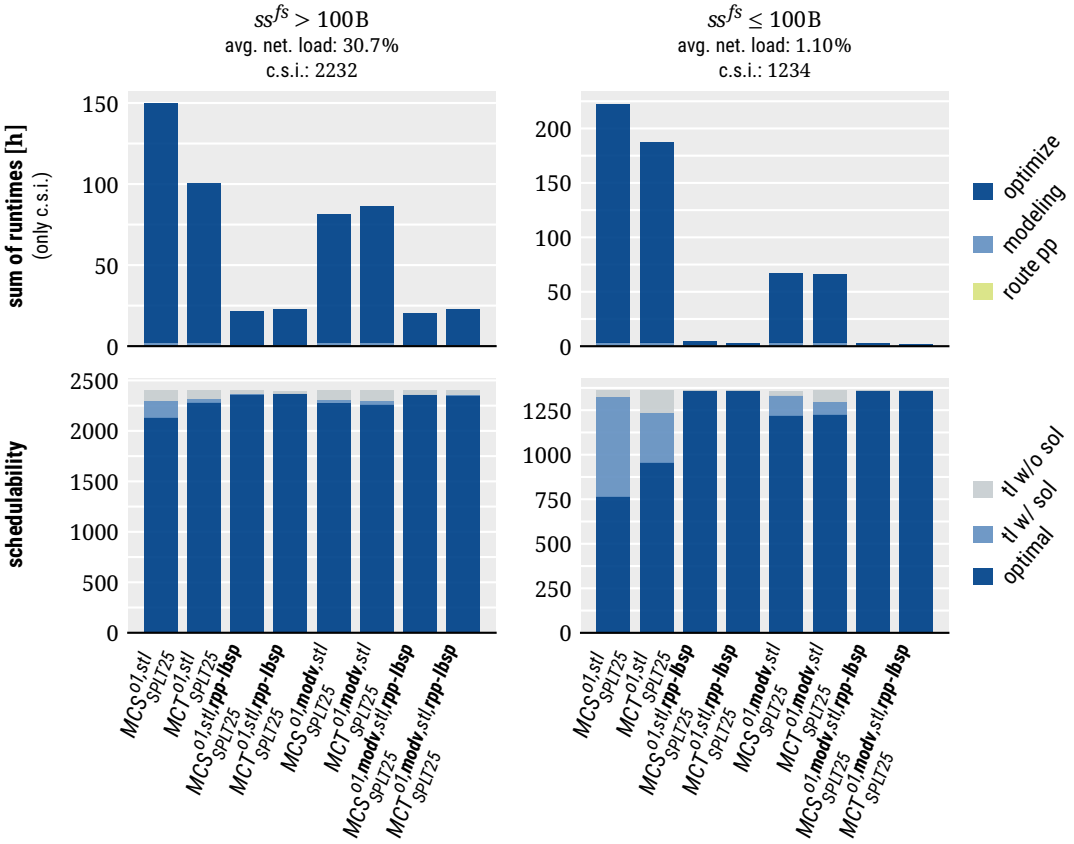


Abbildung 8.1: Schedulability und c.s.i.-Rechenzeiten von *MCS* und *MCT* für den umfassenderen Multicast-Datensatz und den optimierenden Fall.

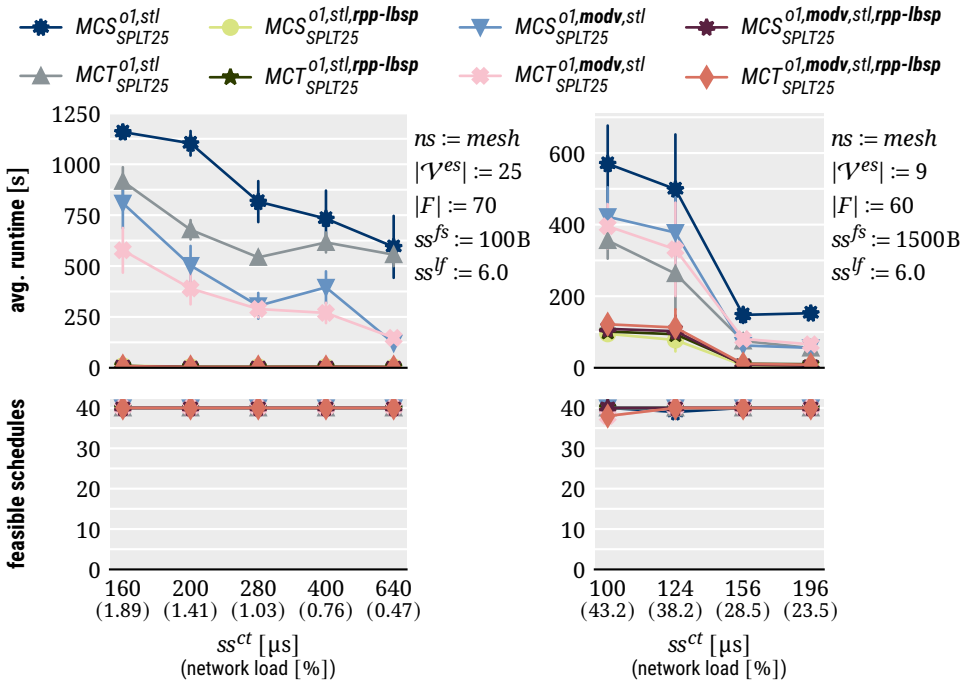


Abbildung 8.2: Ausschnitt der Ergebnisse für TC-DS (links) und TC-L (rechts) des umfassenderen Multicast-Datensatzes für die Basismodelle *MCS* und *MCT* und den optimierenden Fall.

unterscheiden, muss davon ausgegangen werden, dass *MCT* insbesondere dann vorteilhaft ist, wenn es zwecks Multicast zur Aufsplittung von Streams entlang der Route kommt.

Insgesamt ist *MCS* somit trotz der zunächst positiven Ergebnisse in Kapitel 7.1.2 das für die Weiterentwicklung am wenigsten relevante Basismodell: Für Unicaststreams steht mit *UC* ein performanteres Modell zur Verfügung, und mit *MCT* auch für Multicaststreams. Eine aussichtsreiche Modellvariante wäre demnach, je nach Empfängeranzahl zu entscheiden, welche Bedingungen genutzt werden sollen. So könnten für Multicaststreams die Bedingungen aus *MCT* und für Unicaststreams die Bedingungen aus *UC* eingesetzt werden. Eine Kombination beider Varianten innerhalb einer einzigen Probleminstanz ist dabei problemlos möglich. Dieser Ansatz wurde bereits in [E14] vorgeschlagen, jedoch bisher nicht implementiert, da stattdessen versucht wurde, durch *MCS* ein Modell zu formulieren, dass sowohl für Unicast- als auch Multicaststreams performant arbeitet und eine streamspezifische Auswahl der Bedingungen somit obsolet macht. Auch wenn der praktische Einsatz von *MCS* aufgrund der geringen Performanceeinbußen durchaus denkbar ist, deuten die bisherigen Ergebnisse jedoch darauf hin, dass die streamspezifische Kombination von *MCT* und *UC* eine performantere Modellvariante darstellen könnte.

8.2 Zusammenfassung der entwickelten ILP-Modelle

Die Entwicklung der in dieser Arbeit vorgestellten ILP-Modelle begann mit dem Unicastmodell, welches in [E12] vorgestellt wurde. Während das hier gezeigte Basismodell *UC* diesem weitgehend entspricht, wurde die Problemmodellierung gegenüber dem damaligen Entwurf um eine präzisere Modellierung von Verzögerungszeiten (d_{km}^{fwd}) und Zeitschlitzlängen (sl_{km}) erweitert. Im Gegensatz zur ursprünglichen Variante enthalten die aktuellen Modelle somit eine präzise Abbildung von CT- und SF-Bridges und berücksichtigen Bridge- und Linkeigenschaften sowie Framegrößen bei der Berechnung der entsprechenden Zeiten. Weitere Modellverbesserungen bezüglich der Performance sowie hinzugefügte Funktionen wurden in dieser Arbeit in den Kapiteln 6 und 7 vorgestellt und werden im Folgenden zusammengefasst.

8.2.1 Zusammenfassung Kapitel 6 (u.a. Multicast)

In Kapitel 6 wurden bereits für das Unicastmodell die folgenden Modelloptionen eingeführt, um frühzeitig wesentliche Performanceverbesserungen in die umfangreichen Evaluationen einzubringen:

- *bpl* reduziert die pro Stream zur Verfügung stehenden Routen.
- *rcr* entfernt redundante Ressourcenbedingungen und assoziierte Entscheidungsvariablen.
- *ia* rundet bestimmte Ausdrücke in den Schedulingbed. zwecks strengerer LP-Relaxationen.

Hierbei arbeitet nur *bpl* heuristisch, entfernt also gültige Lösungen aus dem Lösungsraum. Die Effektivität dieser Verbesserungen ist in Abbildung 8.3 gezeigt. Für UC_{SPLT25}^{bpl} zeigt sich hierbei eine Rechenzeitreduktion von 72 % bei hoher Last ($ss^{fs} > 100\mathrm{B}$) und 61 % bei niedriger Last ($ss^{fs} \leq 100\mathrm{B}$), sobald alle drei Optionen hinzugeschaltet werden. Die verbesserte Performance wird später zur

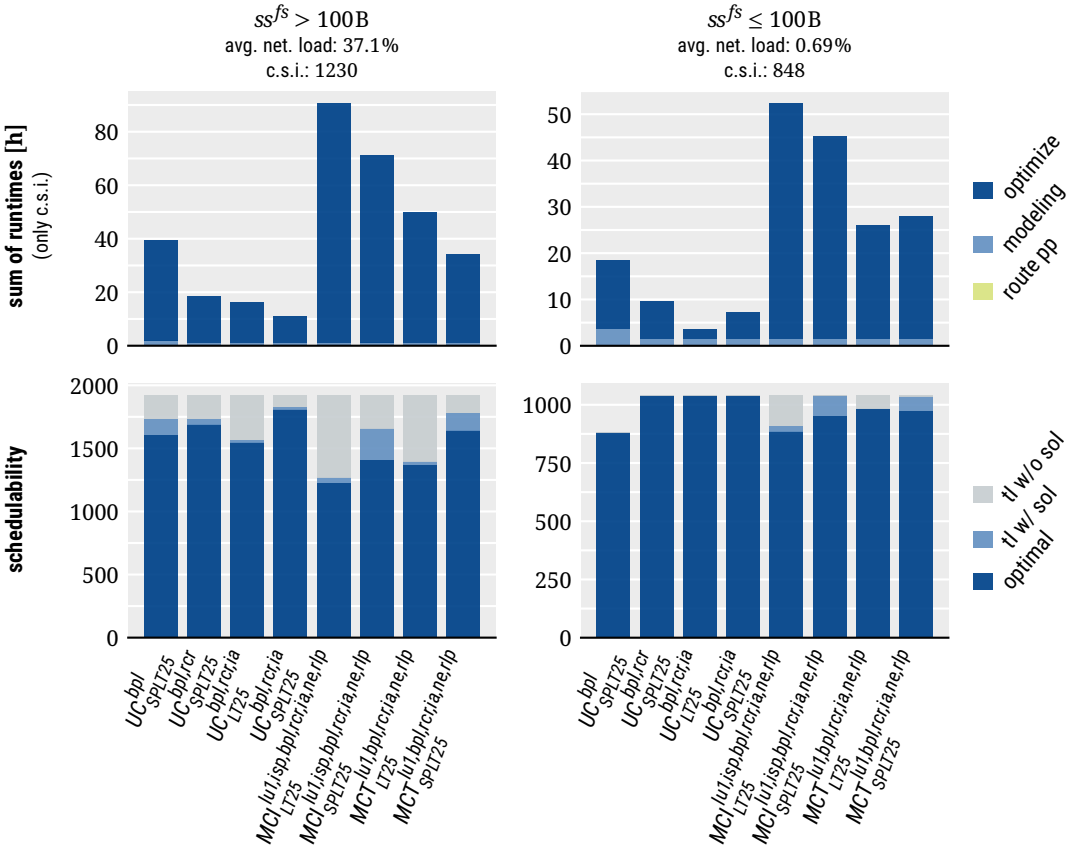


Abbildung 8.3: Schedulability und c.s.i.-Rechenzeiten bei Aktivierung von *bpl*, *rcr* und *ia* sowie Vergleich von Unicast- und Multicastmodellen.

Unterstützung von Multicaststreams genutzt, da entsprechende Modelle grundsätzlich eine etwas schlechtere Performance zeigen (vgl. Abb. 8.3) und die Modellverbesserungen daher essentiell sind, um die gleichen Szenarien wie bisher überhaupt in angemessener Zeit lösen zu können. Die Multicastunterstützung wird im Rahmen neuer Basismodelle durch angepasste Bedingungen für Routing und Pfadscheduling erreicht und zeigt im Vergleich zum Unicastmodell insbesondere für das erste Modell *MCI* eine schwächere Performance. Der Performancenachteil konnte jedoch im Modell *MCT* durch verbesserte Bedingungen für das Pfadscheduling deutlich reduziert werden (siehe Abb. 8.3), sodass der Einsatz des Modells für reale Problemgrößen denkbar ist. Insgesamt wurden in Kapitel 6 somit praxistaugliche Modelle (*UC*/*MCT*) zur Planung von Unicast- und Multicastverkehrsmustern mitsamt bedeutenden Performanceoptimierungen (*bpl*, *rcr*, *ia*) vorgestellt.

Ebenfalls seit der ersten Veröffentlichung in [E12] ergänzt wurden verschiedene ILP-Optimierungsziele. Alternativ zu der von Beginn an genutzten Optimierung der Streamlatenzen (*LT*) wurden die hierarchische Optimierung von Pfadlängen und Latenzen (*SPLT*), die Nutzung dieser Ziele mit Optimality Gap (*LT25*/*SPLT25*), das Lösen ohne Optimierungsziel (*NONE*) und die Optimierung von Pfadlängen mit Abbruchbedingung bei der ersten gefundenen Lösung (*1SP*) implementiert und evaluiert. In den Ergebnissen (Abb. 8.4) ist vor allem die drastische Verbesserung von *1SP* gegenüber *NONE* hervorzuheben, da hiermit die Bedeutung des Optimierungsziels für das Finden initialer Lösungen

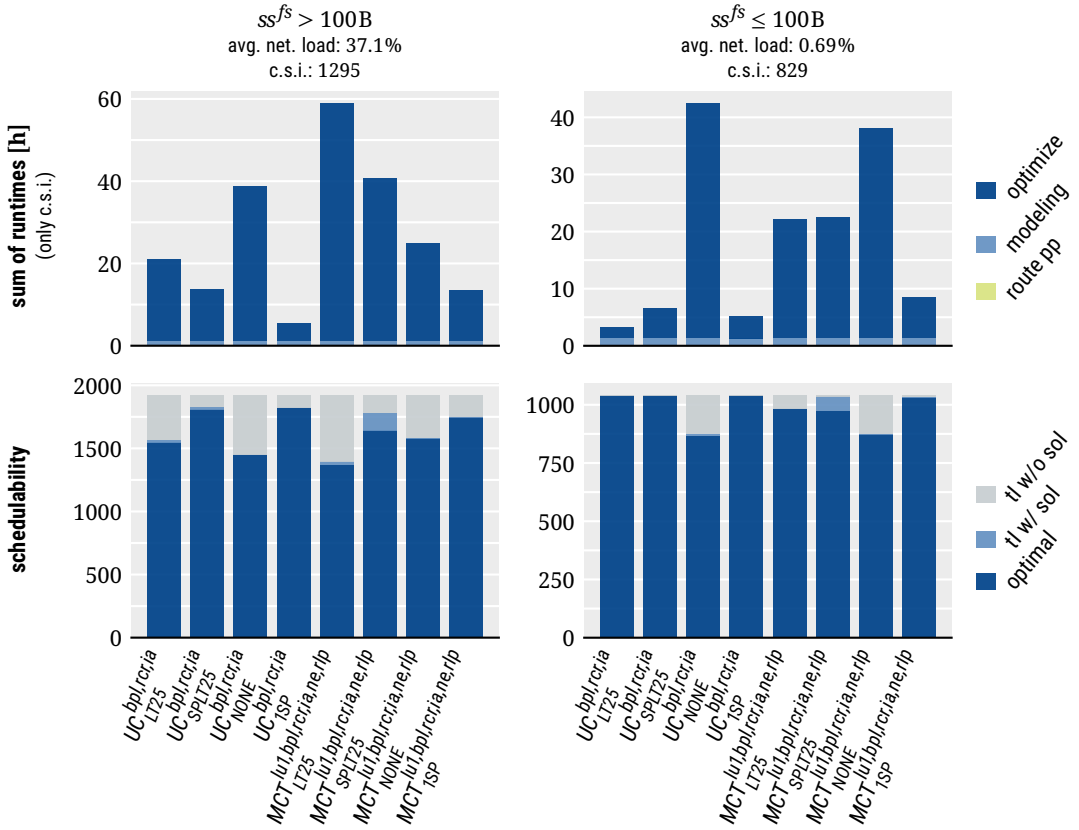


Abbildung 8.4: Schedulability und c.s.i.-Rechenzeiten für verschiedene Optimierungsziele.

gezeigt werden konnte. Dieser Einfluss zeigt sich auch im Vergleich zwischen *LT25* und *SPLT25*, wobei *SPLT25* aufgrund der besseren Schedulability in der Regel zu bevorzugen ist. Insgesamt stehen nach Kapitel 6 somit performanceverbessernde Optimierungsziele sowohl für das Finden initaler Lösungen (*1SP*) als auch für die Optimierung von Pfadlängen und Streamlatenzen (*SPLT25*) zur Verfügung.

8.2.2 Zusammenfassung Kapitel 7 (u.a. Frameisolation)

Die in Kapitel 7 eingeführten Modellvarianten werden im Folgenden zusammengefasst:

- Das neue Modell *MCS* soll die Performance des Multicastmodells weiter an das Unicastmodell *UC* annähern, wobei hierzu die Bedingungen des Pfadschedulings überarbeitet wurden.
- Die Modelloptionen *mod*/*modv* sollen die Performance bei niedriger Last verbessern, welche in den bisherigen Modellen als unerwartet schwach eingestuft wurde, insbesondere in Anbetracht der bei niedriger Last kaum vorhandenen Ressourcenbeschränkung.
- Eine (heuristische) strengere Routenvorverarbeitung (*rpp-ksp*/*rpp-lbsp*) soll die großen Rechenzeitnachteile der JRaS-Modelle gegenüber SRaS-Modellen reduzieren, ohne dabei die Vorteile bezüglich Schedulability und Lösungsgüte zu opfern.

Eine Übersicht der zugehörigen Ergebnisse ist in Abbildung 8.5 gegeben, wobei diese auf die

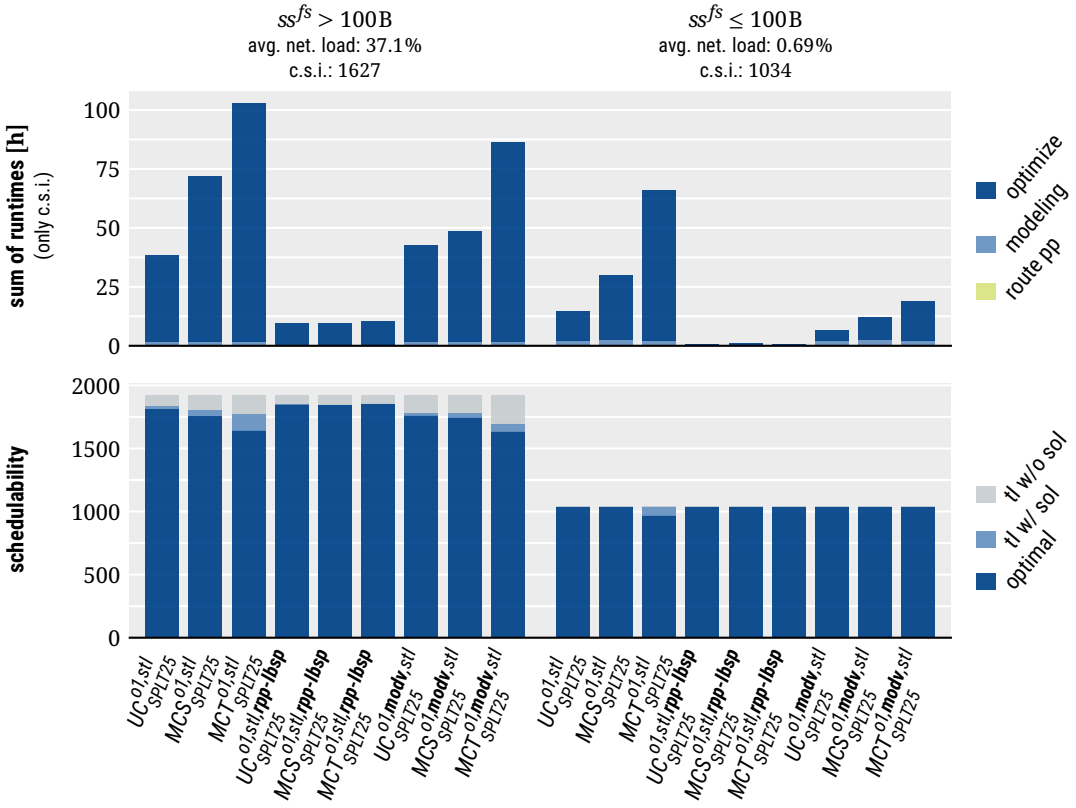


Abbildung 8.5: Schedulability und c.s.i.-Rechenzeiten für das zusätzliche Basismodell *MCS* und beim Hinzuschalten von *modv* oder *rpp-lbsp*.

Modelloptionen *modv* und *rpp-lbsp* beschränkt wurde, welche im ursprünglichen Vergleich in den Kapiteln 7.1.4/7.1.6 gegenüber den Alternativen *mod*/*rpp-ksp* die besseren Ergebnisse erzielt haben. Dabei wurde die Zielsetzung in allen drei Fällen (*MCS*, *modv*, *rpp-lbsp*) erreicht, wobei die gravierendsten Performanceverbesserungen durch *rpp-lbsp* erzielt werden.

Auffällig ist dabei, dass die Performanceunterschiede zwischen den Basismodellen unter Nutzung von *rpp-lbsp* sehr gering ausfallen. Zusammen mit der Erkenntnis aus Kapitel 8.1, dass *MCT* wiederum für Multicast-Szenarien bessere Ergebnisse als *MCS* erzielt, ist *MCS* für eine Weiterentwicklung am wenigsten relevant.

Auch die Modelloption *modv* ist trotz der verbesserten Performance bei niedriger Last nicht allgemein empfehlenswert, da dieser Verbesserung in vielen Fällen eine schlechtere Performance bei hoher Last gegenübersteht. Während sich dieser Effekt in Abbildung 8.5 bezüglich der Rechenzeiten nur für *UC* zeigt, ist er in Kombination mit *rpp-lbsp* und dem Optimierungsziel *1SP* deutlich ausgeprägt (vgl. Abb. 8.6). Des Weiteren ist die von *modv* erzielte Performanceverbesserung bei niedriger Last in Kombination mit *rpp-lbsp* bedeutend weniger ausgeprägt als ohne *rpp-lbsp*, wie der Vergleich der Abbildungen 8.5 und 8.6 zeigt. Insgesamt ist *modv* daher im Wesentlichen dann empfehlenswert, wenn Szenarien mit geringer Last und großer Routenauswahl (d.h. wenig restriktive Routenvorverarbeitung) gelöst werden sollen.

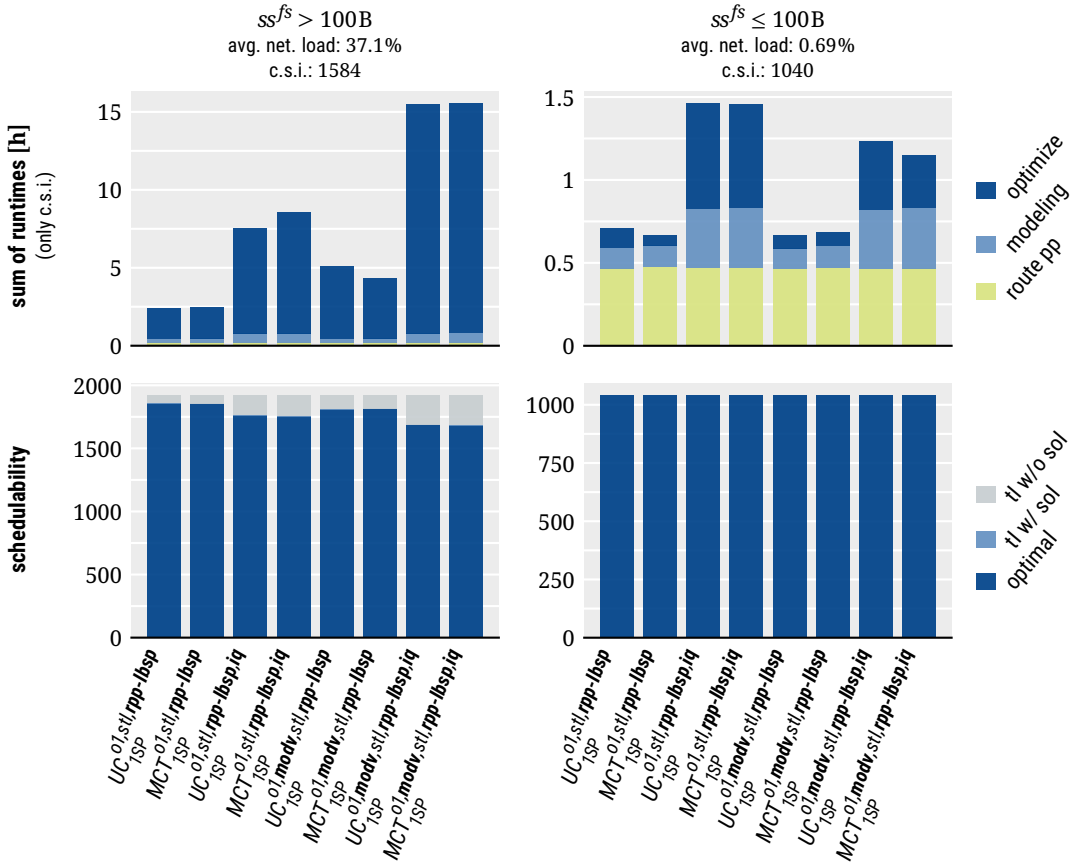


Abbildung 8.6: Schedulability und c.s.i.-Rechenzeiten beim Hinzuschalten von *modv* oder *iq* (Planung von Queuezuweisungen) unter Nutzung von *rpp-lbsp* und mit Optimierungsziel *1SP*.

Die striktere Routenvorverarbeitung *rpp-lbsp* ist dagegen nach bisherigen Tests grundsätzlich empfehlenswert, da deutliche Rechenzeitverbesserungen erzielt werden, ohne dass sich die Nachteile zeigen, die aus Ansätzen mit separaten Routing- und Schedulingschritten bekannt sind. Die konkrete Routenvorverarbeitung steht dabei in Zusammenhang mit der Netzwerktopologie, wobei die in *rpp-lbsp* genutzte Kombination aus LB- und SP-Routing im Multicast-Benchmarking gezeigt hat, dass diese Routenauswahl neben Ring- und Meshtopologien auch im Fat-Tree gute Ergebnisse erzielt.

Wie schon zuvor wurden die drastischen Performanceverbesserungen ausgenutzt, um die Modelle um weitere, komplexitätssteigernde Funktionen zu ergänzen. Konkret wurden Modelloptionen zur Sicherstellung TSN-konformer Schedules realisiert, für die an jedem Bridge-Ausgangsport Queuezuweisungen der geplanten Streams zu maximal 8 FIFO-Queues erfolgen müssen, und zudem zwecks Robustheit Bedingungen zur Frameisolation eingehalten werden müssen. Der Performancenachteil der entworfenen Erweiterungen ist in Abbildung 8.6 für die Modelloption *iq* für die beiden Basismodelle *UC* und *MCT* gezeigt. Die typische Verschlechterung der Rechenzeit liegt hier bei einem Faktor von $2-3$. Die nur geringfügige Verschlechterung der Schedulability aufgrund des Zeitlimits pro Probleminstanz, ohne Nachweis von unlösbaren Szenarien, zeigt jedoch, dass das Modell geeignet ist, in der Praxis TSN-konforme Schedules zu erstellen.

8.3 Bewertung der ILP-Modelle

Im Rahmen der Arbeit wurde eine Vielzahl von Möglichkeiten zur Modellierung von TT-RSP beschrieben und implementiert. Der modulare Aufbau (sowohl in der formalen Beschreibung als auch der Implementierung) ermöglicht die Kombination der verschiedenen Modelloptionen zu mehreren Hundert nutzbaren Modellvarianten, von denen insgesamt 289 systematisch evaluiert wurden. Die systematische Evaluation stützte sich dabei nicht nur auf die umfangreichen Benchmarkingszenarien, sondern auch auf die strategische Auswahl der genutzten Modellverbesserungen. So wurde angestrebt, frühzeitig Modellverbesserungen einzubringen, die konsequent über alle Testszenarien hinweg zu besseren Ergebnissen führen, um die Dauer darauffolgender Tests zu verkürzen. Dies hat es ermöglicht, auch Modelle mit vergrößertem Funktionsumfang (Multicast, Frameisolation) dem umfassenden Benchmarking zu unterziehen. Auch wenn diese komplexeren Modelle zwangsläufig mit Performanceeinbußen einhergehen, konnten diese durch die eingebrachten Verbesserungen soweit ausgeglichen werden, dass am Ende der Entwicklung Modelle zu Verfügung stehen, die TSN-konforme Schedules erzeugen und Multicaststreams unterstützen, und zugleich gegenüber dem Ursprungsmodell eine verbesserte Schedulability und bedeutend bessere Rechenzeiten aufweisen. Als Beispiel für die erzielten Fortschritte zeigt Abbildung 8.7 das Unicast-Benchmarking für den optimierenden Fall (*SPLT25*). Im Falle hoher Netzwerkauslastung kann das optimierte Multicastmodell (ohne Frameisolation) hier 126 Probleminstanzen mehr lösen als das initiale Unicastmodell UC^{bpl} und besitzt dabei eine um $90.48\,\%$ reduzierte c.s.i.-Rechenzeit. Selbst bei Hinzuschalten der Frameisolation (*iq*) werden noch 37 Instanzen mehr gelöst und das Modell weist eine $76.96\,\%$ geringere c.s.i.-Rechenzeit gegenüber dem initialen Modell UC^{bpl} auf. Bei niedriger Last fällt die Rechenzeitreduktion nochmals größer aus.

Die ursprüngliche Zielsetzung einer Modellierung mit möglichst umfassender *Lösungsraumabdeckung* (vgl. Kapitel 6.1) wurde dabei stets beibehalten, da mit Ausnahme der Routenvorverarbeitung keine heuristischen Modellverbesserungen eingebracht wurden. Auch wenn diese Routenvorverarbeitung einen signifikanten Beitrag zur erzielten Performance leistet, konnte zum einen durch die umfangreichen Tests nachgewiesen werden, dass die strikte Routenvorverarbeitung in der Regel keine aus SRaS-Verfahren bekannten Nachteile mit sich bringt. Zum anderen sind alle ILP-Modelle bei Bedarf (z.B. Bearbeitung bisher nicht berücksichtigter Topologien) ohne Anpassungen mit einer weniger strikten Routenvorverarbeitung nutzbar. Eine präzise *Formalisierung* konnte ebenfalls weitgehend in Form von Randbedingungen der ILP-Modelle realisiert werden. Ausnahmen sind beispielsweise in den Algorithmen des Basismodells *MCT* und der Modelloption *rpp-ksp* zu finden, welche in weniger präzisem Pseudocode beschrieben werden. Auch dieser stützt sich jedoch auf die zuvor formal eingeführten mathematischen Symbole, sodass nicht vollständig spezifizierte und ausformulierte Methoden (z.B. $\mathrm{PL}(P)$, $\mathrm{LT}(P, s)$ in Alg. 7.1) auf wenige triviale Bestandteile reduziert werden konnten. Die weiteren Ziele (Ausnutzung von Solvern, Lösbarkeitsinformationen, flexibles Optimierungsziel, Optimalitätsinformationen, Erweiterbarkeit) konnten uneingeschränkt umgesetzt werden. Insgesamt konnte das Ziel einer performanten Referenzlösung für TT-RSP erfüllt werden, und diese zudem um das Queuingmodell von TSN erweitert werden.

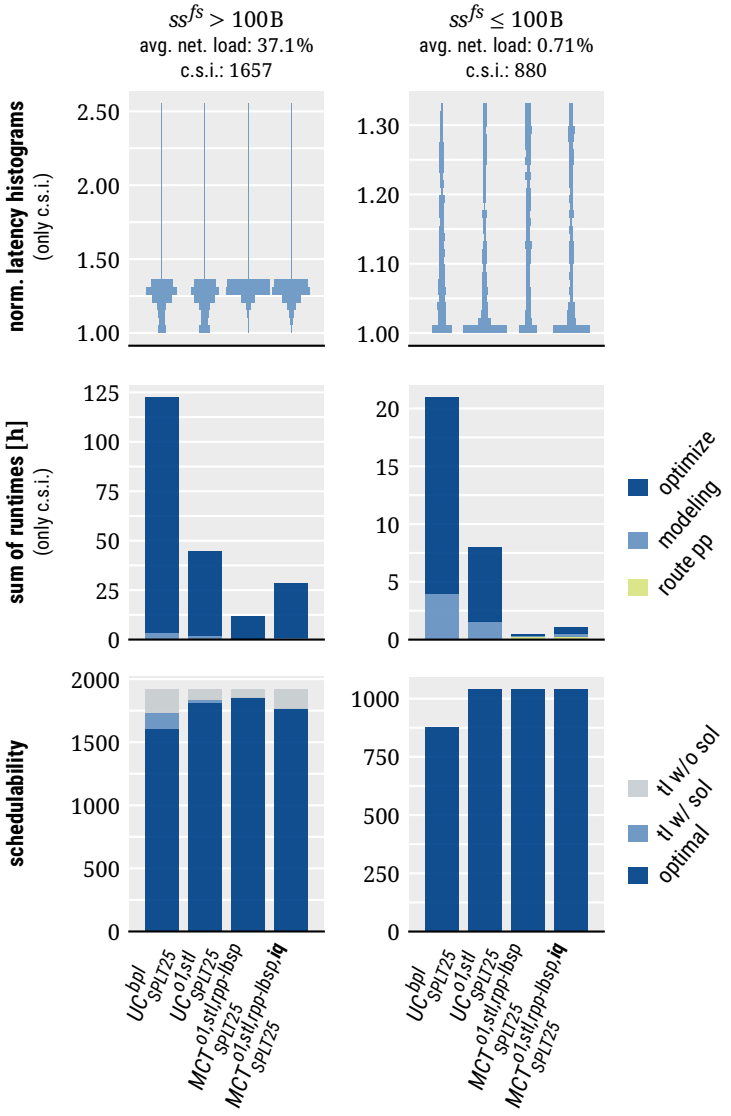


Abbildung 8.7: Schedulability und c.s.i.-Rechenzeiten für das initiale Unicastmodell und die finalen Multicastmodelle im Vergleich.

Ein direkter Performancevergleich mit Lösungen aus der Literatur wurde im Rahmen der Arbeit [E15] auf Basis einer früheren Version des Unicast-Benchmarkings durchgeführt. Dabei zeigte $UC^{bpl,rcr,ia}$ (in [E15] als *B-ILP* bezeichnet) eine vergleichbare Performance wie die ILP-Modelle aus [13], wobei das konkrete Ranking vom Testcase abhängig war. Die Scheduling-Heuristik aus [37] konnte hier nur in wenigen Spezialfällen bessere Ergebnisse erzielen als die ILP-basierten Verfahren, während der SMT-basierte, frei verfügbare Scheduler TSNSched [53] in einer ersten Testphase bereits deutlich höhere Rechenzeiten zeigte und überhaupt nicht im umfangreichen Benchmarking getestet werden konnte. Entsprechende Vergleichstests wurden für das in dieser Arbeit genutzte überarbeitete Benchmarking nicht fortgeführt, da TSNSched aus Performancegründen nicht nutzbar war, während die vorliegenden Implementierungen aus [37, 13] keine heterogenen Zykluszeiten unterstützten und

die Heuristik aus [37] auch aus Performancegründen nur wenig relevant für einen Vergleich mit den hier weiter optimierten ILP-Modellen war. Eine vollständige Nachimplementierung von anderen, hinreichend präzise beschriebenen, formalen Ansätzen aus der Literatur konnte aus Zeitgründen nicht vorgenommen werden, allerdings wurden bestimmte Teilkomponenten nachimplementiert und analysiert, was in Kapitel 10.2 weiter ausgeführt wird. Dabei werden auch Ideen aus [13] wieder aufgegriffen.

Wie in Kapitel 4.1 ausführlich erläutert, haben direkte Vergleiche zu Messungen und quantitativen Analysen aus anderen Publikationen (im Gegensatz zu den zuvor genannten direkten Vergleichstests in derselben Testumgebung) nur einen geringen Informationsgehalt. Daher lässt sich aus dem Vergleich der hier durchgeführten Analysen mit denen aus der Literatur nur die grobe Abschätzung treffen, dass die in der Literatur vorgestellten formalen Methoden in der Regel höhere Rechenzeiten bei ähnlicher Parametrisierung der Szenarien aufweisen und Heuristiken vorwiegend für große aber wenig ausgelastete Szenarien eingesetzt werden. Eine umfassende Charakterisierung der Scheduler aus der Literatur wird in Kapitel 10 vorgenommen. An dieser Stelle sei nur genannt, dass eine umfassende Diskussion und Analyse der Optimierung von ILP-Modellen nur in [13] vorgenommen wird, wobei die Überschneidung zu der hier vorliegenden Arbeit gering ausfällt und auch der Umfang insgesamt geringer ist. Zur Evaluation wurden im Rahmen dieser Arbeit insgesamt 854798 Unicast-Probleminstanzen unter Nutzung von 289 Modellvarianten in einer Gesamtrechenzeit von 5.169 Jahren gelöst, sowie 90238 Multicast-Probleminstanzen von 24 Modellvarianten in einer Rechenzeit von 0.387 Jahren. Durch die Verteilung in einem Rechencluster mit 10 baugleichen Rechnern konnte diese Evaluation in etwa 0.556 Jahren ausgeführt werden. Nach bestem Wissen des Autors existiert in der Literatur des Themengebiets bisher keine Arbeit mit vergleichbarem Umfang.

8.4 Ausblick

In dieser Arbeit wurde eine iterative Optimierung von ILP-Modellen durchgeführt, um unter Beachtung der Entwicklungsziele nach Kapitel 6.1 eine möglichst performante Lösung für TT-RSP zu erstellen, welche die mittels ILP erzielbare Performance angemessen repräsentiert und ein breites Spektrum realer Problemgrößen abdeckt. Ausgehend von den so erzielten Modellen lassen sich in der Weiterentwicklung drei wesentliche Konzepte verfolgen: (a) Implementierung von Heuristiken auf Basis der optimierten Modelle, insbesondere in Form des inkrementellen Lösens der Modelle, (b) Fortsetzung der Modelloptimierungen ohne heuristische Einschränkungen, und (c) weitere funktionale Erweiterungen.

8.4.1 Heuristiken

Bezüglich der Heuristiken ist insbesondere das inkrementelle Lösen der Modelle zu nennen. Dabei wird das ILP-Modell nicht von Beginn an vollständig aufgestellt, sondern nur für einen Teil der Streams des Verkehrsmusters. Nach dem Lösen dieses Teilproblems wird der Schedule für die entsprechenden Streams als statisch angenommen und weitere Streams des Verkehrsmusters in den bereits bestehenden Schedule integriert (*Stream-inkrementelles* Lösen). Dabei sind abgesehen

vom schrittweisen Aufbau und Lösen des Modells keinerlei Veränderungen an der ILP-Formulierung notwendig, da der Schedule von bereits geplanten Streams hierbei einfach durch die Zuweisung von konkreten Werten zu den zugehörigen Entscheidungsvariablen abgebildet wird. Im Rahmen dieser Arbeit wurden entsprechende Techniken aus verschiedenen Gründen nicht umgesetzt. Einerseits handelt es sich unabhängig von der genauen Umsetzung um Heuristiken, die viele der aufgestellten Ziele aus Kapitel 6.1 nicht erfüllen können. Andererseits sind derartige Verfahren bekanntermaßen effektiv [3], sodass eine frühzeitige Umsetzung mit einer der ersten ILP-Modelle (z.B. *UC* ohne weitere Optimierung) nur bedingt neue Erkenntnisse hervorgebracht hätte. Stattdessen sollte sich die Arbeit auf bisher kaum untersuchte Modelloptimierungen konzentrieren.

Zukünftig sollten inkrementelle Verfahren jedoch auch auf die hier vorgestellten Modelle angewendet werden, da der Performancegewinn für praktische Anwendungen nicht vernachlässigbar ist. Über den grundsätzlichen Performancegewinn hinaus wurden außerdem verschiedene Aspekte des inkrementellen Lösens noch nicht ausreichend in der Literatur untersucht. Beispielsweise wurde Backtracking (erneute Planung von zuvor bereits geplanten Streams nach einer fehlgeschlagenen Iteration) in der Literatur zu TT-RSP bisher kaum beachtet (siehe auch Kapitel 10.1.2) und das *Irreducible Inconsistent Subsystem* (IIS), welches der Solver für ungelöste Szenarien zur Verfügung stellen kann, wurde bisher überhaupt nicht berücksichtigt. Auch wurde bisher nicht versucht, im inkrementellen Lösen zwecks Performance zunächst mit strikter Routenvorverarbeitung (z.B. *rpp-lbsp* oder sogar feste Routen) zu arbeiten und den Lösungsraum erst nach Fehlschlägen um weitere Routen zu erweitern (d.h. *Routen-inkrementelles* Backtracking). Des Weiteren haben die inkrementellen Verfahren ebenso vielfältige Optimierungsmöglichkeiten wie die ILP-Formulierung selbst, was bisher nicht ausgeschöpft wurde. So beeinflusst u.a. die Anzahl der Streams pro Iteration, deren Gruppierung und auch das ILP-Optimierungsziel die Effektivität inkrementeller Ansätze (vgl. [35, S1]).

8.4.2 Modelloptimierungen

Abgesehen von der heuristisch einzustufenden Routenvorverarbeitung wurde in dieser Arbeit eine Vielzahl von Modellvarianten vorgeschlagen, die der Performanceverbesserung unter Erhalt des vollständigen Lösungsraums dienen. Dabei konnten beispielsweise mittels der Modelloptionen *rcr*/*ia*, der Weiterentwicklung von *MCI* zu *MCT* oder auch dem Wechsel des Optimierungsziels von *NONE* zu *1SP* dominierende Lösungsverfahren gefunden werden, die über alle Szenarien hinweg zu besseren Lösungen ohne nachweisbare Nachteile führen. Darüber hinaus sind die Vorteile dieser Modellvarianten auch theoretisch begründbar, da beispielsweise Redundanzen entfernt werden (*rcr*), LP-Relaxationen stärker beschränkt (*ia*, *MCT*) und Branching-Entscheidungen verbessert werden (*1SP*). Andererseits lässt sich der Einfluss von vielen Modellveränderungen (z.B. *modv*, *MCS*) nicht präzise einordnen, da Fortschritte für einige Eingangsdaten durch Nachteile in anderen Bereichen erkauft werden, und der Solver keine ausreichenden Informationen zur tiefergehenden Analyse liefert.

Aufgrund der vielen Variationsmöglichkeiten bei der Modellierung und der bisher erzielten Erfolge kann davon ausgegangen werden, dass weiteres Optimierungspotential vorhanden ist. Neben dem Performanceeinfluss weiterer Optimierungsziele (vgl. QIs in Kapitel 4.1.1) könnten beispielsweise auch weitere Darstellungsformen der Ankunfts- und Sendezeiten von Frames in Bridges untersucht

werden. Außerdem wurde die explizite Einflussnahme auf den Lösungsvorgang des Solvers mittels Branching-Prioritäten und Variablen-Hints (vgl. [72, §12.2]) bisher nicht umfassend untersucht.

Auch wenn durch Arbeiten in diesen Bereichen weitere Fortschritte zu erwarten sind, ist jedoch zu beachten, dass diese voraussichtlich einen zunehmenden Aufwand im Vergleich zu den bisher erzielten Verbesserungen erfordern. Unter anderem muss für die Arbeiten am Branching ein Wechsel des ILP-Solvers in Betracht gezogen werden, um den Lösungsvorgang nachvollziehen und Schwachstellen feststellen zu können. Insgesamt muss daher abgewogen werden, ob Modelloptimierungen oder Heuristiken in der Weiterentwicklung bevorzugt werden sollten. Während Modelloptimierungen insbesondere theoretische Aspekte wie das Verständnis des Lösungsvorgangs in den Vordergrund stellen, können durch Heuristiken wie das inkrementelle Lösen voraussichtlich auch kurzfristig signifikante Performanceverbesserungen erzielt werden, welche für den praktischen Einsatz relevanter sind.

8.4.3 Funktionserweiterung

Die bisher entwickelten Modelle lösen das allgemein gehaltene TT-RSP und stellen mittels der Optionen *bq*/*iq* auch TSN-konforme Schedules zur Verfügung. Mögliche Erweiterungen umfassen:

- Modelloptionen, die Kompatibilität zu Profinet gewährleisten, welches die IRT-Kommunikation in einer dedizierten Phase realisiert und daher ggf. die Beschränkung des Schedules auf eine bestimmte Makespan erfordert
- Integration der auf den Endgeräten auszuführenden Echtzeitanwendungen in die Planung, sodass die zeitliche Abfolge von Anwendungsausführung und Netzwerkkommunikation aufeinander abgestimmt werden kann
- Zulassen von Jitter bei der Planung, sodass im Laufe einer Hyperperiode unterschiedliche Sendezeitpunkte zulässig sind und sich somit weitere Planungsmöglichkeiten ergeben
- Integration weiterer Verkehrsklassen in die Planung, wie beispielsweise dem ebenfalls echtzeitfähigen AVB-Verkehr

Die Anwendungsintegration wurde beispielsweise schon in [12] betrachtet, jedoch mit Hilfe heuristischer Methoden gelöst. In der Bachelorarbeit [S2] wurde außerdem ein Modell zur Planung von AVB-Streams untersucht, wobei dieses bisher jedoch keine kombinierte Planung von AVB- und TT-Verkehr berücksichtigt.

Kapitel 9

Korrektheit der ILP-Modelle

Bei der Modellierung eines komplexen mathematischen Problems wie TT-RSP ergibt sich intuitiv die Frage, inwiefern sichergestellt werden kann, dass die gefundenen Schedules tatsächlich in allen Fällen den ursprünglichen Anforderungen und Schedulingbedingungen (siehe Kapitel 3) genügen. In diesem Kapitel sollen daher zunächst kleine Fehler diskutiert werden, die sich aus den Toleranzen des genutzten Solvers ergeben. Anschließend soll die Dimensionierung der M-Konstanten erläutert werden, welche in den präsentierten ILP-Modellen zur bedingten Aktivierung von Randbedingungen sowie zur Modellierung von Entweder-oder-Bedingungen genutzt wurden. Die Wahl dieser Konstanten steht in direktem Zusammenhang mit den Solver-Toleranzen und kann bei falscher Dimensionierung zu fehlerhaften Lösungen führen, in denen Randbedingungen wesentlich stärker verletzt werden, als dies alleine aufgrund der Solver-Toleranzen der Fall wäre. Abschließend werden allgemeingültige Verifikationsalgorithmen vorgeschlagen, welche unabhängig vom genutzten Lösungsverfahren zur Scheduleverifikation eingesetzt werden können.

9.1 Toleranzen des ILP-Solvers Gurobi

Je nach Lösungsverfahren kann es vorkommen, dass zurückgegebene Lösungen in geringem Ausmaß gegen ursprünglich formulierte Bedingungen verstoßen. Der in dieser Arbeit genutzte ILP-Solver Gurobi erlaubt beispielsweise standardmäßig einen Verstoß gegen Randbedingungen um 10^{-6} (Konfigurationsparameter `FeasibilityTol`) und gegen Integralitätsbedingungen um 10^{-5} (Konfigurationsparameter `IntFeasTol`). Da alle Modelle in dieser Arbeit mit einer Granularität von $1\,\mu\mathrm{s}$ gelöst wurden, entspricht Letzteres einem zulässigen Fehler von $0.01\,\mathrm{ns}$, was in den eingesetzten Topologien mit einer Linkbandbreite von $1\,\mathrm{Gbit/s}$ der Sendedauer von $0.001\,25\,\mathrm{B}$ entspräche. Die zulässige Abweichung ist demnach vernachlässigbar und es kann davon ausgegangen werden, dass kein relevanter Einfluss auf Evaluationsergebnisse besteht.

In der Praxis kann es im Kontext von Echtzeitsystemen allerdings notwendig sein, auch solche geringen Abweichungen auszuschließen, da diese potentiell zur Nichteinhaltung des eigentlich geplanten Schedules führen können (beispielsweise durch Frame-Reordering). Derartige Fehler lassen sich vermeiden, indem die Toleranzen des Solvers schon bei der Modellierung berücksichtigt werden

und die Randbedingungen (z.B. Pfadscheduling, Ressourcenbedingungen) entsprechend pessimistisch formuliert werden. Da es sich hierbei ausschließlich um das Hinzufügen von Konstanten in einigen Randbedingungen handelt, ist dies eine triviale Erweiterung der in dieser Arbeit vorgestellten ILP-Modelle. Eine entsprechende Erweiterung wurde in dieser Arbeit nicht vorgenommen, da diese nicht von Anfang an vorgesehen war (z.B. im Rahmen zuerst evaluierter Modelle aus Kapitel 6.4.2) und im Verlauf der Arbeit keine Modelle mit unterschiedlichen Toleranzen verglichen werden sollten. Für den praktischen Einsatz der Modelle sollten die Solver-Toleranzen zukünftig bei der Modellbildung berücksichtigt werden.

9.2 Numerische Probleme der ILP-Modellierung

Aus den genannten Toleranzen des Solvers ergibt sich neben dem offensichtlichen Problem geringfügiger Abweichungen auch die Gefahr deutlich größerer Fehler, ausgelöst durch die in dieser Arbeit vielfach verwendeten M-Terme (vgl. auch [72, §29.3]). Diese Art der Fehler wurde bei der Modellierung konsequent berücksichtigt und soll hier näher erläutert werden.

Als Beispiel soll hier die Ressourcenbedingung (6.7) erneut betrachtet werden. Diese fordert, dass bei gemeinsamer Nutzung von Link m durch die Streams k und l (d.h. $\mathfrak{p}_{km} = \mathfrak{p}_{lm} := 1$) sowie Planung der x-ten Zeitschlitzwiederholung von k vor der y-ten Zeitschlitzwiederholung von l (d.h. $\mathfrak{a}_{klmxy} := 1$) der Zeitschlitz von k enden muss, bevor der Zeitschlitz von l beginnt.

$$\forall (k,l,m,x,y) \in RC^{k..l} \qquad \begin{aligned} &(\boldsymbol{t}_{lm}^{\boldsymbol{mod}} + y \cdot f_l.ct) \;-\; (\boldsymbol{t}_{km}^{\boldsymbol{mod}} + x \cdot f_k.ct) \\ &\geq\; sl'_{km} \;-\; M_{6.7} \cdot (3 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm}) \end{aligned} \tag{6.7}$$

Die beschriebene Funktion wird hier gewährleistet, da der M-Term für $\mathfrak{p}_{km} = \mathfrak{p}_{lm} = \mathfrak{a}_{klmxy} := 1$ den Wert 0 annimmt und nur die notwendige Bedingung überlappungsfreier Zeitschlitze verbleibt. Durch die Solver-Toleranzen bezüglich der Integralitätsbedingungen ($\texttt{IntFeasTol} := 10^{-5}$) können die genannten Variablen in einer solchen Lösung jedoch auch den Wert

$$\mathfrak{p}_{km} = \mathfrak{p}_{lm} = \mathfrak{a}_{klmxy} \;:=\; 1 - \texttt{IntFeasTol} = 0.99999$$

annehmen. Der Term $M_{6.7} \cdot (3 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm})$ ergibt somit nicht den für diesen Fall erwarteten und notwendigen Wert von 0. Da dieser Wert von der Zeitschlitzlänge sl'_{km} subtrahiert wird, wird somit effektiv eine ungewollte Überlappung der Zeitschlitze zulässig. Als Beispiel sei $x = y := 0$ und $M_{6.7} := 10^6$. Dann ergibt (6.7) schlimmstenfalls:

$$\boldsymbol{t}_{lm}^{\boldsymbol{mod}} - \boldsymbol{t}_{km}^{\boldsymbol{mod}} \;\geq\; sl'_{km} - 10^6 \cdot (3 - 3 \cdot 0.99999) = sl'_{km} - 30\ .$$

Somit ist in diesem Fall eine Zeitschlitzüberlappung von bis zu 30 µs zulässig, obwohl die Bedingung theoretisch (d.h. für $\texttt{IntFeasTol} := 0$) gewährleisten sollte, dass der Zeitschlitz von l erst nach dem Ende des Zeitschlitzes von k geplant wird. Wählt man dagegen $M_{6.7} := 10^4$, so ergibt sich:

$$\boldsymbol{t}_{lm}^{\boldsymbol{mod}} - \boldsymbol{t}_{km}^{\boldsymbol{mod}} \;\geq\; sl'_{km} - 10^4 \cdot (3 - 3 \cdot 0.99999) = sl'_{km} - 0.3\ .$$

Da für $\boldsymbol{t}_{km}^{\boldsymbol{mod}}$ und $\boldsymbol{t}_{lm}^{\boldsymbol{mod}}$ ebenfalls Ganzzahlbedingungen gelten, kann hieraus kein Fehler in Form einer Zeitschlitzüberlappung entstehen, sofern $\lceil sl'_{km} \rceil = \lceil sl'_{km} - 0.3 \rceil$ gilt, was beispielsweise bei Nutzung der Modelloption *ia* (d.h. ganzzahliges sl'_{km}) immer erfüllt ist. Zur Fehlervermeidung sollte $M_{6.7}$ somit stets so klein wie möglich gewählt werden.

Eine *untere Grenze* für $M_{6.7}$ ergibt sich allerdings aus dem Fall, in dem die Bedingung aufgrund der getroffenen Routing- und Schedulingentscheidungen nicht aktiv sein soll (d.h., mindestens eine der Binärvariablen $\mathfrak{p}_{km}$, $\mathfrak{p}_{lm}$ oder $\mathfrak{a}_{klmxy}$ hat den Wert 0). In diesem Fall muss $M_{6.7}$ noch ausreichend groß sein, dass die Bedingung aufgrund Subtraktion des M-Terms auf der rechten Seite automatisch erfüllt wird und sich somit keinerlei Einschränkungen mehr in Bezug auf die Sendezeitpunkte $\boldsymbol{t}_{km}^{\boldsymbol{mod}}$ und $\boldsymbol{t}_{lm}^{\boldsymbol{mod}}$ ergeben. Hierfür muss der kleinstmögliche auftretende Wert auf der linken Seite der Gleichung betrachtet werden, um anhand dieses Wertes $M_{6.7}$ zu dimensioneren. Da nach der Definition aus Kapitel 6.4.2.4 bekannt ist, dass $\boldsymbol{t}_{km}^{\boldsymbol{mod}} \leq f_k.ct - 1$ und $\boldsymbol{t}_{lm}^{\boldsymbol{mod}} \geq 0$, ergibt sich der kleinstmögliche Werte der linken Seite von (6.7) zu:

$$(0 + y \cdot f_l.ct) \; - \; ((x+1) \cdot f_k.ct - 1) \; .$$

Nach Einsetzen dieser Werte in (6.7) und Umstellen nach $M_{6.7}$ ergibt sich

$$M_{6.7} \; \geq \; \frac{sl'_{km} - y \cdot f_l.ct + (x+1) \cdot f_k.ct - 1}{(3 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm})}$$

als untere Grenze für $M_{6.7}$, um (6.7) bei Bedarf trivial zu erfüllen, indem die rechte Seite der Gleichung stets kleiner wird, als dies für die linke Seite der Gleichung unter den gegebenen Wertebereichen der Variablen überhaupt möglich ist. Da dieser Ausdruck Entscheidungsvariablen enthält, welche zum Zeitpunkt der Modellbildung noch unbekannt sind, muss weiter vereinfacht werden. Zunächst kann eine strengere untere Grenze gebildet werden, die zudem stets positiv wird. Dies wird durch Weglassen aller negativen Terme im Zähler erreicht:

$$M_{6.7} \; \geq \; \frac{sl'_{km} + (x+1) \cdot f_k.ct}{(3 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm})} \; \geq \; \frac{sl'_{km} - y \cdot f_l.ct + (x+1) \cdot f_k.ct - 1}{(3 - \mathfrak{a}_{klmxy} - \mathfrak{p}_{km} - \mathfrak{p}_{lm})} \; .$$

Die pessimistischste (d.h. größte) untere Grenze ergibt sich nun für den Fall, dass nur eine Entscheidungsvariable im Nenner den Wert 0 annimmt (die anderen beiden nehmen den Wert 1 an, da es sich um Binärvariablen handelt), was die finale untere Grenze

$$M_{6.7} \; \geq \; sl'_{km} + (x+1) \cdot f_k.ct$$

liefert. Es sei angemerkt, dass dieser letzte Schritt nur dank des positiven Zählers möglich ist, weshalb der vorige Schritt notwendig war. Die hier gezeigte Vorgehensweise wurde gleichermaßen zur Dimensionierung aller M-Konstanten angewendet und eine Auflistung aller Konstanten findet sich in Anhang B.4.

Es sei angemerkt, dass alle M-Konstanten dabei so klein wie möglich gewählt wurden, ohne jedoch explizit zu überprüfen, ob dies auch *ausreichend klein* zur Fehlervermeidung durch ein zu großes M ist. Um auszuschließen, dass fehlerhafte Schedules unbemerkt bleiben, kam stattdessen konsequent die im Folgenden erläuterte Verifikation zum Einsatz.

9.3 Verifikation von Schedules

Einschließlich der bereits diskutierten Solver-Toleranzen gibt es die folgenden potentiellen Ursachen für fehlerhafte Schedules:

- *Modellierungsfehler* im Rahmen entworfener Algorithmen und formaler Modelle lassen andere Lösungen zu als erwartet. Als einfaches Beispiel könnte hier dienen, dass ein Multicastmodell ähnlich zu *MCT* auf Basis von Alg. 6.2 (*rlp* := `true`), aber in Kombination mit Routingbedingung (6.10) (anstelle von (6.11)) gebildet wird. Dieses würde unvollständige Routen nach Abbildung 6.9 als Lösung zulassen, ohne dass dies offensichtlich ist und ohne dass dies in einfachen Testszenarien (z.B. beim Debugging) zwangsläufig auftreten würde.
- Die begrenzte *Rechengenauigkeit* von ILP-Solvern (oder auch anderen Methoden) kann zu akkumulierten Fehlern in nicht tolerierbarem Ausmaß führen. Dies wurde im Rahmen der Diskussion der M-Terme gezeigt, wo beispielsweise eine zu große Konstante in den Ressourcenbedingungen zu überlappenden Zeitschlitzen führen kann.
- *Falschantworten* von Solvern können zu fehlerhaften Lösungen führen. Diese sind von Problemen durch begrenzte Rechengenauigkeit zu unterscheiden, da die begrenzte Rechengenauigkeit zum erwarteten Verhalten der Solver gehört und entsprechend behandelt werden kann. Falschantworten befassen sich dagegen mit fehlerhaften Lösungen die nicht im Rahmen des korrekten und erwartbaren Verhaltens der Solver liegen und die somit wesentlich größere Fehler enthalten können.[1]

Die in der Literatur vorherrschenden sowie auch die in dieser Arbeit bisher gezeigten Analysen von Performance- und Qualitätsindikatoren liefern über derartige Probleme keinerlei Informationen. Ohne weitere Maßnahmen besteht also die Gefahr, dass entsprechende Analysen auf Basis falscher Schedulingergebnisse durchgeführt werden. In der Literatur wird dieses Problem trotz der großen Bedeutung für die Entwicklung von Schedulern nur in wenigen Ausnahmefällen überhaupt diskutiert.

So wird in [8, 54] beispielsweise eine Verifikation mittels Simulation angestrebt. Auch wenn dies hier als Zwischenschritt vor dem Ausrollen geplanter Netzwerkkonfigurationen in realen Testbeds prinzipiell als sinnvoll angesehen wird, ergeben sich bei der Nutzung einer Simulation als Verifikation verschiedene Probleme. Unter anderem entstehen zusätzliche Abhängigkeiten und ein vergleichsweise hoher Overhead in der Implementierung, da Szenarien und Schedules in die Formate des jeweiligen Simulators überführt werden müssen. In [8] wird die Verifikation daher auch nicht vollständig für alle Ergebnisse ausgeführt, sondern nur für ausgewählte Szenarien. Außerdem ist der Nutzen als Verifikation unklar, da eine zusätzliche Analyse der Simulationsmodelle (z.B. von Bridges) notwendig wäre, um zu verstehen, wie sich das Netzwerk im Fehlerfall (z.B. durch fehlerhafte Schedules) verhält und um diese Fehler zuverlässig identifizieren zu können. Unter Umständen können Fehler auch unentdeckt bleiben. Überlappend geplante Zeitschlitze könnten je nach Modellierung von Bridges beispielsweise einfach zu einem sequentiellen Senden zugehöriger Frames führen, ohne dass dies sich explizit durch ein Fehlverhalten der Simulation äußert.

[1] Im Rahmen dieser Arbeit ist dieser Fall tatsächlich aufgetreten und musste durch die Entwickler von Gurobi behoben werden. Entsprechende Bugtracker-Einträge sind nicht öffentlich einsehbar.

Alternativ wird in [31] vorgeschlagen, dass die von den Autoren zum Lösen von TT-RSP entworfenen SMT-Modelle auch zur Verifikation von Lösungen anderer Scheduler genutzt werden können (eine Verifikation der mit diesem SMT-Modell erzeugten eigenen Ergebnisse durch ein alternatives Verfahren wird von den Autoren dagegen nicht erwähnt). Als Argument für die Nutzung von SMT-Modellen zur Verifikation wird dabei angebracht, dass diese besonders geeignet sind, die Randbedingungen in standardisierter Form zu repräsentieren. Dies stimmt im Wesentlichen mit der in Kapitel 6.1.1 angebrachten Argumentation für die Nutzung von ILP-Modellen überein, welche ebenfalls die formale Beschreibungsform als Vorteil bezüglich Eindeutigkeit und Reproduzierbarkeit eingestuft hat. Aufgrund dieser Vorteile werden SMT-Solver bereits in verschiedenen Bereichen der Hardware- und Softwareentwicklung eingesetzt und der Vorschlag aus [31] diese zur Scheduleverifikation einzusetzen als sinnvoll angesehen. In dieser Arbeit wurde der SMT-basierte Verifikationsansatz jedoch aus zwei Gründen vermieden. Zum einen sollten zur einfacheren Implementierung zusätzliche Abhängigkeiten (SMT-Solver) vermieden werden. Zum anderen ist die Überschneidung zwischen dem zu überprüfenden Lösungsverfahren (hier: ILP-basiert) und einer SMT-basierten Verifikation unter Umständen problematisch: Eine naheliegende Vorgehensweise wäre beispielsweise, die Bedingungen des ILP in ein SMT-Modell mit *Linear Integer Arithmetic* als Hintergrundtheorie zu überführen. Viele der bei der Modellierung vorgenommenen Abstraktionen (z.B. Repräsentation von Entscheidungen als Binärvariable, Umformulierung von modulo-Operationen), sowie die grundlegende Struktur des Modells bleiben dabei jedoch gleich. Auch wenn SMT die Modellierung durch logische Verknüpfung (`and`/`or`) von Bedingungen (anstelle von M-Termen und binären Hilfsvariablen) gegenüber ILP vereinfacht, besteht somit die Gefahr, dass *Modellierungsfehler* aus dem ILP im SMT-Modell wiederholt werden. Diese Vorgehensweise ist hier daher aufgrund der fehlenden *Unabhängigkeit* der beiden Modelle als Verifikation ungeeignet.

Aufgrund der Schwächen der genannten Verifikationsansätze wurde im Rahmen dieser Arbeit eine Verifikation entworfen, die in der Implementierung keine neuen Abhängigkeiten zur Folge hat, und die zudem eine weitgehend unabhängige Entwicklung von ILP-Modellen und Verifikation erlaubt. Hierzu wurden Algorithmen entworfen und direkt in Python implementiert, welche somit nicht der limitierten *Sprache* der ILP/SMT-Modelle unterliegen.

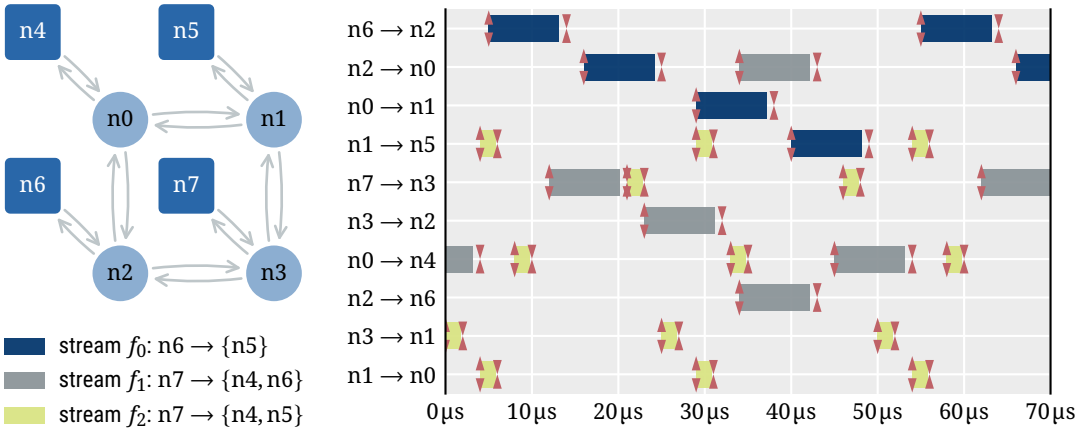
9.3.1 Pfadscheduling, Routing- und Anwendungsbedingungen

Der Sendezeitplan eines Streams k sei repräsentiert durch eine Menge von Pfadelementen PE_k, welche sich im Falle der ILP-Modelle dieser Arbeit direkt aus der ILP-Lösung ableiten lässt (vgl. Kapitel 6.4.1):[2]

$$PE_k := \left\{ \left(m, \left(\boldsymbol{t}^{\boldsymbol{abs}}_{km}, \boldsymbol{t}^{\boldsymbol{abs}}_{km} + sl_{km}\right)\right) \;\middle|\; m \in E_k, \mathfrak{p}_{km} = 1 \right\}$$

Jedes Pfadelement $pe \in PE_k$ ist ein Tupel $pe := (pe.m, pe.t)$, wobei $pe.m \in \mathcal{E}$ den genutzten Link und $pe.t$ den darauf vorgesehenen Sendezeitschlitz als Tupel aus Startzeit $pe.t_0$ und Endzeit $pe.t_1$

[2] Auch wenn die Eingangsdaten der Verifikation (z.B. PE_k) in diesem Kapitel anhand der ILP-Modelle formuliert werden, können äquivalente Daten auch aus den Schedules anderer Lösungsverfahren erstellt und die Verifikation demnach auch für andere Lösungsverfahren verwendet werden.



$$PE_1 := \{((\text{n7}, \text{n3}), (12\,\mu\text{s}, 20.16\,\mu\text{s})), ((\text{n3}, \text{n2}), (23\,\mu\text{s}, 31.16\,\mu\text{s})), ((\text{n2}, \text{n0}), (34\,\mu\text{s}, 42.16\,\mu\text{s})), ((\text{n0}, \text{n4}), (45\,\mu\text{s}, 53.16\,\mu\text{s})), ((\text{n2}, \text{n6}), (34\,\mu\text{s}, 42.16\,\mu\text{s}))\}$$

$$PE^{\rightarrow}_{1,\text{n7}} := \{((\text{n7}, \text{n3}), (12\,\mu\text{s}, 20.16\,\mu\text{s}))\}$$

$$PE^{\rightarrow}_{1,\text{n3}} := \{((\text{n3}, \text{n2}), (23\,\mu\text{s}, 31.16\,\mu\text{s}))\}$$

$$PE^{\rightarrow}_{1,\text{n2}} := \{((\text{n2}, \text{n0}), (34\,\mu\text{s}, 42.16\,\mu\text{s})), ((\text{n2}, \text{n6}), (34\,\mu\text{s}, 42.16\,\mu\text{s}))\}$$

$$PE^{\rightarrow}_{1,\text{n0}} := \{((\text{n0}, \text{n4}), (45\,\mu\text{s}, 53.16\,\mu\text{s}))\}$$

$$V_1^{sol} := \{\text{n7}, \text{n3}, \text{n2}, \text{n0}, \text{n4}, \text{n6}\}$$

Abbildung 9.1: Formale Lösungsbeschreibung für Stream f_1 des Beispiels aus Kapitel 1.

enthält. Zur Veranschaulichung zeigt Abbildung 9.1 die formale Beschreibung des Sendezeitplans für einen Stream des Beispiels aus Kapitel 1. Basierend auf diesem Sendezeitplan von k werden hier die Knoten- und Kantenmengen

$$V_k^{sol} := \{i \in \mathcal{V} \mid \exists pe \in PE_k : i \in pe.m\} \qquad \text{und} \qquad PE^{\rightarrow}_{ki} := \{pe \in PE_k \mid pe.m_0 = i\}$$

definiert. V_k^{sol} enthält demnach alle Knoten der Route von k, während $PE^{\rightarrow}_{ki}$ alle in PE_k enthaltenen Pfadelemente umfasst, deren Kante von i ausgeht. Dies wird in Abbildung 9.1 ebenfalls für einen Beispielstream dargestellt.

Basierend auf diesen Definitionen kann Algorithmus 9.1 genutzt werden, um die Korrektheit des Sendezeitplans PE_k zu prüfen. Der Algorithmus erkennt fehlerhafte Routen in Form von Sackgassen (Pfadenden die nicht zu einem Empfänger führen), redundanten Pfaden, Schleifen, unverbundenen Empfängern und isolierten (unverbundenen) Links. Das Scheduling wird hinsichtlich der Bridge- und Leitungsverzögerungen (Pfadscheduling) und maximal zulässiger Latenz des Streams (Anwendungsbedingung) geprüft. Dabei werden Abweichungen innerhalb einer vorgegebenen Genauigkeit $precision$ akzeptiert, damit Fehler im Rahmen der Solver-Toleranzen noch als gültige Lösungen gelten. Wie in Kapitel 9.1 bereits diskutiert, sollten diese Toleranzen zukünftig bereits bei der Modellbildung berücksichtigt werden. In diesem Fall könnte die Fehlertoleranz aus der Verifikation entfernt werden. In dieser Arbeit erfolgte die Verifikation mit einer Toleranz von $precision := 10^{-4}$, was $0.1\,\text{ns}$ bzw. der Sendezeit von $0.0125\,\text{B}$ entspricht.

Der Ablauf des Algorithmus lässt sich folgendermaßen zusammenfassen: Es wird am Sender gestartet und von dort aus werden iterativ alle erreichbaren Pfadelemente aus PE_k durchlaufen.

```
1  i := f_k.src
2  V^visited := {i}                                   // init: set of visited nodes contains source only
3  PE^predec := ()                                    // init: empty sequence of predecessor path elements
4  next      := {(pe, PE^predec) | pe ∈ PE⃗_ki}        // init: set of tuples of (reachable path element, predecessors)
5  while next ≠ ∅ do
6      pe, PE^predec := pop(next)                     // remove and unpack a tuple from next
7      m := pe.m                                      // current link
8      i := m_1                                       // newly reached node via m
9      foreach pe^new ∈ PE⃗_ki do                      // iterate path elements that became reachable via i
10         if pe^new.t_0 < pe.t_0 + d^fwd_km − precision then
11             report path scheduling violation
12         if pe^new ∈ PE^predec then
13             report loop routing
14     if i ∈ V^visited then
15         report redundant routing
16     if PE⃗_ki = ∅ and i ∉ f_k.dsts then
17         report dead end
18     if i ∈ f_k.dsts then
19         pe^first := PE^predec_0                    // get first path element of current routing
20         if pe.t_0 + d^rcv_km − pe^first.t_0 > f_k.ml + precision then
21             report end-to-end latency violation
22     PE^predec := PE^predec ++ (pe)                 // create new sequence of predecessors including pe
23     next      := next ∪ {(pe^new, PE^predec) | pe^new ∈ PE⃗_ki}
24     V^visited := V^visited ∪ {i}
25 if f_k.dsts ⊈ V^visited then
26     report unvisited destination
27 if V^sol_k ≠ V^visited then
28     report unconnected path element
```

Algorithmus 9.1: Verifikation des Sendezeitplans PE_k von Stream k.

Dabei werden bereits besuchte Knoten $V^{visited}$ gespeichert, um redundante Pfade zu erkennen, und es werden separat für jede Verzweigung der Route (zwecks Multicast) die bereits besuchten Pfadelemente PE^{predec} gespeichert, um Schleifen zu erkennen. Beim Erreichen eines jeden neuen Pfadelements wird das Pfadscheduling bezüglich des Vorgängers geprüft, während immer beim Erreichen eines Empfängers die Einhaltung der Latenzanforderung zwischen erster und letzter Kante des Pfades sichergestellt wird. Pfadelemente, die keine Nachfolger besitzen und nicht zu einem Empfänger führen, werden als Sackgassen erkannt, wohingegen unverbundene Empfänger und unnötigerweise in PE_k enthaltene (d.h. isolierte) Pfadelemente zum Abschluss durch den Vergleich besuchter Knoten $V^{visited}$ mit $f_k.dsts$ bzw. V^{sol}_k identifiziert werden.

Der Ablauf des Algorithmus soll anhand des in Abbildung 9.1 eingeführten Beispielstreams f_1 verdeutlicht werden. Nach der Initialisierung (bis inkl. Z. 4) ergibt sich hier:

$$i \coloneqq \mathrm{n7} \qquad V^{visited} \coloneqq \{\mathrm{n7}\} \qquad PE^{predec} \coloneqq () \qquad next \coloneqq \{(((\mathrm{n7},\mathrm{n3}),(12\,\mu\mathrm{s},20.16\,\mu\mathrm{s})),())\}$$

Es sei angemerkt, dass dabei alle runden Klammern Tupel/Sequenzen darstellen und somit relevant für die Korrektheit des jeweiligen Ausdrucks sind. Sequenzen mit nur einem Element werden zusätzlich mit nachgestelltem Komma kenntlich gemacht. Der erste Schleifendurchlauf (Z. 5) betrachtet die über Link $(\mathrm{n7},\mathrm{n3})$ neu erreichbaren Pfadelemente. Nach Ablauf bis Z. 8 ergibt sich:

$$pe \coloneqq ((\mathrm{n7},\mathrm{n3}),(12\,\mu\mathrm{s},20.16\,\mu\mathrm{s})) \qquad PE^{predec} \coloneqq () \qquad next \coloneqq \{\} \qquad m \coloneqq (\mathrm{n7},\mathrm{n3}) \qquad i \coloneqq \mathrm{n3}$$

In Z. 9 sind demnach die von $\mathrm{n3}$ aus neu erreichbaren Pfadelemente $PE^{\rightarrow}_{1,\mathrm{n3}}$ zu betrachten, was hier nur das Element $((\mathrm{n3},\mathrm{n2}),(23\,\mu\mathrm{s},31.16\,\mu\mathrm{s}))$ ist (siehe Abb. 9.1). Dieses als pe^{new} referenzierte Pfadelement erfüllt die Weiterleitungsverzögerung gegenüber dem Vorgänger pe (Z. 10), die hier laut Topologie- und Streamparametern $d^{fwd}_{1,(\mathrm{n7},\mathrm{n3})} \coloneqq 10.064\,\mu\mathrm{s}$ beträgt. Außerdem ist es nicht in den (noch leeren) Vorgängern PE^{predec} enthalten (Z. 12). Es ist ersichtlich, dass auch alle weiteren Überprüfungen (Z. 14–21) keine Fehler ergeben. Nach Ausführung von Z. 22–24 ergibt sich:

$$PE^{predec} \coloneqq (((\mathrm{n7},\mathrm{n3}),(12\,\mu\mathrm{s},20.16\,\mu\mathrm{s})),) \qquad next \coloneqq \{(((\mathrm{n3},\mathrm{n2}),(23\,\mu\mathrm{s},31.16\,\mu\mathrm{s})),(((\mathrm{n7},\mathrm{n3}),(12\,\mu\mathrm{s},20.16\,\mu\mathrm{s})),))\}$$
$$V^{visited} \coloneqq \{\mathrm{n7},\mathrm{n3}\}$$

Danach startet die nächste Iteration, welche die von Link $(\mathrm{n3},\mathrm{n2})$ aus neu erreichbaren Pfadelemente auf gleiche Weise untersucht. Der Kürze halber werden weitere Iterationen hier nicht beschrieben. Es ist allerdings für das gegebene Beispiel leicht erkennbar, dass sich abschließend ein $V^{visited}$ ergibt, für das auch die letzten Überprüfungen (Z. 25–28) keine Probleme feststellen.

9.3.2 Ressourcenbedingungen und Queuezuweisungen

Im Gegensatz zum Pfadscheduling ist es bei der Überprüfung der Ressourcenbedingungen hilfreich, die Zeitschlitze der Streams zunächst in den Wertebereich der eigenen Zykluszeit zu projizieren. Dies kann entweder auf Basis von PE_k erfolgen, oder, wie hier gezeigt, direkt der ILP-Lösung entnommen werden. Darüber hinaus werden die Pfadelemente hier um die Queuezuweisung ergänzt, welche (sofern vom Scheduler vorgenommen) ebenfalls verifiziert werden muss. Hier wird zur Vereinfachung die Modellierung *iq* angenommen, sodass diese in der ILP-Lösung direkt als $\mathfrak{qi}_{km}$ vorliegt:

$$PE^{mod}_k \coloneqq \left\{(m,(\boldsymbol{t}^{\boldsymbol{mod}}_{km},\boldsymbol{t}^{\boldsymbol{mod}}_{km}+sl_{km}),\mathfrak{qi}_{km}) \;\middle|\; m \in E_k, \mathfrak{p}_{km}=1\right\}$$

Im Folgenden werden die Elemente eines Tupels $pe \in PE^{mod}_k$ auch mit $(pe.m, pe.t, pe.q)$ referenziert. Ein Beispiel bzgl. PE^{mod}_k ist in Abb. 9.2 gegeben. Basierend auf PE^{mod}_k lässt sich die Nutzung LU_m eines Links m für eine Hyperperiode $HP \coloneqq \mathrm{lcm}^{\{\}}(\{f_k.ct \mid k \in F\})$ folgendermaßen beschreiben:[3]

$$LU_m \coloneqq \left\{(k,(pe.t_0+x\cdot f_k.ct, pe.t_1+x\cdot f_k.ct), pe.q) \;\middle|\; k\in F, pe\in PE^{mod}_k, x\in\mathbb{N}^0, pe.m=m, x\le {}^{HP}/_{f_k.ct}\right\}$$

[3] $\mathrm{lcm}^{\{\}}(X)$ sei eine Funktion, die das kleinste gemeinsame Vielfache aller Zahlen der Menge X berechnet.

$$PE_2^{mod} := \{((\mathrm{n7},\mathrm{n3}),(21\,\mu s, 22.76\,\mu s),0),\ ((\mathrm{n3},\mathrm{n1}),(0\,\mu s, 1.76\,\mu s),0),\ ((\mathrm{n1},\mathrm{n5}),(4\,\mu s, 5.76\,\mu s),0),\ ((\mathrm{n1},\mathrm{n0}),(4\,\mu s, 5.76\,\mu s),0),\ ((\mathrm{n0},\mathrm{n4}),(8\,\mu s, 9.76\,\mu s),0)\}$$

$$LU_{(\mathrm{n0},\mathrm{n4})} := \{(1,(45\,\mu s, 53.16\,\mu s),0),\ (1,(95\,\mu s, 103.16\,\mu s),0),\ (2,(8\,\mu s, 9.76\,\mu s),0),\ (2,(33\,\mu s, 34.76\,\mu s),0),\ (2,(58\,\mu s, 59.76\,\mu s),0)\}$$

Abbildung 9.2: Projizierte Zeitschlitze für Stream f_2 sowie Nutzung von $(\mathrm{n0},\mathrm{n4})$ für das Beispiel aus Abb. 9.1.

Demnach enthält LU_m die Nutzung von m durch Zeitschlitze aller Streams einschließlich ihrer periodischen Wiederholungen innerhalb von HP. Dies entspricht im Wesentlichen den Zeitschlitzen, die auch bei der Visualisierung einer Hyperperiode des Schedules für Link m dargestellt werden. In der Visualisierung werden ggf. noch weitere Zeitschlitzwiederholungen dargestellt, wie z.B. solche, die vor dem Zeitpunkt 0 µs liegen (vgl. $(\mathrm{n0},\mathrm{n4})$ in Abb. 9.1). Eine einzelne Linknutzung $lu \in LU_m$ ist ein Tupel $lu := (lu.k, lu.t, lu.q)$, welches den zugehörigen Stream $lu.k$, Start- und Endzeit $lu.t_0/lu.t_1$ der Zeitschlitzwiederholung und die vorgesehene Queue $lu.q$ enthält. Ein Beispiel für diese formale Beschreibung der Linknutzung ist in Abbildung 9.2 gezeigt. Die Verifikation von LU_m bezüglich der Ressourcenbedingungen (d.h. Zeitschlitzkonflikte) kann durch Algorithmus 9.2 erfolgen, welcher über die nach Zeitschlitzanfang geordnete Sequenz der Linknutzungen iteriert und jeweils für zwei aufeinanderfolgende Zeitschlitze prüft, dass diese nicht überlappen.

Die Verifikation der Queuezuweisungen muss sicherstellen, dass für Frames, die der gleichen Queue eines Ausgangsports zugewiesen sind, kein Reordering stattfindet. Außerdem muss in diesem Fall auch die zeitliche Frameisolation für zwei aufeinanderfolgende Frames gewährleistet sein, der zweite Frame darf also erst nach dem Gate-Close-Event des ersten sendebereit sein. Hierfür ist nicht nur die Linknutzung LU_m relevant, sondern es muss zu jedem der dort eingetragenen Zeitschlitze auch bekannt sein, wie lange dieser schon bereitsteht. Dies wird mit Hilfe der Sendezeitpunkte auf den jeweiligen Vorgängerlinks ermittelt und hier formal in

$$wt_{km} := \sum_{\substack{pe \in PE_k | \\ pe.m = m}} pe.t_0 \; - \sum_{\substack{pe \in PE_k | \\ pe.m_1 = m_0}} (pe.t_0 + d_{pe.m}^{min})$$

erfasst, wobei wt_{km} die Wartezeit eines Frames von Stream k in Bridge m_0 vor dem Sendevorgang auf m beschreibt. In diesem Ausdruck enthält die erste Summe per Definition von PE_k nur ein Element und modelliert somit den Zeitschlitzanfang von k auf m. Die zweite Summe enthält ebenfalls

```
1 LU_m^seq := sort(LU_m)                    // sort usage tuples lu by slot start lu.t_0; return as sequence
2 foreach n ∈ ℕ^0, n < |LU_m^seq| − 1 do    // iterate sequence LU_m^seq by index
3     lu^first := LU_m^seq_n                // get n-th element from LU_m^seq
4     lu^second := LU_m^seq_{n+1}           // get (n+1)-th element from LU_m^seq
5     if lu^first.t_1 > lu^second.t_0 then
6         report overlapping time slot
```

Algorithmus 9.2: Verifikation der Ressourcenbedingungen auf Link m.

```
1  foreach q ∈ ℕ⁰, q < q_m^no do                                  // iterate available queues
2      QU := {lu ∈ LU_m | lu.q = q}                               // get link usages with queue q
3      QU^seq := sort(QU)          // sort usage tuples lu by slot start lu.t_0; return as sequence
4      foreach n ∈ ℕ⁰, n < |QU^seq| − 1 do                        // iterate sequence QU^seq by index
5          lu^first := QU_n^seq                                   // get n-th element from QU^seq
6          lu^second := QU_{n+1}^seq                              // get (n+1)-th element from QU^seq
7          k := lu^second.k                                       // get stream of lu^second; needed to find wait time
8          if lu^first.k ≠ lu^second.k and
9            ⌈lu^first.t_1⌉ > lu^second.t_0 − wt_km then
10             report frame isolation violation
```

Algorithmus 9.3: Verifikation der Queuezuweisungen auf Link m.

nur ein Element, da redundante Pfade unzulässig sind (Überprüfung erfolgte in Alg. 9.1) und daher nur ein Pfadelement $pe \in PE_k$ die Bedingung $pe.m_1 = m_0$ erfüllen kann. Somit modelliert diese zweite Summe, ab wann ein Frame von k auf m sendebereit wird, was hier aus dem Sendezeitpunkt und der Verzögerungszeit auf dem Vorgängerlink berechnet wird. Wie in Kapitel 7.2 erläutert, muss dabei der Fall unerwartet kleiner Frames abgedeckt werden, sodass die Verzögerungszeit eines Minimalframes angenommen wird. Basierend auf diesen Definitionen kann die Verifikation der Queuezuweisung auf Link m durch Algorithmus 9.3 erfolgen.

Der Algorithmus iteriert über alle vorhandenen Queues von m und extrahiert zu Beginn alle Linknutzungen mit dieser Queue aus LU_m. Wie schon in Algorithmus 9.2 für die Ressourcenbedingungen werden anschließend die Linknutzungen nach der Startzeit der Zeitschlitze sortiert und über Paare aufeinanderfolgender Linknutzungen iteriert. Für jedes Paar wird dann die Frameisolation überprüft. Dabei ist das Aufrunden $\lceil lu^{first}.t_1 \rceil$ vom Ende eines vorangegangenen Zeitschlitzes erforderlich, da dies den Zeitpunkt des erstmöglichen Gate-Close-Events darstellt.

9.3.3 Integralität und Zeitschlitzlänge

Abschließend sollte für jeden Stream k überprüft werden, ob die in PE_k definierten Startzeiten der Zeitschlitze die Ganzzahlbedingung erfüllen und ob die Zeitschlitze die korrekte Länge besitzen. Dies kann nach Algorithmus 9.4 erfolgen.

```
1  foreach pe ∈ PE_k do                                           // iterate all path elements
2      if |⌊pe.t_0 + 0.5⌋ − pe.t_0| > precision then
3          report integrality violation
4      m := pe.m
5      if pe.t_1 − pe.t_0 < sl_km − precision then
6          report slot length violation
```

Algorithmus 9.4: Verifikation von Ganzzahlbedingung und Zeitschlitzlänge für Stream k.

Kapitel 10

Vergleich zu Schedulern aus der Literatur

Zu Beginn der eigenen Arbeiten an TT-RSP standen nur wenige vergleichbare Arbeiten [27, 34, 37, 9] zur Verfügung. Da diese außerdem durch grundlegende Modellvereinfachungen oder alleinige Betrachtung des Schedulings (ohne Routing) nicht der eigenen Zielsetzung einer möglichst umfassenden Modellierung von TT-RSP entsprachen, wurden die in dieser Arbeit vorgestellten Modelle nur unwesentlich durch andere Entwicklungen aus dem Bereich beeinflusst. Die Zusammenstellung in [4] zeigt allerdings die massive Zunahme an Forschungsarbeiten mit einer Vielzahl von Lösungsansätzen seit Beginn der eigenen Arbeiten.

Zur besseren Einordnung und Abgrenzung dieser Arbeiten wurden in [E15, 4] zeitgleich wichtige *qualitative* Unterscheidungsmerkmale ausgearbeitet. Diese umfassen die eingesetzten mathematischen Methoden sowie die berücksichtigten Technologie- und Streameigenschaften der veröffentlichten Lösungsverfahren für TT-RSP. Basierend auf diesen Arbeiten wird zu Beginn dieses Kapitels ein Überblick über die Unterscheidungsmerkmale gegeben und anhand dieser ein umfassender Vergleich der eigenen Lösungsverfahren mit denen aus der Literatur vorgenommen.

Die Herausforderung beim *quantitativen* Performancevergleich ist dagegen, dass nur selten Implementierungen (z.B. TSNSched [53]) begleitend zu publizierten Lösungsverfahren öffentlich zugänglich gemacht werden und Nachimplementierungen im Allgemeinen nicht trivial sind. Diesbezüglich besitzen modellbasierte Verfahren (SMT/ILP) allerdings den Vorteil, dass die formale Beschreibung besser zur Nachimplementierung geeignet ist, als der oftmals unpräzise, unvollständige Pseudocode veröffentlichter Heuristiken. Darüber hinaus sind die Modelle dieser Arbeit insbesondere hinsichtlich der Lösungsraumabdeckung umfassender als die Modelle aus der Literatur. Diese beiden Eigenschaften ermöglichen es, in diesem Kapitel eine Performanceabschätzung im Vergleich zu Verfahren aus der Literatur vorzunehmen, indem die eigenen ILP-Modelle auf vereinfachte Modelle aus der Literatur reduziert werden.

10.1 Qualitativer Vergleich: Methoden, Features und Einschränkungen

Der qualitative Vergleich fokussiert sich zu Beginn auf eine präzise Erläuterung der wesentlichen Unterscheidungsmerkmale von Schedulern. Zusätzlich werden dabei Symbole und Abkürzungen

eingeführt, welche anschließend in einer umfassenden Übersichtstabelle für Scheduler Anwendung finden. Anhand dieser Übersichtstabelle wird dann eine Einordnung der eigenen Arbeiten vorgenommen. Ähnliche Zusammenstellungen wurden in [E15, 4, 45] veröffentlicht. Im Vergleich zu diesen Arbeiten werden hier jedoch zahlreiche Details ergänzt, während zugleich eine möglichst kompakte Darstellung in Form der abschließenden Übersichtstabelle angestrebt wird.

10.1.1 Unterscheidungsmerkmale von Schedulern

Die Merkmale von Schedulern lassen sich unterteilen in Unterschiede bezüglich der eingesetzten mathematischen Methoden, unterstützten Streamfeatures und Einschränkungen bezüglich der erzeugten Schedules. Diese werden hier ohne Nennung von Beispielen aus der Literatur eingeführt, da diese der später folgenden Tabelle entnommen werden können.

10.1.1.1 Mathematische Methoden

Eine Übersicht über die wichtigsten Merkmale von Schedulern in Bezug auf deren eingesetzte mathematische Methoden ist in Abbildung 10.1 gegeben.

Der erste Aspekt ist der diskutierte *Problemumfang*, der sich aus *Routing*, *Scheduling* und *Queuezuweisung* zusammensetzt. Das in Kapitel 3 definierte TT-RSP umfasst das Routing und Scheduling von Streams innerhalb eines zeitgesteuerten Netzwerkes, und viele der veröffentlichten Arbeiten entwerfen ebenfalls kombinierte Lösungsverfahren für genau diese beiden Teilprobleme. Abweichend hiervon werden im Rahmen des folgenden Literaturvergleichs jedoch auch Ansätze berücksichtigt, die ausschließlich das Scheduling auf Basis eines zuvor festgelegten Routings betrachten, ohne die gegenseitige Abhängigkeit zu analysieren. In einigen Arbeiten werden darüber hinaus Queuezuweisungen vorgenommen, wobei diese in der Regel wie in Kapitel 7 durch die konkreten Restriktionen des TSN-Standards motiviert sind.

Sofern mehrere dieser drei Teilprobleme behandelt werden, ist die Art der *Kopplung* dieser Teilprobleme ein bedeutendes Unterscheidungsmerkmal. Bei den in dieser Arbeit entworfenen ILP-Modellen wird die gegenseitige Abhängigkeit beispielsweise durch ein *gemeinsames mathematisches Modell* bezüglich eines exakten Lösungsverfahrens (ILP) abgebildet [JM]. Dies hat zur Folge, dass die Modellierung einzelner Teilprobleme (z.B. Scheduling) komplexer wird, da diese in generalisierter Form für jede mögliche Lösung eines anderen Teilproblems (z.B. Routing) ausgedrückt werden muss. Berücksichtigt ein Lösungsansatz die gegenseitige Abhängigkeit der Teilprobleme und reagiert adaptiv (z.B. nachträgliche Anpassung zuvor geplanter Routen nach Misserfolg im Scheduling), ohne jedoch ein gemeinsames mathematisches Modell zu formulieren, so wird dies hier als *heuristische Kopplung* bezeichnet [HC]. In einigen Arbeiten werden auch mehrere Teilprobleme betrachtet, wobei die Planung streng *sequentiell* erfolgt (z.B. Routing $\longrightarrow$ Queuezuweisung $\longrightarrow$ Scheduling), ohne zuvor getroffene Entscheidungen erneut anzupassen [SP].

Die eingesetzten *Algorithmen* bzw. *Solver* lassen sich wiederum unterteilen in die genutzten *exakten* und *heuristischen* Verfahren. Seitens der exakten Verfahren sind SMT/OMT und ILP/MIP die populärsten Vertreter, während sich heuristische Verfahren in bekannte *Meta-Heuristiken* und von den

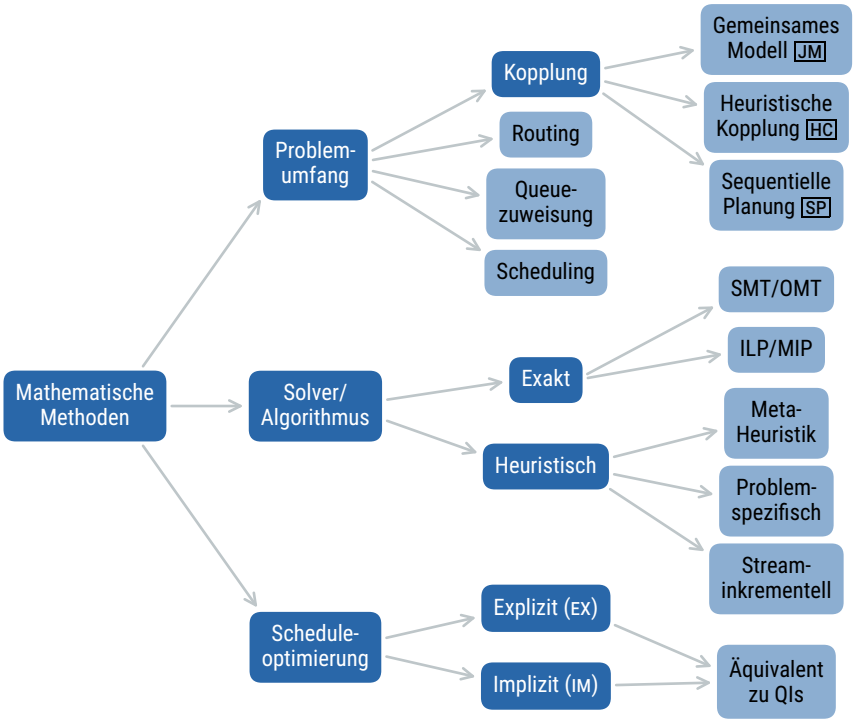


Abbildung 10.1: Unterscheidungsmerkmale bzgl. der mathematische Methoden von Schedulern.

jeweiligen Autoren entworfene *problemspezifische* Heuristiken unterteilen lassen. Gesondert erwähnt wird hier außerdem die (heuristische) *Stream-inkrementelle* Planung, da diese sowohl mit als auch ohne *Backtracking* (BTR) zu den am häufigsten anzutreffenden Strategien in der hier untersuchten Literatur zählt. Dabei werden die Streams einzeln oder in Gruppen iterativ geplant, wobei in jeder Iteration der Schedule aller Streams aus vorherigen Iterationen als fest vorgegeben betrachtet wird und die zu planenden Streams konfliktfrei integriert werden müssen. Wird Backtracking unterstützt, so können im Falle einer Iteration mit unlösbaren Konflikten einige der zuvor geplanten Streams erneut geplant werden, um den Konflikt aufzulösen. In der Regel lassen sich Lösungsverfahren für TT-RSP nicht isoliert einer der genannten Methoden zuordnen. So werden auch exakte Formulierungen nahezu ausnahmslos mit Heuristiken kombiniert, wie beispielsweise durch Routenvorverarbeitung (auch *rpp*, von *route preprocessing*) oder Stream-inkrementelles Lösen des Modells.

Gefundene Schedules sollen in der Regel besondere Gütekriterien erfüllen, welche durch konkrete Qualitätsindikatoren (QIs) erfasst werden (vgl. Kapitel 4.1.1). In der Literatur vorgeschlagene Lösungsverfahren für TT-RSP unterscheiden sich dementsprechend auch bezüglich der *Scheduleoptimierung*. Beim Literaturvergleich wird diese zusätzlich in *explizite* (EX) und *implizite* (IM) Optimierung unterteilt. Das Optimierungsvorgehen wird dabei als explizit eingestuft, wenn für das jeweilige Lösungsverfahren eine Zielfunktion zur Optimierung definiert wird, und das Verfahren außerdem in der Lage ist über das Finden einer initialen Lösung hinaus nach Lösungen mit verbesserter Lösungsgüte zu suchen. Ergeben sich bestimmte Qualitätsmerkmale gefundener Lösungen allein aus der Suchstrategie (z.B. Bevorzugung kurzer Pfade oder geringen Bufferings) ohne Optimierungsvorgang nach dem Finden der initialen Lösung, so wird dies hier als implizit bezeichnet. Die in der

Literatur anzutreffenden Optimierungsziele sind weitgehend äquivalent zu den auch zur Bewertung der Lösungsgüte eingesetzten QIs (Streamlatenzen, Makespan, Lastverteilung, ...), welche bereits in Kapitel 4.1.1 vorgestellt wurden. Auch wenn die mathematische Definition von Gütefunktionen in Bezug auf gefundene Schedules und Zielfunktionen im Rahmen des Optimierungsvorgangs sich im Detail unterscheiden kann, soll dies hier der Einfachheit halber nicht getrennt betrachtet werden.

10.1.1.2 Unterstützte Streameigenschaften

Weitere Unterscheidungsmerkmale zwischen Schedulern finden sich bezüglich der *Streameigenschaften*, die der Scheduler verarbeiten kann. In der Literatur zeigen sich insbesondere Unterschiede hinsichtlich der Unterstützung von verschiedenen *Verkehrsklassen*, *Multicast* und *heterogenen Streamparametern*. Eine fehlende Unterstützung des jeweiligen Features kann bedeuten, dass bestimmte Eingangsdaten überhaupt nicht verarbeitet werden können. Darüber hinaus gibt es in einigen Fällen auch Unterschiede wie weitreichend das jeweilige Feature in den Planungsprozess integriert ist. Eine Übersicht der hier genutzten Abstufungen ist in Abbildung 10.2 gezeigt.

Während sich die meisten der hier untersuchten Arbeiten ebenso wie die hier vorliegende Arbeit ausschließlich mit dem *zeitgesteuerten Verkehr* (TT, von *Time-Triggered*) befassen, berücksichtigen einige Arbeiten bei der Planung auch die QoS-Eigenschaften niedriger priorisierter Verkehrsklassen wie *Audio Video Broadcast* (AVB) und *Best Effort* (BE). Die *Integration* der nicht zeitgesteuerten Verkehrsklassen (AVB, BE) kann wiederum unterschieden werden in die *gemeinsame Planung*, bei der auch die Streams dieser Verkehrsklassen bei der Planung bereits bekannt sind dementsprechend zusammen mit TT-Streams geplant werden [JP]. Wird dagegen bei der Planung von TT-Streams nur auf eine angemessene Lastverteilung und/oder Queuezuweisung gesetzt, um die verbleibenden Ressourcen für niedriger priorisierten Verkehr zu verbessern, so wird dies hier *proaktiv* bezeichnet [PA]. Während sich die meisten Arbeiten im Falle des TT-Verkehrs damit befassen, dass ein Stream maximal einen Frame je Sendeintervall verschickt, wird in einigen Arbeiten auch der Fall von *mehreren Frames* (MFTT, von *multi-frame Time-Triggered*) behandelt. Die Scheduler-Klassifizierung des folgenden Kapitels betrachtet dabei nur solche Ansätze als MFTT, die den verschiedenen Frames eines Streams auch dedizierte Sendezeitpunkte zuweisen, wohingegen die triviale Modellierung als ein einziger, besonders langer Zeitschlitz weiterhin als TT aufgefasst wird.

Bezüglich *Multicast* (d.h. Streams mit mehr als einem Empfänger) ist nicht nur zu unterscheiden, ob dies überhaupt unterstützt wird, sondern auch, wie komplex die entsprechende Modellierung ausfällt. Wie in Kapitel 6.5 ausführlich diskutiert, kann die Integration von Multicast bei einem kombinierten Routing und Scheduling ([JM]) umfangreiche Änderungen gegenüber einem reinen Unicast-Modell erfordern, da Routingbedingungen mit Multicast-Unterstützung erforderlich sind und zusätzlich Auswirkungen auf das Scheduling abgebildet werden müssen. Ein derartiges Vorgehen wird hier als *vollständige* Modellierung von Multicast eingestuft und später mit ✓ gekennzeichnet. Bei separatem Routing und Scheduling und den meisten Lösungsverfahren mit heuristischer Kopplung ist die Unterstützung von Multicast innerhalb des Schedulings dagegen trivial, da im Scheduling-Schritt bereits ein festes Routing vorliegt. Die Multicast-Unterstützung in solchen Modellen wird hier daher als *vereinfachte* Modellierung aufgefasst und mit $\checkmark^{s}$ markiert.

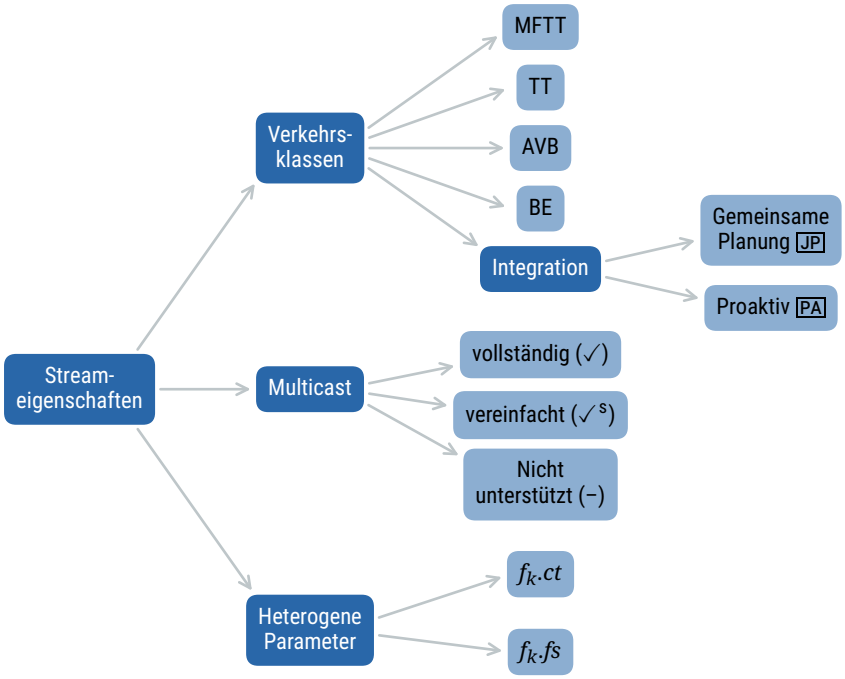


Abbildung 10.2: Unterstützte Streameigenschaften von Schedulern.

Die Unterstützung *heterogener Parameter* (innerhalb eines Verkehrsmusters) bezieht sich darauf, dass Streams mit unterschiedlichen Zykluszeiten $f_k.ct$ oder Framegrößen $f_k.fs$ explizit als solche modelliert und geplant werden. Für $f_k.ct$ kann dies beispielsweise durch Modellierung der Hyperperiode der einzelnen Stream-Zykluszeiten und Sicherstellung der Konfliktfreiheit innerhalb dieser erfolgen. Es sei angemerkt, dass ein vereinfachter Ansatz, der alle Streams mit einer einzigen Basis-Zykluszeit BC plant, grundsätzlich auch heterogene Zykluszeiten im Verkehrsmuster zulassen kann, sofern alle Zykluszeiten ganzzahlige Vielfache von BC sind.[1] In der Praxis würde das bedeuten, dass ein Stream mit $f_k.ct := 2 \cdot BC$ nur jeden zweiten der für ihn vorgesehenen Zeitschlitze tatsächlich nutzt. Aufgrund der offensichtlichen Nachteile bei der Effizienz wird dieses triviale Vorgehen hier *nicht* als Unterstützung heterogener Parameter aufgefasst. Eine ähnliche Argumentation ist bezüglich $f_k.fs$ möglich, wo ein einfaches Modell beispielsweise grundsätzlich MTU-Frames planen könnte, auch wenn tatsächlich kleinere Frames erwartet werden. Dementsprechend wird im Schedulervergleich nur von Unterstützung heterogener Framegrößen gesprochen, wenn die im Verkehrsmuster angegebene Framegröße bei der Planung von Zeitschlitzen tatsächlich berücksichtigt wird.

10.1.1.3 Schedule-Einschränkungen

Während die grundsätzlichen Bedingungen für einen gültigen Schedule bereits in der Problemstellung (vgl. Kapitel 3.2) definiert wurden, unterliegen die tatsächlich erstellten Schedules verschiedener Scheduler aus der Literatur in der Regel weiteren Einschränkungen. Diese stammen vorwiegend von

[1] Der Basiszyklus eines Verkehrsmusters entspricht demnach $BC := \gcd^{\{\}}(\{f_k.ct \mid k \in F\})$, dem größten gemeinsamen Teiler aller Stream-Zykluszeiten.

technologischen Besonderheiten (z.B. FIFO-Queuing in TSN) und von Annahmen, die das Lösungsverfahren vereinfachen sollen. Bei solchen Schedule-Einschränkungen handelt es sich um zusätzliche Begrenzungen des durchsuchten Lösungsraumes, sodass derartige Einschränkungen entscheidend dafür sind, ob ein Scheduler bestimmte Lösungen überhaupt finden kann. Die im Literaturvergleich berücksichtigten Einschränkungen sind in Abbildung 10.3 dargestellt.

Buffering Beim *Buffering* innerhalb von Bridges gehen einige Lösungsverfahren (z.B. die ILP-Modelle dieser Arbeit ohne Modelloptionen *iq*/*bq*) davon aus, dass Frames bei der Zwischenspeicherung in der Bridge in Bezug auf deren Empfangs- und Sendereihenfolge beliebig umsortiert werden können (Reordering). In Anlehnung an Random Access Memory wird dies hier als *Random Access Buffer* (RAB) bezeichnet.[2] Im Vergleich dazu hat die in TSN-Bridges vorgesehene Zwischenspeicherung in maximal acht FIFO-Queues je Ausgangsport weniger Reordering-Möglichkeiten und schränkt somit den Lösungsraum bezüglich des Schedulings ein. Da beim Scheduling für ein TSN in der Regel nicht alle acht Queues genutzt werden sollen, wird dieses Modell in der Literatur in der Regel mit flexibler Queueanzahl modelliert und hier daher als n-FIFO bezeichnet. Entsprechende Lösungsverfahren müssen neben der FIFO-Bedingung für jede einzelne Queue auch die Queuezuweisung von Streams abbilden. Als Spezialfall 1-FIFO kann auf die Queuezuweisung verzichtet werden, womit sich die Modellierung vereinfacht, aber auch der Lösungsraum weiter beschränkt wird (keinerlei Reordering). Ein in der Literatur häufig anzutreffendes Buffering-Modell ist außerdem *No-Wait* (NW), bei dem gefordert wird, dass ein Frame immer sofort nach Verarbeitung durch die Bridge auf den zugehörigen Ausgangsports weitergeleitet werden kann. Es dürfen also keine Wartezeiten entstehen, weil zuerst noch andere Frames übertragen werden müssen (Queueing). Dabei wird in der Literatur in der Regel davon ausgegangen, dass der Ausschluss von Queueing gleichzeitig bedeutet, dass nur eine Queue für den zeitgesteuerten Verkehr benötigt wird (da ohne Queueing auch kein Reordering möglich ist) und somit keine dedizierten Queuezuweisungen vorgenommen werden müssen. Somit handelt es sich um einen bezüglich des Lösungsraumes noch stärker eingeschränkten Spezialfall von 1-FIFO, welcher hier daher auch als 1-NW referenziert wird. Dabei wird in keiner der in diesem Kapitel analysierten Arbeiten diskutiert, dass 1-NW weiterhin anfällig gegen Fehlerfälle durch unerwartet kleine Frames ist (vgl. Kapitel 7.2). Tatsächlich kann es daher zwecks Frameisolation sinnvoll sein, NW mit Queuezuweisungen zu kombinieren, was hier als n-NW bezeichnet wird. Es sei angemerkt, dass sich bei der Kombination von NW mit festen Routen die Problemmodellierung weiter vereinfacht, da nur noch der Sendezeitpunkt am Sender (*Inject Time*) entschieden werden muss, während sich alle weiteren Zeitschlitze entlang einer Route durch die NW-Bedingung und statische Bridge-/Linkverzögerungen ergeben. Eine derartige Modellierung wird im Folgenden als Superskript der Buffering-Strategie mit $<>^{\text{INJ}}$ gekennzeichnet. Bei kombinierten Modellen (JM) ist dagegen weiterhin eine Planung der linkspezifischen Sendezeitpunkte notwendig (*Rescheduling*), da diese an die aktuell gewählte Route angepasst werden müssen. Diese variable Modellierung aller linkspezifischen Sendezeitpunkte wird mit $<>^{\text{RS}}$ markiert, und zusätzlich mit $<>^{\text{IA}}$, sofern diese Sendezeitpunkte auf allen Links ganzzahlig modelliert werden.

[2] Aus der Literatur ist diesbezüglich kein gebräuchlicher Begriff bekannt.

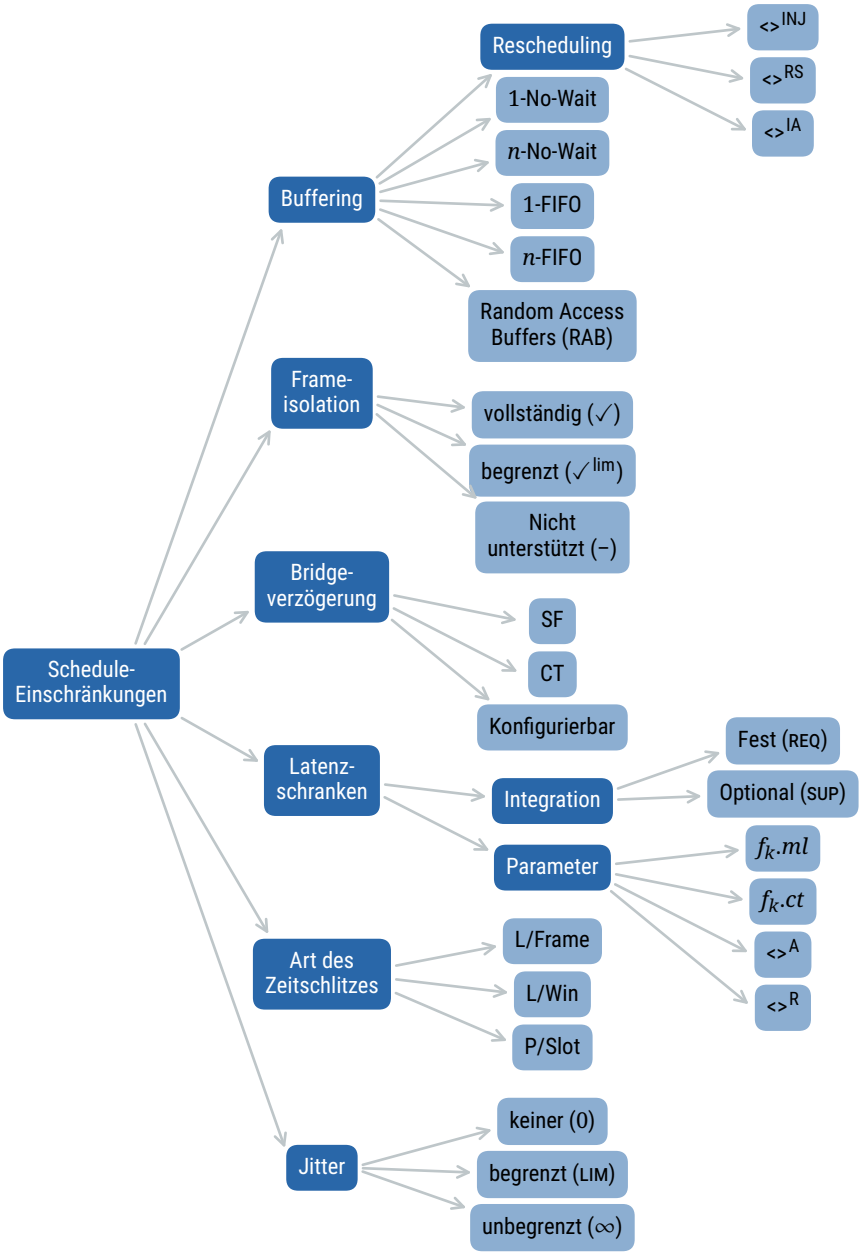


Abbildung 10.3: Einschränkungen in Bezug auf erstellte Schedules verschiedener Scheduler.

Frameisolation Die *Frameisolation* behandelt die in Kapitel 7.2 diskutierte Vermeidung von Fehlerfällen beim Queuing von Frames unterschiedlicher Streams. Sie erfordert in der Regel konkrete Zusatzbedingungen (vergleichbar mit *iq*/*bq*), welche sämtliche Fehlerfälle durch fehlende oder unerwartet kleine Frame ausschließen. Enthält ein Lösungsverfahren solche Zusatzbedingungen so wird dies hier als *vollständige* Frameisolation bezeichnet und später im Schedulervergleich mit ✓ markiert. Ansätze mit NW-Buffering weisen bereits standardmäßig eine begrenzte Frameisolation auf, da Frames beispielsweise nicht durch das Fehlen eines vorangegangenen Frames zu früh gesendet werden können (da kein Queueing stattfindet). Eine Isolation im Falle unerwartet kleiner

Frames liegt jedoch nicht automatisch vor. Diese *begrenzte* Isolation der NW-Modelle wird hier durch $\checkmark^{\text{lim}}$ gekennzeichnet. Es sei angemerkt, dass das Weglassen von expliziter Frameisolation den Lösungsraum vergrößert und die Modellierung vereinfachen kann, allerdings in der Praxis aufgrund der verringerten Zuverlässigkeit problematisch sein kann.

Bridgeverzögerung Die *Bridgeverzögerung* ergibt sich in der Literatur in vielen Fällen nach dem *Store-and-Forward*-Prinzip (SF), welches von den entsprechenden Schedulern im Pfadscheduling fest vorgegeben wird. Einige Scheduler unterstützen jedoch auch optional das Umschalten auf geringere Bridgeverzögerungen nach dem *Cut-Through*-Prinzip (CT). Darüber hinaus finden sich auch spezieller konfigurierbare Scheduler, bei denen sich die Bridgeverzögerungen als Parameter übergeben lässt oder aber aus Topologieinformationen extrahiert wird und somit einzeln für jede Bridge definiert werden kann. Auch wenn die Bridgeverzögerung grundsätzlich für den praktischen Einsatz relevant ist und zudem Einfluss auf den Lösungsraum hat, handelt es sich hierbei um eine vergleichsweise unbedeutende Schedule-Einschränkung, da sie in den Schedulingbedingungen durch Konstanten abgebildet wird. Somit kann sie im Gegensatz zu vielen anderen hier betrachteten Aspekten in der Regel ohne weitreichenden Einfluss auf das Lösungsverfahren ausgetauscht werden.

Latenzschranken Bezüglich der *Latenzschranken* von Streams kommen in der Literatur häufig abweichende Annahmen (im Vergleich zur Problemstellung nach Kapitel 3.1) zum Einsatz. Als *Parameter* für modellierte Latenzschranken kommt beispielsweise neben einer expliziten Angabe der Latenzschranke als Streamparameter $f_k.ml$ in vielen Fällen stattdessen oder zusätzlich die Zykluszeit $f_k.ct$ zum Einsatz. Werden diese Angaben als relative Latenzschranken aufgefasst, so muss ein Frame des Streams k beispielsweise spätestens um $f_k.ml$ bzw. $f_k.ct$ nach dem Absenden am Sender vollständig bei allen Empfängern angekommen sein. Solche relativen Latenzschranken werden hier im Literaturvergleich mit <>$^{\text{R}}$ angezeigt, wie z.B. $f_k.ml^{\text{R}}$. Beide Parameter werden jedoch in einigen Publikationen auch als absolute Latenzschranke aufgefasst, was hier durch <>$^{\text{A}}$ gekennzeichnet wird. Für diese gilt der späteste Empfangszeitpunkt an den Empfängern unabhängig vom Sendezeitpunkt. Der Referenzpunkt ist stattdessen in der Regel der Zeitpunkt 0 bzw. der Beginn eines Sendeintervalls. Von besonderer Bedeutung ist außerdem die *Integration* dieser Latenzschranken in das Lösungsverfahren. Einige Ansätze stellen die Bedingung ohne weitere Verzahnung mit dem restlichen Lösungsverfahren auf (z.B. die Modelle dieser Arbeit), sodass diese Latenzschranken hier als *optional* (SUP) eingestuft werden. In anderen Fällen basiert dagegen die Modellierung auf *festen* (REQ) Annahmen hinsichtlich der Latenzschranken, sodass Änderungen nicht ohne Anpassung des Lösungsverfahrens möglich sind. Als Beispiel sei hier eine feste Latenzschranke „REQ: $f_k.ct^{\text{A}}$“ genannt, aufgrund der sich die Modellierung von Sendezeitpunkten eines Streams k vereinfacht, da diese auf den Wertebereich $[0, f_k.ct - 1]$ beschränkt werden. Somit vereinfacht sich auch die Modellierung von Ressourcenbedingungen (vgl. Kapitel 6.4.2), da für diese keine modulo-Operation bezüglich der Sendezeitpunkte mehr abgebildet werden muss. Demnach können Lösungsverfahren, die diese Einschränkung ausnutzen, in der Regel nicht ohne weitere Modifikationen auf einen größeren Sendezeitraum ausgeweitet werden. Andererseits verhindert diese Vorgabe die zyklusübergreifende Planung und hat somit bedeutende Einschränkungen des Lösungsraums zur Folge, was in Abbildung 10.4 durch einen Vergleich von Schedules mit und ohne diese Einschränkung gezeigt ist.

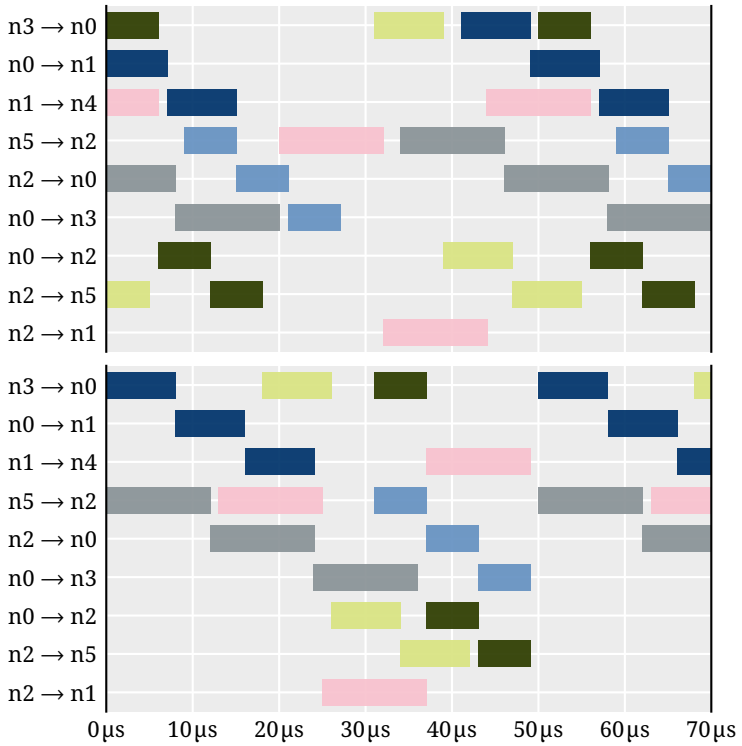


Abbildung 10.4: Zwei mögliche Schedules von 6 Streams mit einer Zykluszeit von 50 µs in einer Ringtopologie (3 Bridges, 3 Endgeräte). Der erste Schedule nutzt zyklusübergreifende Planung, während diese im zweiten vermieden wurde, so wie dies für Scheduler mit Latenzschranke „REQ: $f_k.ct^{A}$“ der Fall ist. Dementsprechend überlappt kein Stream den Zeitpunkt 50 µs.

Art des Zeitschlitzes Die in dieser Arbeit vorgestellten Modelle sowie die Mehrheit der Lösungsverfahren aus der Literatur realisieren den zeitgesteuerten Verkehr, indem jedem einzelnen Frame auf jedem genutzten Link ein spezifischer Zeitschlitz zugewiesen wird, welcher der benötigten Sendedauer des jeweiligen Frames (abhängig von Framegröße und Linkbandbreite) entspricht. Dieses Prinzip der link- und framespezifischen Zeitschlitze wird hier mit L/Frame bezeichnet. Als Alternative existiert der *Window*-basierte Ansatz, in dem auf jedem Link zunächst eine vordefinierte Anzahl von Sendefenstern vorgesehen werden, ohne dass diese vorab einem konkreten Stream zugeordnet sind. Ein Scheduler ermittelt in diesem Fall die Zuordnung von Streams zu Sendefenstern und passt Position und Länge der Sendefenster entsprechend an. Dabei können die Frames unterschiedlicher Streams demselben, ausreichend langen Sendefenster zugewiesen werden, ohne dass die Sendereihenfolge dieser Frames innerhalb des Sendefensters festgelegt wird. Der Window-basierte Ansatz (L/Win) kann zu höheren Streamlatenzen und Jitter führen, kann im Gegenzug jedoch mit weniger Guard Bands und GCL-Einträgen realisiert werden. Eine weitere Alternative besteht in einem vereinfachten Modell, in dem der komplette Pfad vom Sender zu den Empfängern eines Streams so lange reserviert wird, wie die gesamte Übertragung über alle Hops dauert. Es werden bei diesem Ansatz (P/Slot) also wesentlich längere Zeitschlitze auf jedem Link reserviert, als tatsächlich notwendig wären. Eine gleichzeitige Übertragung auf vollständig disjunkten Pfaden ist bei diesem Ansatz weiterhin möglich.

Jitter Des Weiteren unterscheidet sich zwischen Schedulern, inwiefern die erstellten Schedules *Jitter* zulassen. Jitter kann bei zeitgesteuertem Verkehr dann auftreten, wenn Verkehrsmuster mit heterogenen Zykluszeiten geplant werden und einzelnen Streams bei mehreren Sendeintervallen innerhalb der Hyperperiode jeweils unterschiedliche Zeitschlitze zugewiesen werden, sodass sich die für die Anwendung sichtbare Ende-zu-Ende-Latenz von Zyklus zu Zyklus unterscheidet. Das Zulassen von Jitter geht auf diese Weise mit einer komplexeren Problemformulierung einher (Modellierung mehrerer Sendezeitpunkt pro Stream), während zugleich die Anzahl der Lösungsmöglichkeiten steigt. Des Weiteren tritt Jitter bei einem Window-basierten Ansatz (L/Win) auf.

Abschließend sei angemerkt, dass die Tradeoffs der zahlreichen Schedule-Einschränkungen und den hier gezeigten Abstufungen in der Literatur bisher nicht umfassend untersucht sind. Die konkreten Auswirkungen auf grundlegende Perfomanceindikatoren (Rechenzeit, Schedulability) und Qualitätsindikatoren bleiben also in der Regel unklar. Diese Thematik wird in Kapitel 10.2 aufgegriffen, in dem Performancevergleiche bezüglich der zyklusübergreifenden Planung und No-Wait-Buffering vorgenommen werden.

10.1.2 Klassifizierung von Schedulern

Die bisher eingeführten Unterscheidungsmerkmale werden in den Tabellen 10.1–10.3 zum Vergleich von einer Vielzahl von Schedulern aus der Literatur genutzt. Dabei werden für einen umfassenderen Gesamteindruck Lösungsverfahren verschiedenster Methoden (Heuristiken, SMT, ILP) betrachtet, wobei aufgrund der großen Anzahl der vorhandenen verwandten Arbeiten (vgl. [4]) nicht alle hier aufgeführt werden können. Für den direkten Vergleich mit den in dieser Arbeit vorgestellten ILP-Modellen wird jedoch insbesondere für ILP-basierte Verfahren mit gemeinsamer Modellierung von Routing und Scheduling (JM) eine Abdeckung aller bekannten Arbeiten angestrebt.

In der Tabelle wurden stets nur die in den jeweiligen Publikationen explizit ausgeführten Features berücksichtigt. Aussagen, dass bestimmte Erweiterungen möglich seien, wurden also nicht als Unterstützung des entsprechenden Features gewertet. Es sei jedoch angemerkt, dass einige Features sich tatsächlich vergleichsweise einfach verändern bzw. ergänzen lassen. Dies trifft beispielsweise auf verschiedene Weiterleitungsverzögerungen (SF, CT) sowie auf Multicast in den meisten Verfahren mit separaten Schritten für Routing und Scheduling (d.h. SP oder HC) zu. Die Unterstützung heterogener Zykluszeiten kann ebenfalls in vielen Fällen durch Ausweitung der Ressourcenbedingungen auf alle Zeitschlitzwiederholungen innerhalb der Hyperperiode ergänzt werden.

Die Publikationen sind nach Veröffentlichungsjahr sortiert, wobei eigene Arbeiten immer als letzte Arbeit des jeweiligen Jahres aufgeführt sind. Somit können zur Einordnung der eigenen Arbeiten einfach alle in der Tabelle jeweils weiter oben aufgelisteten Arbeiten herangezogen werden. Dieser Literaturvergleich wird untergliedert in die Arbeiten bis 2017 zur Einordnung des in [E12] veröffentlichten Unicastmodells *UC*, die Arbeiten bis 2020 zur Einordnung der in [E14] präsentierten Multicastmodelle, und Arbeiten bis 2024 zur Einordnung der im Rahmen dieser Arbeit vorgenommenen Modellerweiterungen (Tabelleneintrag *TW*).

	Mathematical Approach			Traffic Features			Schedule Limitations					
	Problem Scope	*Solver/ Algorithm*	*Schedule Optimization*	*Traffic Classes*	*Multi-cast*	*Heterog. Param.*	*Buffering Model*	*Frame Isolation*	*Bridge Delay*	*Latency Bounds*	*Time Slot Type*	*Jitter*
[8]	Routing HC Scheduling	Initial Solution (SP/ASAP) + Tabu Search	EX: sab^{st} EX: lt (RC)	MFTT$^{?}$ JP RC	✓	$f_k.ct, f_k.fs$	$\text{RAB}^{\text{RS?}}$ (TTEthernet)	–	$\text{SF}^{?}$	SUP: $f_k.ml^{\text{R?}}$	L/Frame	0
[27]	Queue Assignm. JM Scheduling	OMT (Stream-Inc. w/ BTR) (Non-Inc. supported)	EX: |ttq|	MFTT	–	$f_k.ct, f_k.fs$	n-$\text{FIFO}^{\text{RSIA}}$	✓ (2 variants)	SF	REQ: $f_k.ct^{\text{A}}$ SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[34]	Queue Assignm. JM Scheduling	ILP	EX: |ttq|	MFTT PA AVB	–	$f_k.ct, f_k.fs$	n-$\text{FIFO}^{\text{RSIA}}$	✓	SF	REQ: $f_k.ct^{\text{A}}$ SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[9]	Routing JM Scheduling	ILP (w/ 3 different rpp: S/UR, S/PR, S/FR)	EX: sab^{st} IM: lt^{R}	TT (max. 8 hops)	–	–	1-NW^{iNJ}	✓ (implied)	flexible	REQ: $f_k.ct^{\text{A}}$ REQ: $\frac{f_k.ct_{\text{R}}}{sc}$	P/Slot	near-0 (small jitter due to BE frames)
[37]-1	Scheduling	ILP	EX: span IM: lt^{R}	TT	–	$f_k.fs$	1-NW^{iNJ}	✓$^{\text{lim}}$	SF	REQ: $f_k.ct^{\text{A}}$	L/Frame	0
[37]-2	Scheduling	Greedy Scheduling (Stream-Inc. w/o BTR, Tabu Search on Scheduling Order)	EX: span IM: lt^{R}	TT	–	$f_k.fs$	1-NW^{iNJ}	✓$^{\text{lim}}$	SF	REQ: $f_k.ct^{\text{A}}$	L/Frame	0
[10]	Routing JM Scheduling	ILP (PB/SAT) w/ rpp	EX: |gb|, intf IM: lt^{R}	TT PA other	✓	$f_k.ct, f_k.fs$	1-NW^{RSIA}	✓$^{\text{lim}}$	SF	$-^{?}$	L/Frame	0
[E12]	Routing JM Scheduling	ILP w/ rpp	EX: lt^{R}	TT	–	$f_k.ct, f_k.fs$	RAB^{RSIA}	–	config. const	SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[36]	Routing JM Scheduling	Heur. ILP (w/ rpp) (Stream-Inc. w/o BTR)	EX: lb,pl IM: lt^{R}	TT (max. 7 hops)	✓	$f_k.ct$	1-NW^{iNJ}	✓ (implied)	flexible	REQ: $f_k.ct^{\text{A}}$ REQ: $\frac{f_k.ct_{\text{R}}}{sc}$	P/Slot	near-0 (small jitter due to BE frames)
[38]	Routing JM Scheduling	ILP	IM: lt^{R}	TT	–	$f_k.ct, f_k.fs$	1-NW^{RSIA}	✓$^{\text{lim}}$	config. const	SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[55]	Routing HC Queue Assignm. HC Scheduling	Routing: k Shortest Path Scheduling: GRASP	EX: lt^{R}, |ttq| EX: lt (AVB)	MFTT JP AVB	–	$f_k.ct, f_k.fs$	n-$\text{FIFO}^{\text{RSIA}}$	✓	SF	SUP: $f_k.ml^{\text{R}}$	L/Frame	$0^{?}$

Tabelle 10.1: Schedulervergleich (Teil 1/3).

	Problem Scope	**Mathematical Approach** *Solver/ Algorithm*	*Schedule Optimization*	**Traffic Features** *Traffic Classes*	*Multi-cast*	*Heterog. Param.*	*Buffering Model*	*Frame Isolation*	*Bridge Delay*	**Schedule Limitations** *Latency Bounds*	*Time Slot Type*	*Jitter*
[21]	Routing [SP] Scheduling	ILP (and additonal heur. for repairs after link failure)	EX: *poros*	TT	$\checkmark^{s}$	$f_k.ct, f_k.fs$	RAB$^{\text{RSIA}}$	–	SF	REQ: $f_k.ct^{\text{A}}$ SUP: $f_k.ml^{\text{R}}$ SUP: $f_k.ml^{\text{A}}$	L/Frame	0
[33]	Queue Assignm. [JM] Scheduling	OMT (quantifier-free integer-indexed arrays w/ integer elements)	EX: jit	TT	–	$f_k.ct, f_k.fs$	n-FIFO$^{\text{RSIA}}$	✓	SF	REQ: $f_k.ct^{\text{A}}$ SUP: $f_k.ml^{\text{R}}$	L/Win	LIM per stream
[28]-1	Routing [HC] Scheduling	Custom Heur. SWOTS-(AEAP/ASAP), SWTS (Stream-Inc. w/o BTR)	IM: lt^{R}, |gb|	TT [PA] BE	–	$f_k.fs^{?}$	1-NW$^{\text{INJ}?}$	$\checkmark^{\text{lim}}$	SF	SUP: $f_k.ml^{\text{R}?}$	L/Frame (SWTS is P/Slot)	$0^{?}$
[28]-2	Routing [HC] Scheduling	Custom Heur. SWOTS-(AEAP/ASAP)-WS (Stream-Inc. w/o BTR)	IM: lt^{R}, |gb|	TT [PA] BE	–	$f_k.fs^{?}$	RAB$^{?}$	–	SF	SUP: $f_k.ml^{\text{R}?}$	L/Frame	$0^{?}$
[39]	Routing [JM] Scheduling	ILP (2 variants, 1 w/ rpp)	EX: custom IM: lt^{R}	TT [PA] other	–	$f_k.ct, f_k.fs$	1-NW$^{\text{RSIA}}$	$\checkmark^{\text{lim}}$	SF	SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[26]-1	Queue Assignm. [JM] Scheduling	SMT/OMT	EX: |gcl|	TT	$-^{?}$	$f_k.ct, f_k.fs$	n-FIFO$^{\text{RSIA}}$	✓	SF	REQ: $f_k.ct^{\text{A}}$	L/Frame	∞
[26]-2	Queue Assignm. [SP] Scheduling	Heur. OMT (Stream-Inc. w/o BTR)	EX: |gcl|	TT	$-^{?}$	$f_k.ct, f_k.fs$	n-FIFO$^{\text{RSIA}}$	✓	SF	REQ: $f_k.ct^{\text{A}}$	L/Frame	∞
[26]-3	Scheduling	Custom Heur. NGC (Stream-Inc. w/o BTR)	IM: lt^{R}	TT	$-^{?}$	$f_k.ct, f_k.fs$	1-NW$^{\text{INJ}}$	$\checkmark^{\text{lim}}$	SF	REQ: $f_k.ct^{\text{A}}$	L/Frame	∞
[26]-4	Scheduling (Queue Assignm. ?)	Custom Heur. MF (Stream-Inc. w/o BTR)	$-^{?}$	TT	$-^{?}$	$f_k.ct, f_k.fs$	FIFO$^{?}$	?	SF	REQ: $f_k.ct^{\text{A}}$	L/Frame	∞
[32]	Scheduling	Custom Heur.	IM: lb	TT [PA] BE	?	$f_k.ct, f_k.fs$	n-FIFO$^{\text{RSIA}}$	?	?	SUP: $f_k.ml^{\text{R}?}$	L/Win (incomplete WCD analysis)$^{?}$	$\infty^{?}$
[35]	Routing [SP] Scheduling	Routing: Custom Heur. DAMR Scheduling: Heur. ILP (Stream-Inc. w/o BTR)	IM: lt^{R}	TT (multi-path/ FRER)	–	$f_k.ct, f_k.fs$	1-NW$^{\text{INJ}}$	$\checkmark^{\text{lim}}$	SF	–	L/Frame	0

Tabelle 10.2: Schedulervergleich (Teil 2/3).

	Mathematical Approach			Traffic Features			Schedule Limitations						
	Problem Scope	*Solver/ Algorithm*	*Schedule Optimization*	*Traffic Classes*	*Multi-cast*	*Heterog. Param.*	*Buffering Model*	*Frame Isolation*	*Bridge Delay*	*Latency Bounds*	*Time Slot Type*	*Jitter*
[40]	Routing HC Scheduling	Conflict Graph (Iterative addition of paths/inject times)	EX: sab^{st} IM: lt^{R}	TT	–?	$f_k.ct, f_k.fs$?	1-NW^{INJ}	✓lim	SF	SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[43]	Routing JM Scheduling	ILP w/ rpp (all shortest paths)	EX: lb (2 variants)	TT	–	$f_k.ct, f_k.fs$	?	–?	SF?	–?	L/Win? (missing WCD analysis)?	?
[14]	Queue Assignm. JM Routing JM Scheduling	ILP w/ rpp (Stream-Inc. w/o BTR) (Non-Inc. supported)	EX: lb,pl	TT	✓	$f_k.ct, f_k.fs$	n-$\text{FIFO}^{\text{RSIA}}$	✓	SF	REQ: $f_k.ct^{\text{A}}$ SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[E14]	Routing JM Scheduling	ILP w/ rpp,rcr (4 variants)	EX: lt^{R},pl	TT	✓	$f_k.ct, f_k.fs$	RAB^{RSIA}	–	config. const	SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[13]	Routing JM Scheduling	ILP w/ rpp (13 options)	IM: lt^{R}	TT	–	$f_k.fs$	1-NW^{RSIA}	✓lim	SF,CT	REQ: $f_k.ct^{\text{A}}$	L/Frame	0
[41]	Routing HC Scheduling	Routing: All Simple Paths Scheduling: ILP from [38] (Stream-Inc. w/o BTR)	EX: t_0 at src IM: lt^{R}	TT	–	$f_k.ct, f_k.fs$	1-NW^{RSIA}	✓lim	SF	–	L/Frame	0
[42]	Routing JM Scheduling	ILP w/ rpp, Stream Clustering (Stream-Inc. w/o BTR) (Non-Inc. supported)	EX: span,pl	TT	✓	$f_k.ct, f_k.fs$	n-$\text{FIFO}^{\text{RSIA}}$	✓	SF	SUP: $f_k.ml^{\text{R}}$	L/Frame	0
[29, 30]	Routing HC Scheduling	Custom Heur. (Stream-Inc. w/o BTR)	IM: lb, custom	TT	–	$f_k.ct, f_k.fs$	?	?	SF?	?	L/Win? (missing WCD analysis)?	?
[24]-1	Scheduling	ILP	EX: lt^{A} IM: lt^{R}	TT	–?	$f_k.ct, f_k.fs$	1-NW^{INJ}	✓lim	SF	SUP: $f_k.ml^{\text{A}}$	L/Frame	0
[24]-2	Scheduling	Greedy Heur. (Stream-Inc. w/o BTR)	IM: lt^{A}, lt^{R}	TT	✓s	$f_k.ct, f_k.fs$	1-NW^{INJ}	✓lim	SF	SUP: $f_k.ml^{\text{A}}$	L/Frame	0
TW	Routing JM Queue Assignm. JM Scheduling	ILP w/ rpp (>100 variants)	EX: lt^{R},pl	TT	✓	$f_k.ct, f_k.fs$	RAB^{RSIA} n-$\text{FIFO}^{\text{RSIA}}$ n-NW^{RSIA}	✓	SF,CT	SUP: $f_k.ml^{\text{R}}$	L/Frame	0

Tabelle 10.3: Schedulervergleich (Teil 3/3).

10.1.2.1 Einordnung von *UC* (Arbeiten bis 2017)

In der 2017 veröffentlichten Arbeit [E12] wird das Basismodell *UC* einschließlich Routenvorverarbeitung vorgestellt. Dabei handelt es sich um ein ILP-Modell mit gemeinsamer mathematischer Modellierung von Unicast-Routing und Scheduling (JM), voller Unterstützung für heterogene Streamparameter und RAB-Buffering ohne TSN-spezifische Frameisolation. Das Verfahren von Tămaş-Selicean [8] unterstützt ebenfalls Routing und Scheduling mit RAB-Buffering und darüber hinaus Multicast-Streams, nutzt jedoch Heuristiken für Routing, Scheduleoptimierung und auch die Kopplung der Teilprobleme. Zuvor veröffentlichte mathematische Modelle (SMT/ILP) von Craciunas [27] und Pop [34] integrieren im Gegensatz zum eigenen Modell auch Queuezuweisungen und Frameisolation, gehen jedoch von fest vorgegebenen Routen aus. Eine ILP-Formulierung und eine Heuristik von Duerr [37] gehen nicht nur von festen Routen aus, sondern beschränken das Scheduling außerdem auf NW-Buffering.

Im direkten Vergleich am relevantesten sind somit die ILP-Formulierungen von Nayak [9] und Smirnov [10], da diese wie die eigene Arbeit eine gemeinsame Modellierung von Routing und Scheduling vornehmen. Dabei ist das Modell aus [9] jedoch wesentlich einfacher gehalten, da der Basiszyklus in ein Raster mit Zeitschlitzen von vordefinierter Länge eingeteilt wird, welche ausreichend groß gewählt sind, um einen MTU-Frame über die komplette Route von Sender zu Empfänger zu übertragen (P/Slot). Dieses Vorgehen senkt die erzielbare Netzwerkauslastung bzw. Schedulability von TT-Streams ab und macht eine Einschränkung bei der Hopanzahl notwendig (hier: 8 Hops). In der im gleichen Jahr wie *UC* veröffentlichten Arbeit von Smirnov [10] werden dagegen wie in *UC* feingranulare Zeitschlitze der Frameübertragungen auf jedem Link geplant, wobei sich durch ein rein auf Binärvariablen ausgelegtes Modell große Unterschiede in der resultierenden ILP-Repräsentation ergeben. Während das Modell von Smirnov zwar die Unterstützung von Multicast ermöglicht, ist es auf NW-Buffering beschränkt, da es voraussetzt, dass die Weiterleitungsverzögerung durch eine Bridge zum Zeitpunkt der Modellbildung bereits bekannt ist. Insgesamt ist das in [E12] veröffentlichte Modell *UC* somit das bis dahin einzige mit gemeinsamer Modellierung von Routing, linkspezifischem Scheduling und gleichzeitiger Lösungsraumabdeckung bezüglich Buffering in Bridges.

10.1.2.2 Einordnung von *MCT* (Arbeiten bis 2020)

Die Veröffentlichung im Jahr 2020 hat als Kernthemen das erste multicastfähige ILP-Modell *MCI* für gemeinsames Routing und Scheduling mit RAB-Buffering, sowie das verbesserte Modell *MCT* und die Reduktion der Ressourcenbedingungen (Modelloption *rcr*). Zur Einordnung dieses Modells soll hier nur auf die relevantesten Arbeiten eingegangen werden, welche ebenfalls eine gemeinsame Modellierung von Routing und Scheduling im Rahmen eines exakten Lösungsverfahrens vornehmen.

Bei der Arbeit von Nayak [36] handelt es sich um eine Folgearbeit zu [9], in der für den Ansatz mit pfadweiten Zeitschlitzen (P/Slot) eine ILP-Variante zur Stream-inkrementellen Planung vorgestellt wird, womit eine bessere Skalierbarkeit angestrebt wird. Auch wenn diese Variante außerdem zusätzlich Multicast und heterogene Zykluszeiten unterstützt, bleibt die vereinfachte Modellierung ohne Planung linkspezifischer Übertragungszeitpunkte von Frames erhalten, sodass auch weiterhin Effizienzprobleme und Hoplimitierung existieren. Falk stellt in [38] und [39] zwei grundverschiedene

ILP-Modelle vor, welche jedoch beide kein Multicast unterstützen und auf NW-Buffering beschränkt sind. Die Zielsetzung lag dabei allerdings auch nicht auf einer umfassenderen Modellierung als in vergleichbaren Arbeiten, sondern auf einer ausführlichen Performanceanalyse in [38] und bei der (proaktiven) Verbesserung der QoS-Eigenschaften weiterer Echtzeitstreams (neben den tatsächlich geplanten TT-Streams) in [39]. Die Arbeit von Syed [43] liefert ebenfalls keine Multicast-Unterstützung und folgt mit dem Window-basierten Ansatz (L/Win) einer anderen Art der Zeitschlitzvergabe. Gravierendste Schwächen des Ansatzes sind das Fehlen einer schlüssigen WCD-Analyse und klar erkennbarer Modelle zur Sicherstellung von vorhersagbaren Latenzen sowie Jitter.

Die relevanteste verwandte Arbeit zum Multicast-Modell *MCT* liefert Yu im gleichen Veröffentlichungsjahr in [14]. Das dort vorgestellte ILP-Modell deckt nicht nur die gleichen Funktionen wie *MCT* ab, sondern unterstützt zusätzlich TSN-konforme Queuezuweisungen und Frameisolation. Modellierungsunterschiede gibt es dabei in den Ressourcenbedingungen: Während *MCT* in [E14] eine optimierte Form der Konfliktprüfung über eine Hyperperiode nutzt (vgl. Modelloption *rcr*), reduziert Yu die Anzahl der notwendigen Ressourcenbedingungen durch eine Projektion der Sendezeitpunkte in den Zeitraum des größten gemeinsamen Teilers der Zykluszeiten zweier Streams (GCD-Ansatz).[3] Außerdem wird im Pfadscheduling eine Modellierung genutzt, die mit dem Basismodell *MCI* vergleichbar ist, für das sowohl in [E14] als auch in dieser Arbeit deutliche Performancenachteile gegenüber *MCT* nachgewiesen werden konnten. Resultierende Probleme bei der Skalierbarkeit werden von Yu mit der häufig anzutreffenden Stream-inkrementellen Planung adressiert. Ab 2020 sind somit aus [14, E14] verschiedene ILP-Modelle zur gemeinsamen Planung von Routing und Scheduling inklusive Multicast-Unterstützung bekannt, welche sich jedoch in Modellierungsdetails und der Performanceoptimierung (Stream-inkrementell vs. optimiertes ILP) unterscheiden.

10.1.2.3 Einordnung von Verbesserungen und Erweiterungen (Arbeiten bis 2024)

Seit 2020 ist durch [14] ein ILP-Modell veröffentlicht, das eine gemeinsame Modellierung von TSN-konformen Queuezuweisungen, Routing und Scheduling realisiert und zudem Multicast-Unterstützung und Buffering in Bridges bietet. Damit erfüllt es die auch in den eigenen Arbeiten angestrebte, vollständige Lösungsraumabdeckung. Der Fokus der Entwicklungsarbeit lag daher seitdem noch stärker als zuvor bei Performanceoptimierungen und ausführlichen Analysen, um robuste und vielseitig einsetzbare Modelle zu finden. Somit befasst sich der folgende Vergleich stärker mit konkreten Modellierungsdetails der Lösungsverfahren.

Eine noch nicht genannte Schwäche in [14] ist beispielsweise, dass die Queuezuweisungen und Frameisolation nur in abstrakter Form gezeigt wurden, d.h. unter Nutzung von Implikation und modulo-Operation. Da dies beides keine unterstützten Elemente einer ILP-Formulierung sind, bleibt die konkrete ILP-Repräsentation offen, obwohl diese maßgeblich für die Performance ist. Im Rahmen dieser Arbeit wurden dagegen zwei unterschiedliche ILP-Varianten zur Queuezuweisung entworfen (iq/bq) und miteinander verglichen. Dabei wurde in beiden ausgenutzt, dass durch die Hilfsvariable

[3] Der GCD-Ansatz zur Konfliktprüfung wurde im Rahmen dieser Arbeit ebenfalls evaluiert, jedoch nicht vorgestellt, da vor allem die Rechenzeiten bei hoher Netzwerkauslastung unterlegen gegenüber dem optimierten Hyperperioden-Ansatz waren. Ergebnisse hierfür sind dennoch in [E19] enthalten.

$\mathfrak{a}_{klmxy}$ der Ressourcenbedingungen bereits eine Entscheidung zur Sendereihenfolge der Frames vorliegt, welche bei der Frameisolation wiederverwendet werden kann. Des Weiteren kann äquivalent zur Optimierung der Ressourcenbedingungen durch *rcr* die Anzahl der Bedingungen zur Frameisolation verringert werden (siehe Definition von $FI^{k..l}/FI^{l..k}$ in Kapitel 7.2.2.1). Während solche Details in [14] aufgrund der abstrakten Formulierung unklar bleiben, sind beide Optimierungsschritte auch in keiner der anderen Arbeiten mit ILP-Formulierungen zur Queuezuweisung und Frameisolation [34, 42] zu finden. Darüber hinaus sind nicht alle der in der Literatur genutzten Bedingungen zur Frameisolation im Detail korrekt ausformuliert. So wird in [34, 27] die Position des Gate-Close-Events nicht berücksichtigt, was Fehlerfälle nach Abbildung 7.7c zulässt.

Bezüglich der vielen weiteren in dieser Arbeit vorgestellten Modelloptionen zur Performanceverbesserung finden sich in der Literatur vor allem Vorschläge, die mit der Routenvorverarbeitung vergleichbar sind. In den hier betrachteten Arbeiten findet sich diese erstmals in [9], wobei in der ILP-Formulierung vollständige Pfade zur Auswahl gestellt werden (ähnlich zu *psel-pa/psel-la* aus Kapitel 7.1.5). Die Verwendung streamspezifischer Link- und Bridgemengen (E_k, V_k) bei der Modellbildung findet sich dagegen erstmals in [E12]. Als Routen werden dabei in [E12] alle schleifenfreien Pfade genutzt, während [9, 43] durch die Verwendung aller kürzesten Pfade (sofern mehrere existieren) wesentlich restriktiver sind. In [42, 14] werden dagegen *regelbasiert* solche Links aus der Topologie entfernt, die niemals Teil einer sinnvollen Route des jeweiligen Streams sein können (Ausgangslinks am Empfänger, Eingangslinks am Sender, usw.). Eine vergleichbare Routenvorverarbeitung zu der in dieser Arbeit am besten bewerteten Modelloption *rpp-lbsp* (Kombination zweier bzgl. Lastverteilung und Pfadlänge optimierter Routings) findet sich in der Literatur dagegen nicht. Eine allgemeingültige Aussage zur „optimalen" Routenvorverarbeitung ist allerdings nicht möglich, da zu umfangreiche Abhängigkeiten zur Netzwerktopologie und zu den weiteren Schritten des Lösungsverfahrens bestehen.

Im Gegensatz zur Routenvorverarbeitung findet die in dieser Arbeit als ein Kernthema behandelte Optimierung der ILP-Modelle ohne heuristische Einschränkungen (wie die Routenvorverarbeitung) nur wenig Beachtung. So wurde die Reduktion der Ressourcenbedingungen erstmals in [21] skizziert, bevor sie in [E14] ausgeweitet und hier in Form der Modelloption *rcr* formal beschrieben wurde. In [14] wird zwar mit dem GCD-Ansatz ebenfalls eine spezielle Optimierung der Ressourcenbedingungen vorgenommen, allerdings wird kein Vergleich zum herkömmlichen Hyperperiodenansatz durchgeführt. In [33] wurde eine Optimierung eines Window-basierten Schedulers vorgenommen, indem die Reihenfolge der Sendefenster bereits bei der Modellbildung festgelegt wurde (ohne dabei gültige Lösungen auszuschließen). Obwohl dies zur Optimierung von ILP-Modellen gezählt werden kann, ist die Optimierung nicht relevant für das hier diskutierte framespezifische Scheduling (L/Frame). Die Arbeit von Hellmanns [13] ist dagegen die einzige, in der wie in der hier vorliegenden Arbeit viele verschiedene Modellvarianten eines Basismodells (ähnlich zu *UC*, aber mit NW-Bedingung) erstellt und mit dem expliziten Ziel der Performanceverbesserung miteinander verglichen werden. Die Überschneidung zu den Modelloptimierungen in dieser Arbeit beschränkt sich dabei allerdings auf die Routenvorverarbeitung, welche zugleich der einzige Vorschlag aus [13] ist, der überzeugende Performanceverbesserungen liefert. Andere dort getestete Modellvarianten zeigen dagegen nur geringfügige Performanceeinflüsse oder sogar eine deutliche Verschlechterung.

Insgesamt existieren in der untersuchten Literatur viele der in dieser Arbeit vorgeschlagenen Modelloptimierungen überhaupt nicht. Dies betrifft beispielsweise optimierte Multicastmodelle wie *MCT*/*MCS*, strenger beschränkte Schedulingvariablen nach *ia*, zusätzliche Lösungsfilter wie *lu1*/*ne*, veränderte Entscheidungsvariablen und somit Branchingmöglichkeiten nach *mod*/*modv* und optimierte Frameisolation nach *iq*/*bq*. Auch der Einfluss des ILP-Optimierungsziels wird nicht ausreichend beachtet. In [27, 33, 39] werden zwar einige exakte Modelle mit und ohne Optimierungsziel miteinander verglichen, allerdings wurde in keiner der Arbeiten gezeigt, dass unterschiedliche Optimierungsziele zu ähnlichen Lösungen bei sehr unterschiedlicher Performance führen können (vgl. *LT25* und *SPLT25* in Kapitel 6.4.5). Auch die Nutzung eines Optimierungsziels zur signifikanten Performanceverbesserung beim Auffinden einer ersten gültigen Lösung wurde bisher nicht gezeigt (vgl. *1SP* und *NONE* in Kapitel 6.4.5).

10.2 Quantitativer Vergleich von Modellvereinfachungen

Die Herausforderung beim quantitativen Vergleich der eigenen Modelle mit Lösungsverfahren aus der Literatur liegt darin, dass Ergebnisse aus verschiedenen Publikationen aufgrund der sehr unterschiedlichen Testmethoden bisher kaum vergleichbar sind, während eigene Tests (z.B. mittels der Benchmarkingszenarien) mangels veröffentlichter Implementierungen ebenfalls erschwert werden. Da viele der bisher veröffentlichten Heuristiken darüber hinaus aufgrund der unpräzisen Beschreibungen nur schwer authentisch nachimplementierbar sind, fokussiert sich die folgende Diskussion auf exakte Verfahren, welche durch formale Beschreibungen in vielen Fällen besser reproduzierbar sind.

Verglichen mit den in dieser Arbeit vorgestellten ILP-Modellen finden sich unter diesen exakten Verfahren einerseits Modelle mit grundlegend abweichenden Ansätzen [39, 36, 10] und andererseits sehr ähnliche Modelle [38, 13, 14, 34], die sich durch kleinere Modifikation der hier vorgestellten Modelle erzeugen lassen. Auch wenn ein Vergleich der Modelle dieser Arbeit mit den sich stärker unterscheidenden Modellen aus [39, 36, 10] durchaus von Interesse wäre, wird dieser hier aufgrund des erwarteten Tradeoffs zwischen Aufwand und Ergebnisgüte nicht durchgeführt: Der mit pfadweiten Reservierungen arbeitende Scheduler aus [9, 36] ist bereits in der Theorie ungeeignet für viele der hier getesteten Benchmarkingszenarien, während die Modelle aus [39, 10] umfassende Implementierungsarbeiten erfordern würden, während zugleich weder die in den Publikationen gezeigten Ergebnisse noch theoretische Überlegungen eine vielversprechende Performance vermuten lassen. Somit verbleiben die Modelle mit kleineren Modellierungsunterschieden [38, 13, 14, 34] als sinnvollste Vergleichsmöglichkeit.

Obwohl solche Modelle mit kleineren Modellabweichungen sich in vielen Fällen *vollständig* durch Modelloptionen abbilden ließen, wird eine solche Nachbildung hier weiterhin nicht angestrebt, da dies kaum wertvolle Informationen liefern würde. Bei der vollständigen Nachbildung von Modellen aus der Literatur würden beispielsweise grundsätzlich mehrere Modellierungsunterschiede bestehen, sodass unklar wäre, auf welchen Modellierungsunterschied ggf. vorhandene Performanceunterschiede zurückzuführen wären. Da die genannten Modelle aus der Literatur beispielsweise keine Optimierungen in der Art von *rcr* oder *ia* vornehmen und ggf. andere Schritte zur Routenvorverarbeitung nutzen,

würden diese sehr performancerelevanten Aspekte den Einfluss anderer Modellierungsunterschiede überdecken. Eine vollständige Nachbildung solcher Modelle würde demnach nur zu der (bereits bekannten) Erkenntnis führen, dass Modelle ohne diese Optimierungen signifikant langsamer sind. Interessanter ist dagegen eine Untersuchung *einzelner* Schedule-Einschränkungen dieser Modelle, sodass Performanceeinflüsse einem konkreten Modellierungsunterschied zugeschrieben werden können. Dies umfasst vor allem:

Separate Routing and Scheduling (SRaS) und Routenvorverarbeitung
: Vorgabe fester Routen (*Problem Scope*: „Routing SP“ bzw. kein Routing in Tab. 10.1–10.3) oder reduzierte Routingmöglichkeiten (*Solver/Algorithm*: „w/ rpp“ in Tab. 10.1–10.3)

No-Cross-Cycle (NCC)
: Vermeidung zyklusübergreifender Planung (*Latency Bound*: „REQ: $f_k.ct^{\mathrm{A}}$“ in Tab. 10.1–10.3)

No-Wait (NW)
: Vermeidung von Buffering (*Buffering Model*: „1-NW“ in Tab. 10.1–10.3)

Während die Einschränkung der Routen in der Regel mit der Reduktion der Rechenzeiten begründet wird, werden NCC und NW vorwiegend zur Vereinfachung der Problemmodellierung eingesetzt. Die Kernfrage bei diesen Einschränkungen ist dabei, welcher Tradeoff bezüglich Performance- und Qualitätsindikatoren im Vergleich zu einem weniger restriktiven Ansatz entsteht. Trotz der weiten Verbreitung dieser Ansätze gibt es in der Literatur diesbezüglich bisher keine ausreichenden Analysen. Unter anderem werden derartige Untersuchungen zwar als Teil der Evaluation von neu entwickelten Schedulern in den entsprechenden Publikationen gezeigt, diese weisen jedoch Schwachstellen auf, welche im Folgenden diskutiert werden.

Konkret wird ein Vergleich zwischen JRaS und SRaS außer in der eigenen Arbeit [E12] noch in [42, 14, 10, 36, 55, 41] durchgeführt. Dabei wird in [55] allerdings ein heuristisches Lösungsverfahren, in [41] Stream-inkrementelles Lösen und in [36] pfadweite Zeitschlitze genutzt. Da diese Ansätze grundsätzlich bereits einen stark eingeschränkten Lösungsraum zur Folge haben, bleibt der Einfluss von festem Routing auf exakte Verfahren mit umfassenderer Lösungsraumabdeckung unklar. In [10, 42] wird der Vergleich zwar für umfassendere ILP-Modelle durchgeführt, allerdings ist der Testumfang in beiden Fällen gering und das genutzte feste Routing wird nicht genannt [42] bzw. bleibt unklar, da ein ILP-basiertes Routing genutzt wird, ohne dessen Optimierungsziel zu nennen [10]. Die Analyse in [14] ist wiederum nicht besonders aussagekräftig, da zum einen nur kürzeste Pfade für den SRaS-Ansatz genutzt werden und zum anderen SMT mit ILP verglichen wird, sodass die tatsächliche Ursache von Performanceunterschieden unklar bleibt. Bezüglich der Routenvorverarbeitung finden sich, wie bereits im vorangegangenen Schedulervergleich genannt, einige Vorschläge in [9, 43, 42, 14]. Ein Vergleich unterschiedlicher Verfahren bleibt allerdings aus (abgesehen von zuvor genannten schwachen Vergleichen mit SRaS-Verfahren).

Im Vergleich zu den Routingverfahren sind die NW- und NCC-Einschränkungen in der Literatur noch weniger untersucht. Ein direkter Vergleich von Lösungsverfahren mit und ohne NW-Einschränkung findet sich beispielsweise nur in [26, 28]. Dabei handelt es sich allerdings in beiden Fällen um einen Vergleich für Stream-inkrementelle Heuristiken ohne Backtracking, sodass dieser für

exakte Verfahren mit umfassender Lösungsraumabdeckung kaum Relevanz hat. Die geringe Anzahl entsprechender Analysen lässt sich damit begründen, dass in der Regel nur sich aus den jeweiligen Einschränkungen ergebende, vereinfachte Lösungsansätze entworfen und implementiert wurden. Es steht also kein Referenzmodell ohne die jeweiligen Einschränkungen zum Vergleich zur Verfügung.

Im Gegensatz zu den Veröffentlichungen neuer Scheduler befassen sich [E15] sowie die sehr aktuelle Arbeit [45] speziell mit dem Performancevergleich veröffentlichter Scheduler. Hierzu wurden in [E15] fünf Schedulerkonfigurationen der beteiligten Arbeitsgruppen verwendet und in [45] die Nachimplementierung von 17 Schedulern vorgenommen. Bezüglich der Bewertung von Modelleinschränkungen wie SRaS und NW besteht jedoch bei beiden Arbeiten das übergeordnete Problem, dass bei den verglichenen Schedulern in der Regel mehrere Modellierungsunterschiede zugleich bestehen, sodass nur eine begrenzte Aussagekraft bezüglich *einzelner* Schedule-Einschränkungen vorliegt. Der Performanceeinfluss der NCC-Einschränkung wurde auch in diesen Arbeiten nicht untersucht, sodass für diese bisher überhaupt keine Daten vorliegen.

In diesem Unterkapitel soll daher die in der Literatur bisher mangelhafte Bewertung von SRaS, NW und NCC erfolgen. Die Bewertung von SRaS und Routenvorverarbeitung erfolgt dabei anhand der Ergebnisse früherer Kapitel, bevor Modelloptionen bzgl. NCC und NW erstellt und evaluiert werden. In allen drei Fällen ist der maßgebliche Unterschied zur genannten Literatur, dass die entsprechenden Einschränkungen bei ansonsten identischer Modellierung vorgenommen werden. In der Bewertung wird dabei auch aufgegriffen inwiefern sich die Erkenntnisse gegenüber [E15, 45] unterscheiden.

10.2.1 Bewertung von SRaS und Routenvorverarbeitung

In dieser Arbeit wurden verschiedene Routenvorverarbeitungen (*bpl*, *rpp-ksp*, *rpp-lbsp*) und fünf feste Routings (alle Modelloptionen mit Präfix *r-**) mithilfe des umfassenden Benchmarkings miteinander verglichen. Damit handelt es sich nach bestem Wissen des Autors um den umfangreichsten Vergleich dieser Art. Die Ergebnisse aus Kapitel 6.4.7 haben dabei gezeigt, dass der typische Schedulability-Nachteil eines festen Routings (SRaS) auf Basis kürzester Pfade reduziert werden kann, wenn in der Routingphase die Linkauslastung berücksichtigt wird (*r-nwsp-lu1*) oder gar ein eine explizite Lastverteilung vorgenommen wird (*r-lb-nwsp-lta*). In Kapitel 7.1.5 konnte wiederum gezeigt werden, dass der massive Rechenzeitnachteil eines JRaS-Ansatzes durch eine besonders strikte Routenvorverarbeitung (*rpp-lbsp*) weitgehend elimiert werden kann, während die Vorteile (ggü. SRaS) in Schedulability und Ergebnisgüte erhalten bleiben. Obwohl bei niedriger Netzwerkauslastung ein Rechenzeitnachteil gegenüber SRaS verbleibt, überwiegen unter Einbeziehung der Ergebnisse für hohe Last die Vorteile und machen einen JRaS-Ansatz mit strikter Routenvorverarbeitung zur bevorzugten Lösung für TT-RSP, sofern ein möglichst breiter Einsatzbereich angestrebt wird. Ein derart präziser Performancevergleich verschiedener Routingoptionen ist in [E15, 45] nicht möglich, da veränderte Routingoptionen mit weiteren Modellierungsunterschieden einhergehen. Darüber hinaus kann die in [45] herausgestellte Erkenntnis, dass JRaS-Ansätze bezüglich der Skalierbarkeit bedeutende Probleme gegenüber SRaS-Ansätzen aufweisen, nicht bestätigt werden. Die kritische Einschätzung von JRaS erfolgt dort insbesondere aufgrund weniger restriktiver Routenvorverarbeitung und ist vergleichbar mit dem Zwischenergebnis aus Kapitel 6.4.8.

10.2.2 *No-Wait* und *No-Cross-Cycle* als Modelloption für *UC*

Um die resultierenden Tradeoffs der NW- und NCC-Einschränkungen bewerten zu können, werden diese hier als Modelloption für das Basismodell *UC* realisiert. Dabei kommt der Vorteil des ganzheitlichen Modellierungsansatzes zum Tragen, da beide Einschränkungen im Basismodell standardmäßig nicht vorhanden sind und dieses sich außerdem vergleichsweise einfach zu den stärker eingeschränkten Modellen modifizieren lässt. In *UC* lässt sich die NW-Bedingung durch Modifikation des Pfadschedulings nach Gleichung (6.5) realisieren, während sich die zyklusübergreifende Planung durch eine Zusatzbedingung einschränken lässt.

Modelloption *nw* Bei Aktivierung der Modelloption *nw* wird (6.5) durch (10.1) ausgetauscht. Der einzige Unterschied zwischen den beiden Modellen besteht darin, dass Gleichheit zwischen rechter und linker Seite der Gleichung gefordert wird. Für nicht in der Route des Streams k enthaltene Knoten ist dies trivial erfüllt, da alle Summen 0 ergeben, wohingegen für genutzte Knoten genau die gewünschte NW-Bedingung zwischen den Sendezeitpunkten auf Eingangs- und Ausgangslink realisiert wird.

$$\forall k \in F,\ \forall i \in V_k^{br} \qquad \sum_{m \in E_{ki}^{\rightarrow}} \boldsymbol{t}_{km}^{\boldsymbol{abs}} - \sum_{m \in E_{ki}^{\leftarrow}} \boldsymbol{t}_{km}^{\boldsymbol{abs}} = \sum_{m \in E_{ki}^{\leftarrow}} d_{km}^{fwd\prime} \cdot \mathfrak{p}_{km} \tag{10.1}$$

Modelloption *cc* Bei dieser Modelloption wird die zyklusübergreifende Planung durch Ergänzung von (10.2) eingeschränkt. Darin kommt ein wählbarer Parameter cc^{no} zum Einsatz, wobei kleinere Werte das Scheduling stärker einschränken. Die in der Literatur häufig anzutreffende Variante der vollständigen Vermeidung zyklusübergreifender Planung (NCC) entspricht $cc^{no} := 0$. In der Evaluation ist der Wert von cc^{no} als Suffix zur Modelloption angegeben, wie beispielsweise *cc0* für $cc^{no} := 0$.

$$\forall k \in F,\ \forall m \in E_k \qquad \boldsymbol{t}_{km}^{\boldsymbol{abs}} + sl_{km} \leq (cc^{no} + 1) \cdot f_k.ct \tag{10.2}$$

Es sei angemerkt, dass *cc* in dieser Form nur für *UC*/*MCS* nutzbar ist, während für *MCT* ein zusätzlicher M-Term notwendig wäre, der die Bedingung für ungenutzte Links ausschaltet. Außerdem ist *nw* nur in Kombination mit *ia* nutzbar, da die Differenz auf der linken Seite von (10.1) immer ganzzahlig ist ($\boldsymbol{t}_{km}^{\boldsymbol{abs}}$ enthält nur ganzzahlige Variablen), während $d_{km}^{fwd\prime}$ auf der rechten Seite der Gleichung nur unter Nutzung von *ia* ebenfalls ganzzahlig wird.

10.2.3 Evaluation der Modellvereinfachungen

Anhand der in Abbildung 10.5 gezeigten Ergebnisse für den optimierenden Fall (Optimierungsziel *SPLT25*) soll hier zunächst die Modelloption *nw* bewertet werden. Für niedrige Netzwerkauslastung zeigen sich erwartungsgemäß keinerlei Probleme bei der Schedulability und Streamlatenzen nahe der idealen Latenzen. Die erzielten Rechenzeiten sind dabei geringer als ohne *nw*, was sich intuitiv damit erklären lässt, dass der Lösungsraum bezüglich des Optimierungsziels der Streamlatenzen ausschließlich gute (nahezu optimale) Lösungen enthält und somit im Branching des Solvers keine Zeit für das Durchsuchen von Bereichen des Lösungsraumes aufgewendet werden kann, die weniger gute Lösungen enthalten. Ohne *nw* sind diese dagegen im Lösungsraum enthalten und der Branching-Prozess betrachtet auch diese weniger guten Bereiche. Diese These lässt sich mit Blick auf den nichtoptimierenden Fall (Optimierungsziel *1SP*) bestätigen, in dem der Performanceunterschied

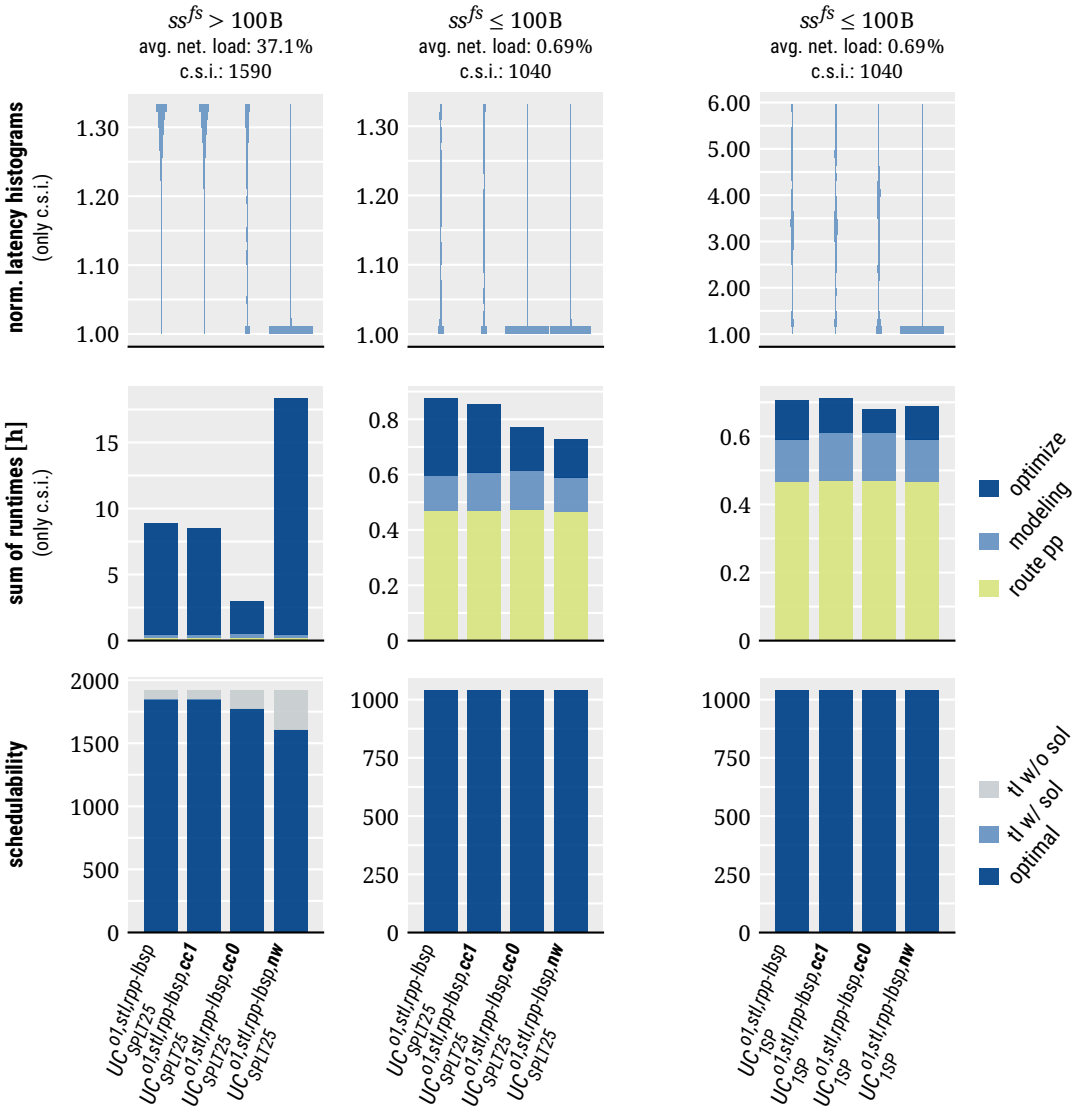


Abbildung 10.5: Schedulability, c.s.i.-Rechenzeiten und Latenzverteilung bei Aktivierung von *cc* und *nw*.

des relevanten Lösungsschritts (*optimize*) nahezu verschwindet. Bei hoher Netzwerkauslastung stellt sich erwartungsgemäß ein deutlicher Nachteil bei der Schedulability ein, da die durch *nw* ausgeschlossenen Lösungen in diesem Fall relevant sind, um überhaupt eine Lösung zu finden (d.h., Buffering in Bridges kann zur Konfliktvermeidung notwendig sein). Darüber hinaus zeigt sich auch für die gelösten Fälle ein deutlicher Rechenzeitanstieg gegenüber Modellen ohne *nw*, was ebenfalls auf die geringere Anzahl an Lösungsmöglichkeiten zurückzuführen ist. Insgesamt wird die Modelloption *nw* hier so bewertet, dass Modelle ohne diese Einschränkung aufgrund der besseren Anwendbarkeit bei hoher Netzwerkauslastung mehr Einsatzszenarien abdecken und daher in der Regel bevorzugt werden sollten. Ausnahmen könnten Anwendungen mit besonders strikten Latenzanforderungen sein, sodass stets die nahezu idealen Streamlatenzen erreicht werden sollen. Für diesen Fall können allerdings auch Modelle ohne *nw* und mit kleinerer Optimality Gap in Betracht gezogen werden.

Bezüglich der zyklusübergreifenden Planung zeigen die Ergebnisse, dass moderate Einschränkungen (hier *cc1*, in [E19] auch für *cc2* gezeigt) keinen relevanten Einfluss haben. Der Fall NCC bzw. *cc0* zeigt dagegen sowohl bei hoher als auch bei niedriger Netzwerkauslastung deutliche Vorteile in den Rechenzeiten. Dies ist unter anderem auf die implizit enthaltenen Modellvereinfachungen zurückzuführen. So wird beispielsweise die Entscheidungsvariable $\mathfrak{o}_{km}$ obsolet, da sie immer den Wert 0 annehmen muss, während einige der Ressourcenbedingungen ebenfalls immer erfüllt sind (vgl. z.B. $(x, y) = (2, 2)$ in Abb. 7.8). Darüber hinaus handelt es sich bei den aus dem Lösungsraum ausgeschlossenen Bereichen vorwiegend um solche, die ungünstige Lösungen bezüglich der Streamlatenzen enthalten. Die Suche wird also stärker auf günstige Bereiche eingeschränkt, ohne dabei durch die Vermeidung von Buffering so restriktiv wie *nw* vorzugehen. Der durch den Ausschluss von Lösungen hervorgerufene Nachteil bezüglich der Schedulability bei hoher Last fällt dabei gering aus, was sich intuitiv erklären lässt. So ist die zyklusübergreifende Planung vor allem dann hilfreich, wenn die (idealen) Streamlatenzen nahe an der Zykluszeit des jeweiligen Streams liegen, da in diesem Fall sehr viele zusätzliche Schedulingmöglichkeiten durch die zyklusübergreifende Planung zur Verfügung gestellt werden. Je größer die Zykluszeit im Vergleich zur Streamlatenz wird, desto weniger relevant ist die zyklusübergreifende Planung. Da die eingesetzten Benchmarkingszenarien Cut-Through definieren, sind die erzielbaren Streamlatenzen gering und selbst in den für *cc0* ungünstigen Szenarien stehen $18\,\mu s - 37\,\mu s$ einer Zykluszeit von mindestens $100\,\mu s$ gegenüber. Somit sind die Benchmarkingszenarien für *cc0* weitgehend unproblematisch bzw. sogar günstig gewählt. Insgesamt ist die Vermeidung der zyklusübergreifenden Planung also durchaus empfehlenswert, solange keine Szenarien mit ungünstigem Verhältnis aus Streamlatenzen und Zykluszeiten geplant werden sollen. Diese Erkenntnis ist insofern bemerkenswert, da keine der Arbeiten aus der Literatur, die dieses Vorgehen anwenden, diese Einschränkung als Performanceoptimierung betrachtet, sondern diese vielmehr zur Vereinfachung der Modellierung und ohne weitergehende Diskussion oder Analyse eingeführt wird.

Die Performanceeinflüsse von *nw* und *cc* wurden in dieser Arbeit zusätzlich auch unter Nutzung von *modv* und *iq* (Planung mit Frameisolation) analysiert, was in Abbildung 10.6 gezeigt ist (ohne *nw* zwecks Skalierung der y-Achse). Grundsätzlich bestätigen sich dabei die bisher erläuterten Ergebnisse, wobei der Schedulability-Nachteil beim Hinzuschalten von *cc0* für die weiteren Modellvarianten sogar geringer ausfällt. Dies ist darauf zurückzuführen, dass sowohl *modv* als auch *iq* bereits von sich aus zu einer etwas geringeren Schedulability bei hoher Last führen. Somit sind einige der für *cc0* problematischen Szenarien bereits im Vergleichsmodell ohne *cc0* ungelöst, sodass *cc0* hier nicht mehr im Nachteil ist. Eine weitere bemerkenswerte Beobachtung ist, dass die Modelle mit und ohne *modv* nach dem Hinzuschalten von *cc0* die gleiche Performance zeigen. Dies ist jedoch plausibel, da *cc0* diese beiden Modellvarianten durch das Erfordern von $\mathfrak{o}_{km} = 0$ (für alle Streams und Links) auf beinahe identische Modelle reduziert. Zukünftig sollte evaluiert werden, ob sich die Rechenzeitvorteile von *nw* und *cc0* auch ohne die damit einhergehenden Schedulability-Nachteile realisieren lassen. Hierfür können beispielsweise Optionen der ILP-Solver genutzt werden, um die entsprechenden Bereiche des Lösungsraums (z.B. $\mathfrak{o}_{km} = 0$) bevorzugt zu durchsuchen, ohne die entsprechende Einschränkung durch eine Randbedingung zu erzwingen.

Während die Einschränkung zyklusübergreifender Planung in [E15, 45] nicht analysiert wird, bieten

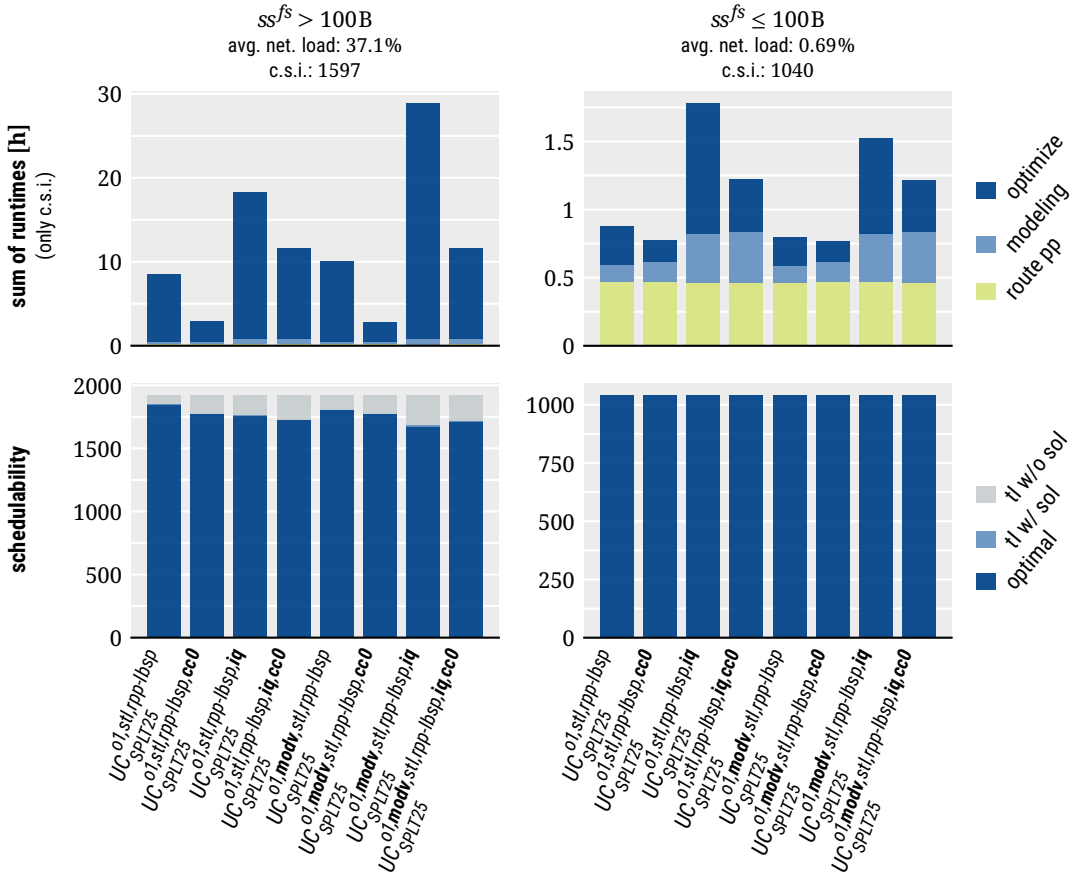


Abbildung 10.6: Schedulability und c.s.i.-Rechenzeiten bei Aktivierung von *cc0* für verschiedene Kombinationen von *modv*/*iq*.

beide Arbeiten einen Vergleich von Schedulern mit und ohne NW-Bedingung, wobei diese allerdings noch weiteren Modellierungsunterschieden unterliegen. Daher wird in [E15] von einer Bewertung der NW-Einschränkung abgesehen. In [45] werden die Ergebnisse dagegen so interpretiert, dass das Zulassen von Buffering aufgrund der höheren Komplexität bei der Modellierung möglicherweise zu keinen praxisrelevanten Vorteilen führt. Die hier gezeigte Analyse betrachtet NW dagegen als dedizierte Modelloption und kann durch die Strukturierung des Benchmarkings in Testcases bzw. die Gruppierung nach Netzwerkauslastung präzisere Aussagen treffen. Konkret lässt sich sagen, dass Modelle ohne NW-Einschränkung aufgrund der Vorteile bei hoher Last eindeutig zu bevorzugen sind, da diese Vorteile wesentlich ausgeprägter sind als die Nachteile bei niedriger Last. Darüber hinaus ist es naheliegend, dass die Nachteile bei niedriger Last durch weitere Modelloptimierungen adressiert werden können. Den Ansichten aus [45] kann somit nicht zugestimmt werden.

10.3 Zusammenfassung des Literaturvergleichs

In diesem Kapitel wurden die Unterscheidungsmerkmale von Schedulern herausgearbeitet, die in dieser Arbeit beschriebenen Modelle hinsichtlich dieser Kriterien mit anderen veröffentlichten

Schedulern verglichen und abschließend eine Performancebewertung von in der Literatur häufig anzutreffenden Modellvereinfachungen vorgenommen. Neben den *mathematischen Methoden* unterscheiden sich Scheduler für TT-RSP in den *unterstützen Streameigenschaften* und den *zusätzlichen Einschränkung* bezüglich der auffindbaren Lösungen. Unterstützte Streameigenschaften wie beispielsweise Multicast oder heterogene Sendeintervalle stellen dabei Einschränkungen hinsichtlich der Eingangsdaten dar, also der überhaupt bearbeitbaren Szenarien. Unterschiedliche Einschränkungen beim Scheduling bedeuten dagegen, dass verschiedene Scheduler oftmals eine abweichende Lösungsraumabdeckung bieten. Konkret kann es daher vorkommen, dass bestimmte Lösungen einer Probleminstanz von einem Scheduler zurückgegeben werden, die von einem anderen Scheduler überhaupt nicht in Betracht gezogen werden.

Ordnet man die in [E12, E14] veröffentlichten und in dieser Arbeit erweiterten ILP-Modelle nach diesen Kriterien ein, so zeichnen sie sich im Vergleich zur Literatur durch eine besonders umfassende Lösungsraumabdeckung und Unterstützung von Eingangsdaten aus:

- *UC* ist das erste veröffentlichte JRaS-Modell mit linkspezifischem Scheduling und Unterstützung für Buffering in Bridges.
- *MCT* ist zusammen mit dem im gleichen Jahr veröffentlichten Modell aus [14] das erste JRaS-Modell mit linkspezifischem Scheduling, Buffering und Multicastunterstützung. Als einzigartiges Unterscheidungsmerkmal für *MCT* ist hierbei die Performanceoptimierung mittels einer verbesserten ILP-Repräsentation zu nennen.
- Viele der Modellvariationen aus dieser Monographie wurden bisher nicht in vergleichbarer Weise beschrieben und analysiert. Dies umfasst u.a. die Multicastmodelle *MCT*/*MCS*, Modelloptionen *ia*/*modv* oder den Vergleich zweier Varianten zur Frameisolation (*iq*/*bq*).

Insgesamt handelt es sich bei dieser Arbeit nach bestem Wissen des Autors um die bisher umfangreichste Diskussion und Analyse von ILP-Modellen zum Lösen von TT-RSP.

Der in diesem Kapitel vorgenommene Performancevergleich populärer Modellvereinfachungen zeichnet sich im Vergleich zur Literatur (z.B. [45]) u.a. dadurch aus, dass *einzelne* Modellveränderungen isoliert bewertet werden. Darüber hinaus werden Modellvarianten verglichen, für die bisher überhaupt keine Performanceanalysen vorlagen, wie beispielsweise die Routenvorverarbeitung *rpp-lbsp* oder variable Einschränkungen bei der zyklusübergreifenden Planung (Modelloption *cc*). Wesentliche in der Literatur bisher nicht gezeigte Erkenntnisse sind:

- Ein JRaS-Ansatz mit besonders strikter Routenvorverarbeitung kann die grundsätzlichen Vorteile der gemeinsamen Betrachtung von Routing und Scheduling erhalten, während sich die Rechenzeiten an die von SRaS-Ansätzen annähern.
- Eine Einschränkung auf NW-Scheduling sollte in einem Scheduler mit allgemeiner Anwendbarkeit vermieden werden, da die Nachteile bei hoher Netzwerkauslastung gegenüber geringen Rechenzeitvorteilen bei niedriger Last überwiegen.
- Die Vermeidung zyklusübergreifender Planung (*cc0*) reduziert die Rechenzeiten des Schedulers sowohl bei niedriger als auch hoher Netzwerkauslastung bei nur geringfügiger Auswirkung auf die Schedulability unter hoher Last.

Insgesamt deuten die Ergebnisse in dieser Arbeit darauf hin, dass komplexere ILP-Modelle (d.h. JRaS mit Buffering und ggf. Queuezuweisungen) in der Literatur bisher zu pessimistisch bewertet werden. Grund hierfür sind fehlende Optimierungen der komplexeren Modelle sowie das Testen bei zu geringer Netzwerkauslastung (wobei diese oftmals nicht explizit charakterisiert wird), sodass Rechenzeitvorteile vereinfachter Modelle in den Ergebnissen stärker vertreten sind als etwaige Nachteile bei hoher Netzwerkauslastung. Ziel und Aufgabe zukünftiger Entwicklungen ist es, die jeweiligen Vorteile komplexer und vereinfachter Modelle noch stärker in einem Lösungsverfahren zusammenzuführen.

Kapitel 11

Zusammenfassung und weitere Entwicklung

Die Kernthemen dieser Arbeit lagen beim Entwurf eines weitreichend optimierten ILP-Modells zur Planung des zeitgesteuerten Verkehrs innerhalb von Echtzeitnetzwerken sowie dem Entwurf eines umfassenden Benchmarkings für derartige Scheduler. Im Folgenden werden die wesentlichen Herausforderungen dieser Arbeit, erzielte Ergebnisse sowie Möglichkeiten zur weiteren Entwicklung zusammengefasst.

11.1 Herausforderungen in der Entwicklung

Zu Beginn der Arbeit lagen wesentliche Hürden bei der schwachen Informationslage im Themenbereich zeitgesteuerter Netzwerke auf Basis von Ethernet. Da TSN sich zu Beginn der eigenen Arbeiten (d.h. insbesondere der Ausarbeitung von [E11, E12]) zu weiten Teilen noch im Entwurfsstadium befand, waren die angestrebte Funktionalität und Konfigurationsarchitektur noch nicht vollständig spezifiziert, wichtige Standarddokumente teilweise noch nicht verfügbar und nur wenige relevante Forschungsarbeiten veröffentlicht. Folglich mussten zu Beginn bedeutende Entwurfsentscheidungen getroffen werden, ohne dass etablierte Methoden und Annahmen aus zuvor veröffentlichten Arbeiten zur Verfügung standen. Daraus resultierte z.B. der Fokus auf die gemeinsame Modellierung von Routing und Scheduling und das allgemein gehaltene Buffering in Bridges ohne Queuezuweisungen, wobei formale Beschreibung und Implementierung möglichst modular und erweiterbar umgesetzt wurden, um zunächst noch unklare Anforderungen später ergänzen zu können.

Dieser modulare Aufbau resultierte in einer Vielzahl von ILP-Modellvarianten, die sich im Funktionsumfang (z.B. Multicast, Queuezuweisungen), getroffenen Annahmen (z.B. NCC, NW), Optimierungszielen (z.B. Pfadlängen, Streamlatenzen) und Optionen zur Performanceoptimierung unterscheiden. Für diese Vielfalt an Modellvarianten lagen dabei aus der Literatur bislang überhaupt keine Vergleiche oder nur Vergleiche mit unzureichendem Testumfang vor, sodass neben den Lösungsverfahren selbst auch umfassende Teststrategien entwickelt werden mussten. Das hierzu entworfene Benchmarking ist dabei geprägt von den vielfältigen relevanten Faktoren für die Schedu-

lerperformance. Selbst ohne die Variation der Verkehrsklassen (siehe Kapitel 4.2.1) wurden in den Benchmarkingszenarien die sechs Einflussfaktoren Netzwerkstruktur, Netzwerkgröße, Verkehrsmustergröße, Streamzykluszeit, Framegrößen und Latenzschranken berücksichtigt und Testcases mit verschiedenen Parametersweeps entworfen. Insgesamt steht also eine große Anzahl von Schedulern einer Vielzahl möglicher Eingangsdaten gegenüber, wobei zwecks statischer Signifikanz außerdem Rechenwiederholungen durchgeführt werden müssen. Die zwei Kernprobleme die sich aus dieser komplexen Kombination aus Modellvarianten und Eingangsdaten ergeben, sind die *Testdauer* und die *schwierige Interpretation der Ergebnisse*. Die problematische Testdauer ergibt sich dabei aus der Vielzahl der zu lösenden Probleminstanzen, wobei jede einzelne Instanz bereits hohe Rechenzeiten von mehreren Hundert Sekunden aufweisen kann. Die Interpretation der Ergebnisse ist problematisch, da viele der Modellvarianten zu Tradeoffs führen, bei denen die Performancegewinne in einigen Testcases gegenüber Nachteilen in anderen Testcases abgewogen werden müssen. Darüber hinaus ist es für das Erkennen solcher Tradeoffs notwendig, eine Vielzahl von Einzelergebnissen zu betrachten. Eine Reduktion auf eine einzige Metrik zum Ranking von Schedulern ist somit nicht möglich.

In dieser Arbeit wurde dem Problem der Testdauer einerseits begegnet, indem eine Verteilung unabhängiger Probleminstanzen auf ein Rechencluster aus Systemen mit identischer Hardware und Software vorgenommen wurde. Andererseits wurde im Rahmen eines iterativen Entwicklungsprozesses angestrebt, allgemeingültige Performanceverbesserungen frühzeitig einzubringen, um die Dauer folgender Tests zu verringern. Des Weiteren wurde die Interpretation von Ergebnissen vereinfacht, indem diese gruppiert nach Szenarien mit hoher bzw. niedriger Netzwerklast zusammengefasst wurden, da es sich hierbei oftmals um den größten Einflussfaktor auf das Schedulerverhalten handelt.

Die genannten Grundprobleme von Testdauer und Ergebnisinterpretation sind jedoch unabhängig von den hier entworfenen Lösungsverfahren und werden voraussichtlich auch zukünftige Arbeiten in dem Bereich begleiten. Bezüglich der Schedulingverfahren wurde im Rahmen dieser Arbeit umfassend nachgewiesen, dass auch die Analyse kleinster Anpassungen unverzichtbar ist, um optimale Ergebnisse zu erzielen. Der Testumfang lässt sich ebenfalls nicht reduzieren, solange die Entwicklung eines robusten und vielseitig einsetzbaren Schedulers im Vordergrund steht. Ein geringerer Testumfang ist dagegen denkbar, falls die Entwicklung eines Schedulers für einen bestimmten Einsatzbereich mit enger eingrenzbaren Szenarioparametern angestrebt wird.

11.2 Zusammenfassung erzielter Ergebnisse

Die beiden wichtigsten Resultate dieser Arbeit sind die *Benchmarkingszenarien* sowie das weitreichend optimierte *ILP-Modell* zur Planung von zeitgesteuertem Netzwerkverkehr. Die Benchmarkingszenarien umfassen einen Unicast- sowie Multicast-Datensatz und berücksichtigen darin die zuvor genannten sechs Einflussfaktoren auf die Schedulerperformance. Im Vergleich zu bisherigen Testmethoden aus der Literatur decken sie somit vielfältigere Parameterkombinationen ab und beachten auch die Interaktion bestimmter Parameter. Die zweistufige Darstellung der Ergebnisse, einerseits geordnet nach Testcases und andererseits zusammengefasst nach hoher bzw. niedriger Last ermöglicht dabei sowohl eine übergeordnete Performanceeinschätzung von Schedulern als

auch eine detaillierte Analyse der Stärken und Schwächen. Zwecks Wiederverwendbarkeit sind die Szenarien in [E18] veröffentlicht und dokumentiert. Das modular aufgebaute ILP-Modell ist in Anbetracht der vielen Modelloptionen das umfassendste, bisher in der Literatur beschriebene Modell. Dabei sind sowohl die Aspekte der Performanceoptimierung als auch der Funktionalität (Unterstützung von SF/CT-Bridges, Multicast, Queuezuweisungen, RAB/NW/FIFO-Buffering) bisher in keiner anderen Arbeit in vergleichbarer Weise beschrieben worden. Insgesamt konnte somit auch die ursprüngliche Zielsetzung erfüllt werden, ein performantes Referenzmodell mit bestmöglicher Lösungsraumabdeckung zu entwerfen.

Aus der Arbeit an diesen beiden Kernthemen der Arbeit sind außerdem ein unabhängiges *Verifikationsmodell* und die nach bestem Wissen des Autors *umfangreichste Evaluation ILP-basierter Lösungsverfahren* hervorgegangen. Die Ergebnisdaten dieser Evaluation sind in [E19] veröffentlicht und zeigen einen umfassenden Performancevergleich von verschiedenen Modellierungsansätzen, deren Performanceeinfluss zuvor nur unzureichend untersucht war. Anhand dieser Evaluation konnten u.a. folgende Thesen nachgewiesen werden:

- *Exakte Modelloptimierungen* (d.h. ohne heuristische Einschränkungen), wie beispielsweise ganzzahlige Randbedingungen (*ia*) oder der Entfernung redundanter Bedingungen (*rcr*), können einen maßgeblichen Beitrag zur Performance ILP-basierter Scheduler leisten.
- Ein kombiniertes Routing- und Schedulingverfahren mit *strikter Routenvorverarbeitung* erzielt eine konkurrenzfähige Performance zu Verfahren mit separatem Routing, jedoch ohne deren Nachteile bezüglich Lösungsgüte und Schedulability.
- Bei strikter Routenvorverarbeitung können *Multicastmodelle* mit vernachlässigbaren Performanceeinbußen gegenüber ähnlich strukturierten Unicastmodellen realisiert werden.
- ILP-Modelle für ein *kombiniertes Routing, Scheduling und Queuezuweisungen* sind mit praxistauglicher Performance realisierbar, sodass Szenarien mit einigen Dutzend Bridges und Endgeräten sowie mehr als hundert Echtzeitstreams in weniger als $20\,\mathrm{min}$ planbar sind.
- TSN-konforme *Queuezuweisungen und Frameisolation* führen zu signifikanten Performanceeinbußen, haben jedoch nur geringe Nachteile bei der Schedulability. Technische Limitierungen von TSN-Bridges auf 8 FIFO-Queues sind in der praktischen Umsetzung von zeitgesteuertem Verkehr somit in der Regel kein maßgebliches Hindernis.
- In der Literatur häufig anzutreffende *No-Wait-Modelle* führen zu vermeidbaren Nachteilen bei Rechenzeit und Schedulability, wohingegen der *Ausschluss von zyklusübergreifender Planung* Möglichkeiten zur Performanceoptimierung bietet, solange die Streamlatenzen klein gegenüber den Zykluszeiten sind.

Bisher wurden diese Themen in der Literatur nicht in vergleichbarer Weise diskutiert, da in der Regel sowohl hinreichend modulare Modelle als auch entsprechend vielseitige Testumgebungen fehlten.

11.3 Weitere Entwicklung

Sowohl die Benchmarkingszenarien als auch die ILP-Modelle bieten umfassendes Potential zur weiteren Entwicklung. Bei den Benchmarkingszenarien wurde beispielsweise der Einfluss verschiedener zeitgesteuerter Verkehrsklassen (d.h. Variation der Zykluszeiten und Framegrößen innerhalb eines Verkehrsmusters nach Kapitel 4.2.1) noch nicht ausreichend untersucht. Die im Rahmen dieser Arbeit genutzten Verkehrsmuster enthalten zwar heterogene Streameigenschaften, die Zusammensetzung dieser Streameigenschaften wurde jedoch nicht variiert. Des Weiteren muss zukünftig untersucht werden, ob reale Szenarien durch die verschiedenen Benchmarkingszenarien ausreichend repräsentiert werden oder ob bestimmte Parameterkombinationen noch ergänzt werden müssen. Bisher sind derartige Analysen problematisch, da keine umfassenden, öffentlich verfügbaren Datensätze existieren, die reale Echtzeitnetzwerke charakterisieren. Darüber hinaus muss zukünftig der Schwierigkeitsgrad der bisher definierten Testcases durch Parameteränderungen an stetig steigende Fähigkeiten der Scheduler angepasst werden. Zugleich sollte der Testaufwand insgesamt nicht steigen oder gar gesenkt werden, um Vergleichstests zukünftig zu ermöglichen bzw. zu vereinfachen. Hierzu könnte beispielsweise analysiert werden, inwiefern bestimmte Parameterkombinationen der bisherigen Testcases weggelassen werden können, da sie nur einen geringen Mehrwert (d.h. Erkenntnisgewinn) gegenüber anderen Parameterkombinationen hervorbringen.

Bezüglich der ILP-Modelle wurden 3 Kernthemen zur Weiterentwicklung identifiziert:

- Die tiefergehende Analyse des Lösungsvorgangs des Solvers, um weitere exakte Modelloptimierungen vornehmen zu können. Dies umfasst neben der Umformulierung von Randbedingungen auch den Eingriff in den Lösungsvorgang mittels Optimierungszielen, Branching-Prioritäten und Variablen-Hints.
- Die Umsetzung inkrementeller Lösungsverfahren auf Basis der bisher entwickelten Modelle. Dabei können u.a. bisher kaum betrachtete Backtracking-Ansätze mittels IIS oder dem Hinzufügen von Routen untersucht werden.
- Die funktionale Erweiterung hinsichtlich Ablaufplanung auf den Endgeräten, Unterstützung weiterer IE-Systeme neben TSN sowie weiterer Verkehrsklassen neben dem zeitgesteuerten TT-Verkehr. Relevante Verkehrsklassen sind dabei der AVB-Verkehr und, als Zwischenschritt zwischen AVB und TT, auch TT-Verkehr mit zulässigem Jitter.

In all diesen Möglichkeiten zur Weiterentwicklung nehmen die im Rahmen dieser Arbeit entworfenen Modelle und Benchmarkingszenarien eine zentrale Rolle ein. Die optimierten ILP-Modelle sind zugleich die Basis für inkrementelle Lösungsverfahren und erweiterte Modelle, sowie Referenz bezüglich Performance- und Gütekriterien. Das vorgestellte Benchmarking stellt wiederum die Entwicklung möglichst robuster und allgemein nutzbarer Scheduler sicher.

Anhang A

Liste mathematischer Symbole

In diesem Anhang werden häufig verwendete mathematische Symbole erläutert. Symbole, die sich auf einziges Kapitel beschränken, werden hier nicht wiederholt.

A.1 Eingangsdaten des Schedulers und abgeleitete Größen

A.1.1 Netzwerktopologie

$\mathcal{G} \coloneqq (\mathcal{V}, \mathcal{E})$	Netzwerktopologie
$\mathcal{V} \coloneqq \mathcal{V}^{br} \cup \mathcal{V}^{es}$	Menge der Knoten der Netzwerktopologie
$\mathcal{V}^{br} \subseteq \mathcal{V}$	Menge der Bridges
$\mathcal{V}^{es} \subseteq \mathcal{V}$	Menge der Endgeräte
$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$	Menge gerichteter Kanten (Links) der Netzwerktopologie
$(v_i)_{i \in \mathcal{V}^{br}}$	Familie der Bridgeeigenschaften
$v_i \coloneqq (v_i.hb, v_i.prc)$	Eigenschaften von Bridge i
$v_i.hb$	Headerbytes, die eine Cut-Through-Bridge für eine Weiterleitungsentscheidung benötigt ($v_i.hb \coloneqq \infty$ für Store-and-Forward-Bridge)
$v_i.prc$	Statische Verarbeitungsverzögerung von Bridge i
$(e_m)_{m \in \mathcal{E}}$	Familie der Linkeigenschaften
$e_m \coloneqq (e_m.ls, e_m.prp)$	Eigenschaften von Link m
$e_m.ls$	Linkbandbreite von Link m
$e_m.prp$	Signallaufzeit von Link m
$V_k \subseteq \mathcal{V}$	Zulässige Knoten zum Routing von Stream k laut Routenvorverarbeitung
$V_k^{br} \coloneqq V_k \setminus (\{f_k.src\} \cup f_k.dsts)$	Menge der durch Stream k nutzbaren Bridges
$E_k \subseteq \mathcal{E}$	Zulässige Kanten zum Routing von Stream k laut Routenvorverarbeitung
$E_{kl}^{\cap} \coloneqq E_k \cap E_l$	Potentiell gemeinsam genutzte Kanten der Streams k und l
$E_{ki}^{\leftarrow} \coloneqq \{m \in E_k \mid m_1 = i\}$	Durch Stream k nutzbare Eingangslinks von Knoten i
$E_{ki}^{\rightarrow} \coloneqq \{m \in E_k \mid m_0 = i\}$	Durch Stream k nutzbare Ausgangslinks von Knoten i

A.1.2 Verkehrsmuster

F	Menge der Stream-IDs des Verkehrsmusters
$(f_k)_{k \in F}$	Familie der Streameigenschaften
$f_k := (f_k.src, f_k.dsts, f_k.ct, f_k.fs, f_k.ml)$	Eigenschaften von Stream k
$f_k.src$	Sender (Talker) von Stream k
$f_k.dsts$	Menge der Empfänger (Listener) von Stream k
$f_k.ct$	Sendeintervall/Zykluszeit von Stream k
$f_k.fs$	(Maximale) Layer-2-Framegröße von Stream k
$f_k.ml$	Relative Latenzschranke von Stream k
ss^{ct}	Parameter des Szenariogenerators zur Festlegung der Sendeintervalle $f_k.ct$ (siehe §4.2.1)
ss^{fs}	Parameter des Szenariogenerators zur Festlegung der Framegrößen $f_k.fs$ (siehe §4.2.1)
ss^{lf}	Parameter des Szenariogenerators zur Festlegung der Latenzschranken $f_k.ml$ (siehe §4.2.1)
$hp_{kl} := \mathrm{lcm}(f_k.ct, f_l.ct)$	Hyperperiode der Streams k und l

A.1.3 Verzögerungszeiten

$pre := 7\mathrm{B}$	Größe der Präambel eines Frames (konstant für Ethernet)
$sfd := 1\mathrm{B}$	Größe des SFDs eines Frames (konstant für Ethernet)
$ifg := 12\mathrm{B}$	Größe der IFG zwischen Frames (konstant für Ethernet)
$d_{km}^{fwd} := \frac{\min(v_{m_1}.hb, f_k.fs + pre + sfd)}{e_m.ls} + e_m.prp + v_{m_1}.prc$	Verzögerung von Stream k beim Durchlaufen von Link m und Bridge m_1
$d_m^{min} := \frac{\min(v_{m_1}.hb, 64\mathrm{B} + pre + sfd)}{e_m.ls} + e_m.prp + v_{m_1}.prc$	Verzögerung eines Minimalframes beim Durchlaufen von Link m und Bridge m_1
$d_{km}^{rcv} := \frac{f_k.fs + pre + sfd}{e_m.ls} + e_m.prp$	Verzögerung beim Empfang von Stream k über Link m
$sl_{km} := \frac{f_k.fs + pre + sfd + ifg}{e_m.ls}$	Sendedauer (Zeitschlitzlänge) des Streams k auf Link m
$D(x) := \begin{cases} x \\ \lceil x \rceil \end{cases}$	Initiale Definition der Vorverarbeitung für Verzögerungszeiten Vorverarbeitungsfunktion bei Aktivierung der Modelloption *ia*
$d_{km}^{fwd\prime} := D\left(d_{km}^{fwd}\right)$	Vorverarbeitete Form der Weiterleitungsverzögerung
$sl'_{km} := D\left(sl_{km}\right)$	Vorverarbeitete Form der Zeitschlitzlänge
$lt_{kj}^{i,sp}$	Ideale Latenz des kürzesten Pfades von Stream k zu Empfänger $j \in f_k.dsts$ (siehe §4.2.1)
$lt_{ijs}^{i,sp\prime}$	Ideale Latenz des kürzesten Pfades von Knoten i zu j für Framegröße s (siehe §7.1.5)

A.2 Ergebnisse/Entscheidungen des Schedulers

A.2.1 Entscheidungsvariablen

$\mathfrak{p}_{km} \in \{0,1\}$	$\mathfrak{p}_{km} \coloneqq 1$ bedeutet, dass Stream k über Link m geroutet wird
$\mathfrak{o}_{km} \in \begin{cases}\{0\} & m_0 = f_k.src \\ \mathbb{N}^0 & \text{sonst}\end{cases}$	Komponente des Zeitschlitzanfangs $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$ von Stream k auf Link m
$\mathfrak{t}_{km} \in \{x \in \mathbb{N}^0 \mid x \leq f_k.ct - 1\}$	Komponente des Zeitschlitzanfangs $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$ von Stream k auf Link m; beeinflusst durch Modelloption *mod* (siehe §7.1.3)
$\mathfrak{a}_{klmxy} \in \{0,1\}$	$\mathfrak{a}_{klmxy} \coloneqq 1$ bedeutet, dass auf Link m die x-te Zeitschlitzwiederholung von Stream k vor der y-ten Zeitschlitzwiederholung von Stream l geplant ist

A.2.2 Lineare Ausdrücke

$\boldsymbol{t}^{\boldsymbol{abs}}_{km} \coloneqq \mathfrak{o}_{km} \cdot f_k.ct + \mathfrak{t}_{km}$	Zeitschlitzanfang von Stream k auf Link m; gleiche Semantik, aber abweichende Definition bei Modelloptionen *mod*/*modv* (siehe §7.1.3)
$\boldsymbol{t}^{\boldsymbol{mod}}_{km} \coloneqq \mathfrak{t}_{km}$	Projektion des Zeitschlitzanfangs von Stream k auf Link m in das Sendeintervall $f_k.ct$; gleiche Semantik, aber abweichende Definition bei Modelloption *mod* (siehe §7.1.3)
$\boldsymbol{lt}_{kj} \coloneqq \sum_{m \in E^{\leftarrow}_{kj}} \left(\boldsymbol{t}^{\boldsymbol{abs}}_{km} + d^{rcv}_{km}\right) - \sum_{m \in E^{\rightarrow}_{kf_k.src}} \boldsymbol{t}^{\boldsymbol{abs}}_{km}$	E2E-Latenz des Streams k zum Empfänger $j \in f_k.dsts$; nur korrekt unter der Annahme, dass beide Summenformeln aufgrund der verfügbaren Links an Sender/Empfänger nur ein Element enthalten

A.3 Testcaseparameter der Multicastszenarien

Die Parameterliste zur Erzeugung der Testcases für den Multicast-Datensatz ist in Tabelle A.1 gezeigt. Die Parameterliste des Unicast-Datensatzes ist in Kapitel 4.2.3, Tabelle 4.6 gezeigt.

TC-L Load	$((fattree), (16), (54, 70), (76\,\mu s, 84\,\mu s, 100\,\mu s, 124\,\mu s), (1500\,B), (6)),$ $((ring), (8), (46, 62), (124\,\mu s, 156\,\mu s, 196\,\mu s, 244\,\mu s), (1500\,B), (6)),$ $((mesh), (9), (44, 60), (100\,\mu s, 124\,\mu s, 156\,\mu s, 196\,\mu s), (1500\,B), (6))$
TC-LL Load-Latency	$((fattree), (16), (70), (76\,\mu s, 84\,\mu s, 100\,\mu s, 124\,\mu s), (1500\,B), (1.5, 3, 6)),$ $((ring), (8), (62), (124\,\mu s, 156\,\mu s, 196\,\mu s, 244\,\mu s), (1500\,B), (1.5, 3, 6)),$ $((mesh), (9), (60), (100\,\mu s, 124\,\mu s, 156\,\mu s, 196\,\mu s), (1500\,B), (1.5, 3, 6))$
TC-G Granularity	$((fattree), (16), (54, 62, 70), (76\,\mu s), (1500\,B), (6)),$ $((ring), (8), (46, 54, 62), (124\,\mu s), (1500\,B), (6)),$ $((mesh), (9), (44, 52, 60), (100\,\mu s), (1500\,B), (6)),$ $((fattree), (16), (67, 77, 87), (76\,\mu s), (1200\,B), (6)),$ $((ring), (8), (57, 67, 77), (124\,\mu s), (1200\,B), (6)),$ $((mesh), (9), (55, 65, 75), (100\,\mu s), (1200\,B), (6))$
TC-TS Topology Size	$((fattree), (16, 54), (110), (400\,\mu s), (100\,B), (6)),$ $((ring), (12, 24, 48, 96), (70), (400\,\mu s), (100\,B), (6)),$ $((mesh), (12, 25, 47, 95), (70), (400\,\mu s), (100\,B), (6))$
TC-SSS Stream Set Size	$((fattree), (16), (80, 95, 110), (400\,\mu s), (100\,B), (6)),$ $((ring), (24), (70, 85, 100), (400\,\mu s), (100\,B), (6)),$ $((mesh), (25), (70, 85, 100), (400\,\mu s), (100\,B), (6))$
TC-DS Design Space	$((fattree), (16), (80), (160\,\mu s, 200\,\mu s, 280\,\mu s, 400\,\mu s, 640\,\mu s), (100\,B), (6)),$ $((ring), (24), (70), (160\,\mu s, 200\,\mu s, 280\,\mu s, 400\,\mu s, 640\,\mu s), (100\,B), (6)),$ $((mesh), (25), (70), (160\,\mu s, 200\,\mu s, 280\,\mu s, 400\,\mu s, 640\,\mu s), (100\,B), (6)),$ $((fattree), (16), (80), (400\,\mu s), (100\,B), (1.5, 3, 6)),$ $((ring), (24), (70), (400\,\mu s), (100\,B), (1.5, 3, 6)),$ $((mesh), (25), (70), (400\,\mu s), (100\,B), (1.5, 3, 6))$

Tabelle A.1: Parameterliste der Multicast-Testcases. Bei der Erzeugung der Testcases wird aus jedem Tupel das kartesische Produkt gebildet, in dem jedes Einzelelement eine Parameterkombination $sg := (ns, |\mathcal{V}^{es}|, |F|, ss^{ct}, ss^{fs}, ss^{lf})$ für den Szenariogenerator darstellt (vgl. Kapitel 4.2.2.1).

Anhang B

Liste aller Basismodelle, Modelloptionen und Optimierungsziele

B.1 Basismodelle

UC	Initiales Unicastmodell, bestehend aus: (6.1), (6.2), (6.3), (6.4), (6.5), (6.6), (6.7), (6.8)	§6.4.2
MCI	Initiales Multicastmodell, bestehend aus: (6.6), (6.7), (6.8), (6.9), (6.10), Alg. 6.1	§6.5.1
MCT	Optimiertes Multicastmodell mit Pfadscheduling in Baumstruktur, bestehend aus: (6.6), (6.7), (6.8), (6.9), (6.10), Alg. 6.2	§6.5.2
MCS	Optimiertes Multicastmodell mit summenbasiertem Pfadscheduling, bestehend aus: (6.4), (6.5), (6.6), (6.7), (6.8), (6.9), (6.10), (7.1), (7.2)	§7.1.1

B.2 Optimierungsziele

LT	Minimierung der Streamlatenzen.	§6.4.3
SP	Minimierung der Anzahl genutzter Kanten, was für Unicaststreams in homogenen Netzwerken dem Suchen kürzester Pfade entspricht.	§6.4.3
SPLT	Hierarchische Optimierung bzgl. *SP* und *LT*.	§6.4.3
NONE	ILP-Modell ohne Optimierungsziel.	§6.4.3
<>25	Das Suffix kennzeichnet die konfigurierte Optimality Gap, welche zum Abbruch der Optimierung beim Erreichen dieser Gap führt (hier 25%).	§6.4.3
1<>	Das Prefix kennzeichnet die Abbruchbedingung beim Finden der ersten gültigen Lösung. Im Gegensatz zu *NONE* kommt dabei dennoch ein Optimierungsziel zum Einsatz, wie beispielsweise in *1SP*.	§6.4.5

B.3 Modelloptionen

B.3.1 Routenvorverarbeitung (heuristisch)

bpl	*bounded path length* Einschränkung der wählbaren Pfade eines Streams durch Begrenzung der zulässigen Pfadlänge (d.h. Kantenanzahl) mittels absolutem Cutoff, sowie Cutoff relativ zum kürzesten Pfad.	§6.4.4.1
rpp-lbsp	*route preprocessing: load balanced + shortest path* Routenvorverarbeitung, die die ILP-basierten Routings *r-nwsp-lu1-lta* und *r-lb-nwsp-lta* berechnet und die streamspezifischen Mengen V_k und E_k durch Vereinigung der beiden ermitteln Pfade für Stream k bildet.	§7.1.5.1
rpp-ksp	*route preprocessing: k shortest paths* Routenvorverarbeitung, die die streamspezifischen Mengen V_k und E_k eines Streams k auf Basis eines kSP-Algorithmus mit Cutoffs bildet.	§7.1.5.2
psel-pa	*path selection: path activation* Ersetzt die regulären Routingbedingungen des ILPs durch Routingbedingungen und zusätzliche Binärvariablen, welche statt des konstruktiven Ansatzes durch Weiterleitungsbedingungen vollständige Pfade von Sender zu Empfänger modellieren und auswählen lassen.	§7.1.5.3
psel-la	*path selection: link activation* Alternative Modellierung des Prinzips der Pfadselektion durch Binärvariablen.	§7.1.5.3
r-nwsp-lu1	*routing: no-wait shortest path, link utilization < 1* Festes Routing (SRaS), das mittels ILP berechnet und hinsichtlich bestmöglicher Streamlatenzen optimiert wird. Die erzielbare Streamlatenz eines Pfades berücksichtigt dabei Bridge- und Linkverzögerungen, jedoch kein Queuing durch Ressourcenkonflikte. Dabei Vermeidung von Linküberlastung.	§6.4.6
r-nwsp	*routing: no-wait shortest path* Festes Routing (SRaS) entsprechend *r-nwsp-lu1*, aber ohne Überprüfung der Linkauslastung.	§6.4.6
r-lb-nwsp	*routing: load balanced, no-wait shortest path* Festes Routing (SRaS), das mittels ILP berechnet und zuerst hinsichtlich maximaler Linkauslastung der Topologie optimiert wird, bevor als sekundäres Optimierungsziel Pfade mit geringen Streamlatenzen gesucht werden.	§6.4.6
r-nwsp-lu1-lta	*routing: no-wait shortest path, link utilization < 1, latency-aware* Festes Routing (SRaS) entsprechend *r-nwsp-lu1*, mit zusätzlicher Überprüfung, dass der gewählte Pfad eines Streams k unter idealen Bedingungen (kein Queuing) die Latenzschranke $f_k.ml$ einhält.	§6.4.6
r-lb-nwsp-lta	*routing: load balanced, no-wait shortest path, latency-aware* Festes Routing (SRaS) entsprechend *r-lb-nwsp*, mit zusätzlicher Überprüfung, dass der gewählte Pfad eines Streams k unter idealen Bedingungen (kein Queuing) die Latenzschranke $f_k.ml$ einhält.	§6.4.6

B.3.2 Exakte Modelloptimierung

ia	*integer alignment* Aufrunden der Zeitschlitzlängen sl'_{km} und Weiterleitungsverzögerungen $d^{fwd\prime}_{km}$ auf Ganzzahlwerte.	§6.4.4.2
rcr	*resource constraint reduction* Reduktion von Ressourcenbedingungen (6.7)/(6.8) durch Entfernen von Redundanzen und Auflösung vorab entscheidbarer Zeitschlitzreihenfolgen.	§6.4.4.3
rlp	*routing loop prevention* Nur relevant für Multicastmodelle. Schleifenvermeidung zwischen Nachbarknoten mittels Routingbedingungen statt Schedulingbedingungen. Ersetzt (6.10) durch (6.11) und hat Auswirkungen auf Algorithmus 6.1/6.2.	§6.5.1.1
isp	*ideal shortest path* Nur relevant für *MCI*. Setzt mittels (6.12) absolute Sendezeitpunkte $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$ aller Links in E_k auf mindestens die ideale Latenz des kürzesten Pfades ab dem Sender.	§6.5.1.2
ne	*no extra links* Nur relevant für Multicastmodelle. Verhindert mittels (6.13) und (6.14), dass Sackgassen und redundante Pfade in zurückgelieferten Lösungen enthalten sind.	§6.5.3
lu1	*link utilization < 1* Stellt mittels (6.15) sicher, dass die Linkauslastung aller Links unter 100% liegt, ohne hierbei Abhängigkeiten zu Schedulingvariablen zu besitzen.	§6.5.3
o1	*option summary* Abkürzung für *bpl,rcr,ia,lu1* für *UC* bzw. *bpl,rcr,ia,lu1,ne,rlp* für Multicastmodelle.	§6.5.5
mod *modv*	*Alternative Modellierung der Sendezeitpunkte* Modellierung der absoluten Sendezeitpunkte $\boldsymbol{t}^{\boldsymbol{abs}}_{km}$ durch eine einzige Entscheidungsvariable anstelle eines linearen Ausdrucks.	§7.1.3

B.3.3 Funktionserweiterung/Schedule-Einschränkungen

bq *iq*	*binary queue modeling* *integer queue modeling* Realisiert TSN-konforme Frameisolation und Queuezuweisungen, wobei letztere durch Binärvariablen (*bq*) bzw. Ganzzahlvariablen (*iq*) modelliert werden.	§7.2.2.2
nw	*no-wait* Nur nutzbar für *UC*. Erlaubt Scheduling entlang eines Pfades nur in Form von sofortiger Weiterleitung ohne Queuing (No-Wait).	§10.2.2
cc	*cycle crossing* Nur nutzbar für *UC*/*MCS*. Schränkt beim Scheduling entlang eines Pfades ein, inwiefern sich dieses über mehrere Sendeintervalle $f_k.ct$ erstrecken darf.	§10.2.2

B.4 Dimensionierung der M-Konstanten

Im Folgenden werden alle bisher nicht näher spezifizierten M-Konstanten definiert. Die Dimensionierung wird in Kapitel 9.2 erläutert. Der Index der jeweiligen M-Konstante entspricht immer der Bedingung, in der die Konstante genutzt wird. Alle in der Definition vorkommenden Ausdrücke und Indizes (x, y, k, m, usw.) sind stets im Kontext der jeweiligen Bedingung zu verstehen, so wie wenn die hier gezeigte Definition der jeweiligen M-Konstante diese in der entsprechenden Gleichung ersetzen würde.

$$M_{6.4} := \left(\lceil f_k.ml / f_k.ct \rceil + 1\right) \cdot f_k.ct$$

$$M_{6.7} := sl'_{km} + (x+1) \cdot f_k.ct$$

$$M_{6.8} := sl'_{lm} + (y+1) \cdot f_l.ct$$

$$M_{6.10} := |E^{\rightarrow}_{ki}|$$

$$M_{DPS} := \begin{cases} f_k.ct + f_k.ml + d^{fwd\prime}_{km^{\leftarrow}} & \text{Alg. 6.1} \\ f_k.ct + \max\left(f_k.ml, lt^{i,sp\prime\prime}_{km^{\leftarrow}}\right) + d^{fwd\prime}_{km^{\leftarrow}} & \text{Alg. 6.1 mit Modelloption } \textit{isp} \\ f_k.ct + f_k.ml + delay & \text{Alg. 6.2} \end{cases}$$

$$M_{7.1} := \max_{m \in E^{\leftarrow}_{ki}} \left(f_k.ct + f_k.ml + d^{fwd\prime}_{km}\right)$$

$$M_{7.11} = M_{7.13} := \lceil sl_{km} \rceil + (x+1) \cdot f_k.ct$$

$$M_{7.12} = M_{7.14} := \lceil sl_{lm} \rceil + (y+1) \cdot f_l.ct$$

Anhang C

Weitere Algorithmen und Modelle

C.1 Gewichtung des Optimierungsziels *SP*

Gleichung (C.1) zeigt eine angepasste Form des Optimierungsziels *SP*, die auch in inhomogenen Netzwerken zu einer Optimierung der Netzwerkauslastung führt.

$$\text{Minimize} \sum_{k \in F} \sum_{m \in E_k} \mathfrak{p}_{km} \cdot \frac{sl'_{km}}{f_k.ct} \tag{C.1}$$

C.2 ILP-Modelle der SRaS-Routingverfahren

Das ILP-basierte Routing der zweischrittigen SRaS-Lösungsverfahren ist angelehnt an die multicastfähigen JRaS-Modelle einschließlich verschiedener Modelloptionen und weist daher große Überschneidungen auf. Für die ausführliche Beschreibung solcher Modellbestandteile sei hier daher auf Kapitel 6 verwiesen. Die wiederverwendeten Routingbedingungen können folgendermaßen zusammengefasst werden:

- (6.9) und (6.11) initieren das Routing eines Streams am Empfänger und vervollständigen die Route durch Hinzufügen von Vorgängern bis hin zum Sender.
- (6.13) und (6.14) schließen redundante Pfade und Sackgassen aus der Lösung aus.
- (6.15) verhindert, dass eine Linkauslastung von mehr als 100 % akzeptiert wird.

Da die hier diskutierten Modelle ausschließlich dem Routing dienen, sind keine der bekannten Schedulingbedingungen enthalten. Problematisch daran ist, dass die genannten Routingbedingungen es zulassen, dass Schleifen gebildet werden und der Sender des Streams unverbunden bleibt (siehe auch Kapitel 6.5.1.1). Während diese fehlerhaften Routen in den multicastfähigen JRaS-Modellen durch die Schedulingbedingungen verhindert werden, wird hier stattdessen eine einfache kontinuierliche (d.h. nicht auf ganzzahlige Lösungen beschränkte) Zählvariable $0 \leq \mathfrak{r}_{km}$ eingeführt und durch Bedingung (C.2) beschränkt.

$$\forall k \in F, \forall i \in V_k, \forall \vec{m} \in E^{\rightarrow}_{ki}, \forall \overleftarrow{m} \in E^{\leftarrow}_{ki} \qquad \mathfrak{r}_{km^{\rightarrow}} - \mathfrak{r}_{km^{\leftarrow}} \geq 1 - M_{C.2} \cdot (1 - \mathfrak{p}_{km^{\rightarrow}}) \tag{C.2}$$

Ähnlich zu den Schedulingbedingungen wird durch (C.2) erzielt, dass die Zählvariable $\mathfrak{r}_{km\overrightarrow{i}}$ für jeden aktiven Ausgangslink eines Knotens i größer sein muss als die Zählvariable eines jeden Eingangslinks. Da das Aufwärtszählen auf einer Schleife nicht erfüllbar ist, werden Schleifen im Routing hierdurch vermieden. Die weiteren Anpassungen für die verschiedenen, in Kapitel 6.4.6 genannten Varianten werden im Folgenden erläutert.

Ein Unterscheidungsmerkmal der verschiedenen SRaS-Varianten ist das Optimierungsziel des zuvor diskutierten Modells. Das in (C.3) gezeigte Ziel dient dabei der Bevorzugung von Pfaden, die im Idealfall geringe Streamlatenzen ermöglichen. Diese werden unter Berücksichtigung der Topologie- und Streamspezifikationen, jedoch ohne Queueing ermittelt. Dabei kommt bewusst ein anderes Ziel zum Einsatz als im Optimierungsziel *LT* des Scheduling-ILPs, da ein derartiges Optimierungsziel hier zu ähnlichen Performanceproblemen wie bereits für MCI_{LT} führen würde. Dabei wird in Kauf genommen, dass es sich hier für Multicaststreams nicht um effektive Latenzoptimierung handelt, da ggf. Pfade mit einem möglichst späten Aufsplitten zwecks Multicast gegenüber Pfaden mit geringeren E2E-Latenzen zu jedem Empfänger bevorzugt werden.

$$\text{Minimize} \sum_{k \in F} \left(\sum_{m \in E_k | m_1 \notin f_k.dsts} \mathfrak{p}_{km} \cdot d_{km}^{fwd\prime} + \sum_{m \in E_k | m_1 \in f_k.dsts} \mathfrak{p}_{km} \cdot d_{km}^{rcv} \right) \tag{C.3}$$

Das zweite Optimierungsziel dient dem Load Balancing und erfordert eine zusätzliche kontinuierliche Variable $0 \leq \mathfrak{u} \leq luf$ und die Bedingung (C.4) zur Erfassung der höchsten Linkauslastung der Topologie.

$$\forall m \in \mathcal{E} \qquad \sum_{k \in F | m \in E_k} luf \cdot \frac{sl'_{km}}{f_k.ct} \cdot \mathfrak{p}_{km} \;\leq\; \mathfrak{u} \tag{C.4}$$

Durch (C.4) wird zunächst nur sichergestellt, dass $\mathfrak{u}$ einen beliebigen Wert enthält, der größer als jede Linkauslastung der Topologie ist. Durch die einfache Zielfunktion (C.5) kann das Routing dann bezüglich der maximal auftretenden Linkauslastung optimiert werden.

$$\text{Minimize} \quad \mathfrak{u} \tag{C.5}$$

Basierend auf den bisher gezeigten Gleichungen kann beschrieben werden wie die ersten drei, nicht latenzsensitiven Modelloptionen aus Kapitel 6.4.6 insgesamt zusammengesetzt sind.

(a) Alle drei Modelloptionen nutzen (6.9), (6.11), (6.13), (6.14), (C.2) und eine relative Optimality Gap von 10 % bei einem Zeitlimit von 600 s

(b) *r-nwsp* nutzt zusätzlich zu (a) das Optimierungsziel (C.3)

(c) *r-nwsp-lu1* nutzt zusätzlich zu (a) die Bedingung (6.15) und das Optimierungsziel (C.3)

(d) *r-lb-nwsp* nutzt zusätzlich zu (a) die Bedingungen (6.15), (C.4) und ein hierarchisches Optimierungsziel mit primärer Zielfunktion (C.5) und zweiter Zielfunktion (C.3), sowie die zusätzliche *absolute* Optimality Gap $0.01\,luf$ für das primäre Ziel (C.5) (d.h., Abbruch der Optimierung wenn die höchste Linkauslastung weniger als 1 % über dem Optimalwert liegt)

Die latenzsensitiven Modelloptionen *r-nwsp-lu1-lta*/*r-lb-nwsp-lta* unterscheiden sich gegenüber den jeweiligen Optionen ohne das Suffix **-lta* durch das Ersetzen von (C.2) durch (C.6) und das Hinzufügen von (C.7).

$$\forall k \in F, \forall i \in V_k, \forall \vec{m} \in E_{ki}^{\rightarrow}, \forall \overleftarrow{m} \in E_{ki}^{\leftarrow} \qquad \mathfrak{r}_{k\vec{m}} \;-\; \mathfrak{r}_{k\overleftarrow{m}} \;\geq\; d_{k\overleftarrow{m}}^{fwd\prime} \;-\; M_{C.6}\cdot(1-\mathfrak{p}_{k\vec{m}}) \tag{C.6}$$

$$\forall k \in F, \forall j \in f_k.dsts \qquad \sum_{m \in E_{kj}^{\leftarrow}} \left(\mathfrak{r}_{km} + d_{km}^{rcv}\right) \;-\; \sum_{m \in E_{kf_k.src}^{\rightarrow}} \mathfrak{r}_{km} \;\leq\; f_k.ml \tag{C.7}$$

Wie schon in anderen Teilen des ILP-Modells wird angenommen, dass es nur einen Eingangslink zu Empfängern bzw. einen Ausgangslink aus dem Sender gibt, sodass die Summen in (C.7) jeweils nur ein Element enthalten. Nach (C.6) bildet $\mathfrak{r}_{km}$ (vergleichbar mit $\boldsymbol{t}_{km}^{\boldsymbol{abs}}$ des Scheduling-ILPs) die Sendezeitpunkte entlang einer Route ab, wobei diese im hier diskutierten Routingmodell ohne Berücksichtigung von Zeitschlitzkonflikten festgelegt werden. Die Bedingung (C.7) stellt wiederum sicher, dass diese ideale Latenz der gewählten Route die Streamanforderungen erfüllt.

C.3 Berechnung idealer Latenzen zu jedem Link

Algorithmus C.1 zeigt eine Möglichkeit zur Bestimmung der idealen Streamlatenzen vom Sender zu jedem nutzbaren Link des Streams. Dabei wird ausgeschlossen, dass ein Link genutzt wird, wenn der Vorgängerlink die Gegenrichtung des Links selbst ist (Z. 10, Vermeidung von Schleifen zwischen Nachbarknoten). Der Algorithmus initialisiert dabei $lt_{km}^{i,sp\prime\prime}$ für alle relevanten Links m. Genutzt werden die Ergebnisse in der Modelloption *isp* des Basismodells *MCI* (Kapitel 6.5.1.2). Der Algorithmus weist große Überschneidungen zu Algorithmus 6.2 auf, wobei hier nur die Latenzen ermittelt

```
1  foreach k ∈ F do
2      i := f_k.src
3      visited := E_ki^→                                  // initialize set of visited links
4      next := ((m, d_km^fwd′))_{m∈E_ki^→}                // a sequence of (link, delay) tuples
5      while next ≠ () do
6          sort next by delay (2nd field of each element)
7          m←, delay := pop_front(next)                   // remove and return first element of next
8          i := m←_1
9          foreach m→ ∈ E_ki^→ do                         // iterate successor links of m←
10             if not (m→ ∈ visited or m←_0 = m→_1) then
11                 lt_km→^{i,sp″} := delay
12                 visited := visited ∪ {m→}
13                 append (m→, delay + d_km→^fwd′) to next
```

Algorithmus C.1: Berechnung der idealen Latenzen vom Sender zu jedem nutzbaren Link, unter Vermeidung von Schleifen zwischen Nachbarknoten.

werden sollen, während der gleiche Ablauf in Algorithmus 6.2 zur Bildung von ILP-Bedingungen zum Pfadscheduling genutzt wird.

Literatur

Allgemein

[1] PROFIBUS Nutzerorganisation e. V. (PNO). *PROFINET - The Solution Platform for Process Automation*. URL: https://www.profibus.com/index.php?eID=dumpFile&t=r&r=55896&dl=1&token=79179e49c534c91591945872c5f102a5789ca63a (besucht am 16. 08. 2024).

[2] Zdeněk Hanzálek, Pavel Burget und Přemysl Šůcha. „Profinet IO IRT Message Scheduling With Temporal Constraints“. In: *IEEE Transactions on Industrial Informatics* 6.3 (2010), S. 369–380. DOI: 10.1109/TII.2010.2052819.

[3] Wilfried Steiner. „An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks“. In: *2010 31st IEEE Real-Time Systems Symposium*. 2010, S. 375–384. DOI: 10.1109/RTSS.2010.25.

[4] Thomas Stüber u. a. „A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)“. In: *IEEE Access* 11 (2023), S. 61192–61233. DOI: 10.1109/ACCESS.2023.3286370.

[5] Nick Feamster, Jennifer Rexford und Ellen Zegura. „The road to SDN: an intellectual history of programmable networks“. In: *SIGCOMM Comput. Commun. Rev.* 44.2 (Apr. 2014), S. 87–98. ISSN: 0146-4833. DOI: 10.1145/2602204.2602219. URL: https://doi.org/10.1145/2602204.2602219.

[6] Nick McKeown u. a. „OpenFlow: enabling innovation in campus networks“. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (März 2008), S. 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[7] Martin Lukasiewycz u. a. „Modular scheduling of distributed heterogeneous time-triggered automotive systems“. In: *17th Asia and South Pacific Design Automation Conference*. 2012, S. 665–670. DOI: 10.1109/ASPDAC.2012.6165039.

[8] Domiţian Tămaş-Selicean, Paul Pop und Wilfried Steiner. „Design optimization of TTEthernet-based distributed real-time systems“. In: *Springer Real-Time Systems* 51 (2015), S. 1–35. DOI: 10.1007/s11241-014-9214-8.

[9] Naresh Ganesh Nayak, Frank Dürr und Kurt Rothermel. „Time-sensitive Software-defined Network (TSSDN) for Real-time Applications“. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 2016, S. 193–202. ISBN: 9781450347877. DOI: 10.1145/2997465.2997487. URL: https://doi.org/10.1145/2997465.2997487.

[10] Fedor Smirnov u. a. „Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks“. In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2017, S. 1–6. DOI: 10.1145/3061639.3062298.

[11] *Website der Plattform Industrie 4.0*. URL: https://www.plattform-i40.de/ (besucht am 07. 04. 2024).

[12] Maryam Pahlevan und Roman Obermaisser. „Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks“. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Bd. 1. 2018, S. 337–344. DOI: 10.1109/ETFA.2018.8502515.

[13] David Hellmanns u. a. „How to Optimize Joint Routing and Scheduling Models for TSN Using Integer Linear Programming“. In: *Proceedings of the 29th International Conference on Real-Time Networks and Systems*. RTNS '21. NANTES, France: Association for Computing Machinery, 2021, S. 100–111. ISBN: 9781450390019. DOI: 10.1145/3453417.3453421. URL: https://doi.org/10.1145/3453417.3453421.

[14] Qinghan Yu und Ming Gu. „Adaptive Group Routing and Scheduling in Multicast Time-Sensitive Networks“. In: *IEEE Access* 8 (2020), S. 37855–37865. DOI: 10.1109/ACCESS.2020.2974580.

[15] Amaury Van Bemten und Wolfgang Kellerer. *Network Calculus: A Comprehensive Guide*. Techn. Ber. Okt. 2016. DOI: 10.13140/RG.2.2.32305.89448.

[16] Jochen W. Guck und Wolfgang Kellerer. „Achieving end-to-end real-time Quality of Service with Software Defined Networking“. In: *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. 2014, S. 70–76. DOI: 10.1109/CloudNet.2014.6968971.

[17] Luxi Zhao u. a. „Timing Analysis of AVB Traffic in TSN Networks Using Network Calculus“. In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2018, S. 25–36. DOI: 10.1109/RTAS.2018.00009.

[18] Astrit Ademaj und TTTech Computertechnik AG. *Time-Triggered Ethernet - A Powerful Network Solution for Multiple Purpose*. URL: https://www.tttech.com/sites/default/files/documents/TTEthernet_Technical-Whitepaper_TTTech.pdf (besucht am 03. 06. 2024).

[19] PI North America. *PROFINET IRT - The Solution for Synchronous Real-time Applications*. URL: https://us.profinet.com/wp-content/uploads/2020/10/PROFINET-IRT-final.pdf (besucht am 03. 06. 2024).

[20] Dip Goswami u. a. „Time-triggered implementations of mixed-criticality automotive software“. In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, S. 1227–1232. DOI: 10.1109/DATE.2012.6176680.

[21] Francisco Pozo, Guillermo Rodriguez-Navas und Hans Hansson. „Schedule Reparability: Enhancing Time-Triggered Network Recovery Upon Link Failures“. In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2018, S. 147–156. DOI: 10.1109/RTCSA.2018.00026.

[22] Gaetano Patti, Lucia Lo Bello und Luca Leonardi. „Deadline-Aware Online Scheduling of TSN Flows for Automotive Applications“. In: *IEEE Transactions on Industrial Informatics* 19.4 (2023), S. 5774–5784. DOI: 10.1109/TII.2022.3184069.

[23] Junli Xue u. a. „Time-Aware Traffic Scheduling with Virtual Queues in Time-Sensitive Networking“. In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2021, S. 604–607.

[24] Yanzhou Zhang u. a. „Scalable No-wait Scheduling with Flow-aware Model Conversion in Time-Sensitive Networking“. In: *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*. 2022, S. 413–418. DOI: 10.1109/GLOBECOM48099.2022.10001004.

[25] Yanzhou Zhang u. a. „Efficient Flow Scheduling for Industrial Time-Sensitive Networking: A Divisibility Theory-Based Method“. In: *IEEE Transactions on Industrial Informatics* 18.12 (2022), S. 9312–9323. DOI: 10.1109/TII.2022.3151810.

[26] Xi Jin u. a. „Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries“. In: *IEEE Access* 8 (2020), S. 6751–6767. DOI: 10.1109/ACCESS.2020.2964690.

[27] Silviu S. Craciunas u. a. „Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks“. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 2016, S. 183–192. ISBN: 9781450347877. DOI: 10.1145/2997465.2997470. URL: https://doi.org/10.1145/2997465.2997470.

[28] Abdullah Alnajim, Seyedmohammad Salehi und Chien-Chung Shen. „Incremental Path-Selection and Scheduling for Time-Sensitive Networks“. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, S. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013427.

[29] Ammad Ali Syed u. a. „Dynamic Scheduling and Routing for TSN based In-vehicle Networks“. In: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2021, S. 1–6. DOI: 10.1109/ICCWorkshops50388.2021.9473810.

[30] Ammad Ali Syed u. a. „Fault-Tolerant Dynamic Scheduling and Routing for TSN based In-vehicle Networks“. In: *2021 IEEE Vehicular Networking Conference (VNC)*. 2021, S. 72–75. DOI: 10.1109/VNC52810.2021.9644662.

[31] Wilfried Steiner, Silviu S. Craciunas und Ramon Serna Oliver. „Traffic Planning for Time-Sensitive Communication“. In: *IEEE Communications Standards Magazine* 2.2 (2018), S. 42–47. DOI: 10.1109/MCOMSTD.2018.1700055.

[32] Niklas Reusch u. a. „Window-Based Schedule Synthesis for Industrial IEEE 802.1Qbv TSN Networks“. In: *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. 2020, S. 1–4. DOI: 10.1109/WFCS47810.2020.9114414.

[33] Ramon Serna Oliver, Silviu S. Craciunas und Wilfried Steiner. „IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding“. In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2018, S. 13–24. DOI: 10.1109/RTAS.2018.00008.

[34] Paul Pop u. a. „Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks“. In: *IET Cyber-Physical Systems: Theory & Applications* 1.1 (2016), S. 86–94. DOI: https://doi.org/10.1049/iet-cps.2016.0021. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cps.2016.0021. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cps.2016.0021.

[35] Ayman A. Atallah, Ghaith Bany Hamad und Otmane Ait Mohamed. „Routing and Scheduling of Time-Triggered Traffic in Time-Sensitive Networks“. In: *IEEE Transactions on Industrial Informatics* 16.7 (2020), S. 4525–4534. DOI: 10.1109/TII.2019.2950887.

[36] Naresh Ganesh Nayak, Frank Dürr und Kurt Rothermel. „Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks“. In: *IEEE Transactions on Industrial Informatics* 14.5 (2018), S. 2066–2075. DOI: 10.1109/TII.2017.2782235.

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

[38] Jonathan Falk, Frank Dürr und Kurt Rothermel. „Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP“. In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2018, S. 136–146. DOI: 10.1109/RTCSA.2018.00025.

[39] Jonathan Falk u. a. „Optimal routing and scheduling of complemental flows in converged networks“. In: *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. RTNS '19. Toulouse, France: Association for Computing Machinery, 2019, S. 154–164. ISBN: 9781450372237. DOI: 10.1145/3356401.3356415. URL: https://doi.org/10.1145/3356401.3356415.

[40] Jonathan Falk, Frank Dürr und Kurt Rothermel. „Time-Triggered Traffic Planning for Data Networks with Conflict Graphs“. In: *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2020, S. 124–136. DOI: 10.1109/RTAS48715.2020.00-12.

[41] Yudong Huang u. a. „Online Routing and Scheduling for Time-Sensitive Networks“. In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. 2021, S. 272–281. DOI: 10.1109/ICDCS51616.2021.00034.

[42] Cong Li u. a. „Joint Routing and Scheduling for Dynamic Applications in Multicast Time-Sensitive Networks“. In: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2021, S. 1–6. DOI: 10.1109/ICCWorkshops50388.2021.9473540.

[43] Ammad Ali Syed u. a. „MIP-based Joint Scheduling and Routing with Load Balancing for TSN based In-vehicle Networks“. In: *2020 IEEE Vehicular Networking Conference (VNC)*. 2020, S. 1–7. DOI: 10.1109/VNC51378.2020.9318350.

[44] Michael Lander Raagaard und Paul Pop. *Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN)*. Techn. Ber. Jan. 2017.

[45] Chuanyu Xue u. a. „Real-Time Scheduling for 802.1Qbv Time-Sensitive Networking (TSN): A Systematic Review and Experimental Study“. In: *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2024, S. 108–121. DOI: 10.1109/RTAS61025.2024.00017.

[46] Mohammad Al-Fares, Alexander Loukissas und Amin Vahdat. „A scalable, commodity data center network architecture“. In: *SIGCOMM Comput. Commun. Rev.* 38.4 (Aug. 2008), S. 63–74. ISSN: 0146-4833. DOI: 10.1145/1402946.1402967. URL: https://doi.org/10.1145/1402946.1402967.

[47] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. International Series in Operations Research & Management Science. Springer New York, NY, 2014. ISBN: 9781461476306. DOI: 10.1007/978-1-4614-7630-6. URL: https://doi.org/10.1007/978-1-4614-7630-6.

[48] Gurobi Optimization, LLC. *Mixed-Integer Programming (MIP) – A Primer on the Basics*. URL: https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/ (besucht am 12. 08. 2024).

[49] Marek Vlk, Zdeněk Hanzálek und Siyu Tang. „Constraint programming approaches to joint routing and scheduling in time-sensitive networks“. In: *Computers and Industrial Engineering* 157 (2021), S. 107317. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2021.107317. URL: https://www.sciencedirect.com/science/article/pii/S0360835221002217.

[50] Marian Ulbricht u. a. „TSN-FlexTest: Flexible TSN Measurement Testbed“. In: *IEEE Transactions on Network and Service Management* 21.2 (2024), S. 1387–1402. DOI: 10.1109/TNSM.2023.3327108.

[51] Michael Albus. „Strategien zur Ableitung zulässiger TSN-Konfigurationen“. Bachelor's Thesis. University of Rostock, 2023.

[52] Jin Yen. „Finding the K Shortest Loopless Paths in a Network“. In: *Management Science* 17 (11 1971), S. 712–716. DOI: https://doi.org/10.1287/mnsc.17.11.712.

[53] Aellison Cassimiro T. dos Santos, Ben Schneider und Vivek Nigam. „TSNSCHED: Automated Schedule Generation for Time Sensitive Networking“. In: *2019 Formal Methods in Computer Aided Design (FMCAD)*. 2019, S. 69–77. DOI: 10.23919/FMCAD.2019.8894249.

[54] David Hellmanns u. a. „Scaling TSN Scheduling for Factory Automation Networks“. In: *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. 2020, S. 1–8. DOI: 10.1109/WFCS47810.2020.9114415.

[55] Voica Gavriluţ u. a. „AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN“. In: *IEEE Access* 6 (2018), S. 75229–75243. DOI: 10.1109/ACCESS.2018.2883644.

Standarddokumente

[56] *OPC 10000-14: UA Part 14: PubSub*. URL: https://reference.opcfoundation.org/Core/Part14/v105/docs/ (besucht am 07. 04. 2024).

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

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

[59] *Website of the IEEE 802.1 Working Group*. URL: https://1.ieee802.org/ (besucht am 30. 08. 2023). Relevante Unterseiten: https://1.ieee802.org/, https://1.ieee802.org/tsn/.

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

[61] „IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks“. In: *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)* (2022), S. 1–2163. DOI: 10.1109/IEEESTD.2022.10004498.

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

[63] „IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability“. In: *IEEE Std 802.1CB-2017* (2017), S. 1–102. DOI: 10.1109/IEEESTD.2017.8091139.

[64] „IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications“. In: *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)* (2020), S. 1–421. DOI: 10.1109/IEEESTD.2020.9121845.

[65] „IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements“. In: *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)* (2018), S. 1–208. DOI: 10.1109/IEEESTD.2018.8514112.

[66] „IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic“. In: *IEEE Std 802.3br-2016 (Amendment to IEEE Std 802.3-2015 as amended by IEEE Std 802.3bw-2015, IEEE Std 802.3by-2016, IEEE Std 802.3bq-2016, and IEEE Std 802.3bp-2016)* (2016), S. 1–58. DOI: 10.1109/IEEESTD.2016.7592835.

[67] „IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption“. In: *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)* (2016), S. 1–52. DOI: 10.1109/IEEESTD.2016.7553415.

[68] „IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing“. In: *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)* (2017), S. 1–65. DOI: 10.1109/IEEESTD.2017.8064221.

[69] Rob Enns u. a. *Network Configuration Protocol (NETCONF)*. RFC 6241. Juni 2011. DOI: 10.17487/RFC6241. URL: https://www.rfc-editor.org/info/rfc6241.

[70] Andy Bierman, Martin Björklund und Kent Watsen. *RESTCONF Protocol*. RFC 8040. Jan. 2017. DOI: 10.17487/RFC8040. URL: https://www.rfc-editor.org/info/rfc8040.

[71] „Ethernet Services Attributes Phase 3“. In: *MEF Technical Specification 10.3* (2013), S. 1–120. URL: https://www.mef.net/wp-content/uploads/2013/10/MEF-10-3.pdf (besucht am 23. 05. 2024).

Softwarebibliotheken

[72] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual Version 11.0*. 2024. URL: https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/11.0/refman.pdf (besucht am 13. 08. 2024).

[73] John D. Hunter. „Matplotlib: A 2D Graphics Environment“. In: *Computing in Science and Engineering* 9.3 (2007), S. 90–95. DOI: 10.1109/MCSE.2007.55.

[74] Aric A. Hagberg, Daniel A. Schult und Pieter J. Swart. „Exploring Network Structure, Dynamics, and Function using NetworkX“. In: *Proceedings of the 7th Python in Science Conference*. Hrsg. von Gaël Varoquaux, Travis Vaught und Jarrod Millman. Pasadena, CA USA, 2008, S. 11–15.

(Mit-)Betreute studentische Arbeiten

[S1] Upoma Saha. „Heuristic Runtime Optimization of ILP-based Joint Routing and Scheduling for Time-Triggered Networks“. Masterarbeit. Universität Rostock, 2020.

[S2] Tobias Reincke. „Routing von latenzbeschränkten AVB Streams in einem TSN unter Berücksichtigung von Zeitschlitzen des zeitgesteuerten Netzwerkverkehrs“. Bachelorarbeit. Universität Rostock, 2022.

[S3] Stefan Arth. „Evaluierung von Algorithmen zur Ressourcenzuteilung in einem zeitschlitzbasierten Echtzeitkommunikationssystem“. Bachelorarbeit. Universität Rostock, 2016.

[S4] Niklas Gröne. „Entwicklung einer Automatisierungslösung zur Verringerung der Zeit eines Prozessschrittes bei der Herstellung von Stents“. Bachelorarbeit. Universität Rostock, 2020.

[S5] Leonard Sebastian Thiele. „Verwendung eines Testbeds zur Evaluation von SLMT: Einem Ansatz zur Zeitsynchronisation mittels linearer Programmierung, Multicasts und Temperaturkompensation“. Bachelorarbeit. Universität Rostock, 2020.

[S6] Upoma Saha. „Time-Sensitive Software-Defined Networking Routing and Scheduling: Implementation of Routing and Scheduling Algorithms“. Spezialisierungsmodul. Universität Rostock, 2019.

[S7] Keivan Ali Rezazad. „Simulation Model of a Real-Time Compliant Ethernet Switch in OMNeT++“. Spezialisierungsmodul. Universität Rostock, 2018.

[S8] Irfan Fachrudin Priyanta. „Development of an OPC UA based Configuration Concept for Deterministic Network“. Masterarbeit. Universität Rostock, 2020.

[S9] Humza Asghar. „Implementation and Evaluation of PTP-LP: An Approach to Use Linear Programming For Increasing Synchronization Accuracy“. Masterarbeit. Universität Rostock, 2019.

[S10] Benjamin Rother. „Automatische Konfiguration eines TSN-Netzwerks für IEEE 11073 SDC-basierte Medizingeräteverbünde“. Masterarbeit. Universität Rostock, 2019.

Eigene Arbeiten

[E1] Peter Danielis u. a. „Survey on real-time communication via ethernet in industrial automation environments“. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. 2014, S. 1–8. DOI: 10.1109/ETFA.2014.7005074.

[E2] Helge Parzyjegla u. a. „Implementing content-based publish/subscribe with OpenFlow“. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: Association for Computing Machinery, 2019, S. 1392–1395. ISBN: 9781450359337. DOI: 10.1145/3297280.3297589.

[E3] Henning Puttnies u. a. „Clock Synchronization Using Linear Programming, Multicasts, and Temperature Compensation“. In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, S. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013260.

[E4] Peter Danielis u. a. „Emulation of SDN-supported automation networks“. In: *2015 IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA)*. 2015, S. 1–8. DOI: 10.1109/ETFA.2015.7301517.

[E5] Peter Danielis u. a. „Real- Time Capable Internet Technologies for Wired Communication in the Industrial IoT-a Survey“. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Bd. 1. 2018, S. 266–273. DOI: 10.1109/ETFA.2018.8502528.

[E6] Jan Skodzik u. a. „CoHaRT: A P2P-based deterministic transmission of large data amounts using CoAP“. In: *2015 IEEE International Conference on Industrial Technology (ICIT)*. 2015, S. 1851–1856. DOI: 10.1109/ICIT.2015.7125366.

[E7] Christian Wernecke u. a. „Stitching Notification Distribution Trees for Content-based Publish/Subscribe with P4“. In: *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2020, S. 100–104. DOI: 10.1109/NFV-SDN50289.2020.9289916.

[E8] Christian Wernecke u. a. „Evaluating P4-based Virtual Delivery Trees for Content-based Publish/Subscribe“. In: *2022 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2022, S. 78–84. DOI: 10.1109/NFV-SDN56302.2022.9974746.

[E9] Christian Wernecke u. a. „Flexible Notification Forwarding for Content-Based Publish/Subscribe Using P4“. In: *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2019, S. 1–5. DOI: 10.1109/NFV-SDN47374.2019.9040048.

[E10] Jan Skodzik u. a. „PSP-Auto: A DHT-based data storage and retrieval system for automation“. In: *2015 Annual IEEE Systems Conference (SysCon) Proceedings*. 2015, S. 853–858. DOI: 10.1109/SYSCON.2015.7116857.

[E11] Eike Schweissguth u. a. „Application-aware industrial ethernet based on an SDN-supported TDMA approach“. In: *2016 IEEE World Conference on Factory Communication Systems (WFCS)*. 2016, S. 1–8. DOI: 10.1109/WFCS.2016.7496496.

[E12] Eike Schweissguth u. a. „ILP-Based Joint Routing and Scheduling for Time-Triggered Networks“. In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. RTNS '17. Grenoble, France: Association for Computing Machinery, 2017, S. 8–17. ISBN: 9781450352864. DOI: 10.1145/3139258.3139289.

[E13] Benjamin Rother u. a. „Automatic Configuration of a TSN Network for SDC-based Medical Device Networks“. In: *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. 2020, S. 1–8. DOI: 10.1109/WFCS47810.2020.9114471.

[E14] Eike Schweissguth u. a. „ILP-Based Routing and Scheduling of Multicast Realtime Traffic in Time-Sensitive Networks“. In: *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2020, S. 1–11. DOI: 10.1109/RTCSA50079.2020.9203662.

[E15] Eike Schweissguth u. a. „TSN Scheduler Benchmarking“. In: *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*. 2023, S. 1–8. DOI: 10.1109/WFCS57264.2023.10144227.

[E16] Eike Schweissguth, Stefan Mehner und David Hellmanns. *TSN Scheduler Benchmarking: Scenarios v1.0.0*. DOI: 10.5281/zenodo.7738090.

[E17] Eike Schweissguth, Stefan Mehner und David Hellmanns. *TSN Scheduler Benchmarking: Results v1.0.0*. DOI: 10.5281/zenodo.7738206.

[E18] Eike Schweissguth. *TSN Scheduler Benchmarking: Scenarios v2.0.0*. DOI: 10.5281/zenodo.13993512. URL: https://github.com/EikeSG/TSNBenchScenarios/tree/v2.0.0.

[E19] Eike Schweissguth. *TSN Scheduler Benchmarking: Results v2.0.1*. DOI: 10.5281/zenodo.13995705. URL: https://github.com/EikeSG/TSNBenchResults/tree/v2.0.1.

[E20] Eike Schweissguth u. a. „Realtime Publish/Subscribe for the Industrial Internet of Things“. In: *2. GI/ITG KuVS Fachgespräch Network Softwarization*. URL: https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/kommunikationsnetze/kuvs-fg-netsoft/2020/program/ (besucht am 22. 11. 2024).