Time-Sensitive Traffic and Time-Triggered Mechanisms: Traffic Planning and Analysis

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines Doktors
der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Jonathan Falk

aus Leonberg, Baden-Württemberg

Hauptberichter: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Mitberichter: Prof. Dr. habil. Michael Menth

Tag der mündlichen Prüfung: 29.09.2022

Universität Stuttgart

Institut für Parallele und Verteilte Systeme der Universität Stuttgart

2022

Submitted to the University of Stuttgart

Involved institutions and departments:

Institute for Parallel and Distributed Systems

Chair of Distributed Systems

Jonathan Falk
University of Stuttgart
Stuttgart, Baden-Württemberg
Deutschland

D 93 (dissertation)

“*Every time they try 2 clock u*
Tick more than they tock

— Rosie Gaines & Prince (Push)

Danksagungen

Ich danke meinem Herrn Jesus Christus, in dem Gottes Gnade offenbar wurde und durch dessen Blut ich erlöst bin, für mein Leben.

Ich danke Kurt Rothermel, der mich während meiner Zeit als Mitarbeiter und bei meinem Promotionsvorhaben an der Universität Stuttgart begleitet und unterstützt hat. Ebenso danke ich Michael Menth für seine Unterstützung bei meinem Promotionsvorhaben in der Rolle als Mitberichter.

Ich danke Frank Dürr, über dessen Schreibtisch meine Paperentwürfe wahrscheinlich stapelweise gewandert sind, für alle Hilfe und Unterstützung.

Für die fachübergreifende Zusammenarbeit im Rahmen des DFG-geförderten Projekts „Integrierte Reglerentwurfsverfahren und Kommunikationsdienste für digital vernetzte Regelungssysteme“ bedanke ich mich auch bei Frank Allgöwer, Steffen Linsenmayer, Stefan Wildhagen und Michael Hertneck von der „Regelungsseite“.

Ich bedanke mich bei den Leuten am IPVS, dabei insbesondere denjenigen aus der Abteilung für verteilte Systeme und der Nachbarabteilung für die gemeinsame Zeit beim Lehren, Forschen und dazwischen. Ich bedanke mich bei Naresh Nayak und David Hellmanns, mit denen ich jeweils nicht nur räumlich sondern auch thematisch verbunden war, und bei Sukanya Bhowmik, Ahmad Slo, Ben Carabelli, Mohamed Abdelaal, Heiko Geppert, Michael Schramm, Johannes Kässinger, Otto Bibartiu, Saravana Murthy Palanisamy, Christoph Dibak, Zohaib Riaz, Christian Mayer, Henriette Röger, Ruben Mayer, Thomas Kohler und Michael Behringer unter anderem auch für den Gedankenaustausch in vielerlei Hinsicht. Ich danke Eva Strähle, die mit den Interna des Universitätsapparats gut vertraut ist.

Ich danke meiner Familie und meinen Freunden für alle Unterstützung.

Stellvertretend für die vielen Musiker, deren Werke mich während meine Zeit an der Universität begleitet haben, danke ich den unzähligen Hip-Hop Sprechgesangskünstlern, deren lyrischen Darbietungen ich viele hilfreiche Vokabeln und Wendungen zum Verfassen englischsprachiger Texte entnehmen konnte. Weiter danke ich auch einigen stark gitarrenlastigen Musikgruppen, deren Aufnahmen teilweise meine Gefühle nach Paper-Notifications widerspiegelten.

Contents

Lists of Figures, Tables, Algorithms, and Theorems

List of Figures

List of Tables

List of Algorithms

List of Theorems

List of Acronyms

C

CPS Cyber-physical System

F

FIFO First In, First Out

G

GFH Greedy Flow Heap (Heuristic)

I

ILP Integer Linear Program(ing)
IP Internet Protocol

L

LAN Local Area Network

M

MAC Medium Access Control
MIMO Multiple Input and Multiple Output
MIVS Maximal Independent Vertex Set

N

NC Network Calculus
NCS Networked Control System

O

OMT Optimization Modulo Theories

P

PCP Priority-code Point

Q

QoS Quality of Service

S

SAT Satisfiability (Problem)
SDN Software-defined Networking
SFP Small Form-factor Pluggable
SMT Satisfiability Modulo Theories

T

TDMA Time-division Multiple Access
TSN Time-sensitive Networking
TSSDN Time-sensitive Software-defined Networking

W

WHRT Weakly-hard Real-time

Abstract/Kurzzusammenfassung

Abstract

Networked applications increasingly rely on real-time-capable communication networks as the backbone to connect individual components. Traditionally, we find such applications, for example, in industrial automation or vehicular on-board networks. As software encroaches further into infrastructures, for example, in the context of smart grids or connected driving scenarios, the demand for real-time communication increases. As a result, networks are enhanced with additional mechanisms that can be used to combine deterministic medium access with suitable scheduling, routing, and resource reservation mechanisms to guarantee worst-case bounds, for instance, on latency or delay.

Therefore, in the first part of this thesis, we address the problem of how to synthesize a traffic plan that provides real-time guarantees by design for a given network and a set of flow requests. We present approaches for different variations of the traffic planning problem for time-triggered flows and evaluate them for a wide array of synthetic scenarios using proof-of-concept implementations. Our approaches differ, for example, with respect to the degrees of freedom in time (time-discretization) and space (routing), and the method used to compute the traffic plan.

We present two different approaches for traffic planning of time-triggered flows that rely on constraint-based programming. To be exact, we express the traffic planning constraints via a set of linear-(in)equalities as integer-linear programs (ILPs) with different granularity regarding the routing and scheduling decisions. We extend these two approaches for the computation of traffic plans for complemental flows which consist of a time-triggered part and an event-triggered part. Here, the event-triggered part is considered for the optimization of the traffic plan. Following that, we present two different approaches for traffic planning of time-triggered flows that use conflict-graph-based modeling. First, we show how to iteratively construct the conflict graph and use a combination of heuristics and exact algorithms to efficiently compute traffic plans for static scenarios. Then we extend the conflict-graph-based approach to dynamic scenarios. We identify the additional challenges and provide means to quantify and control the service offered to flows throughout the traffic plan updates.

In the second part of this thesis, we focus on the analysis of network elements with time-triggered service intermittence in the framework of Network Calculus. Time-triggered service intermittence, for instance, occurs if the bridges in the network use a so-called gating mechanism to enforce such transmission schedules as synthesized in the first part of this thesis. Here, this gating mechanism can have the effect that some other traffic will not be forwarded during certain time intervals. However, we can also observe similar effects, that is, some traffic is offered no (forwarding) service, in network elements with power-cycling for energy conservation reasons. Therefore, we more generally investigate this phenomenon—referred to as time-triggered service intermittence—where network elements stop offering service to some traffic according to a given schedule.

To be able to analyze the worst-case bounds on latency or buffer usage in these scenarios with Network Calculus, we require a formal description of these systems first. To this end, we identify two generic archetypes of network elements with time-triggered service intermittence which require different treatment in the analysis. We provide time-variant as well as time-invariant service-curve formulations for both types of systems. Service curves capture the forwarding behavior of these network elements and can be used for the analysis of composite systems using Network Calculus. We numerically evaluate our service-curve formulations with regard to the under-approximation of offered service and discuss influencing factors and limitations.

Kurzzusammenfassung

In zunehmendem Maße bilden echtzeitfähige Kommunikationsnetzwerke das Rückgrat, das einzelne Komponenten von vernetzten Anwendungen verbindet. Herkömmlicherweise finden sich solche Anwendungen zum Beispiel in Industrieautomationsanlagen oder Fahrzeugbordnetzen. Je mehr sich Software in Infrastruktursysteme einnistet, beispielsweise im Zusammenhang mit Smart Grids oder Connected Driving, desto mehr steigt der Bedarf an Echtzeitkommunikation. In der Folge werden Netzwerke mit zusätzlichen Mechanismen ausgestattet, die durch ein Zusammenspiel von deterministischem Medienzugriff, zeitlicher Koordination, Pfadberechnung und Ressourcenreservierung Garantien, zum Beispiel hinsichtlich Latenz oder Verzögerung, gewährleisten können.

Im ersten Teil dieser Arbeit wird daher das Problem behandelt, wie sich ein Verkehrsplan erzeugen lässt, der Echtzeitgarantien für ein gegebenes Netzwerk und eine Menge von Datenströmen sicherstellt. Dazu werden unterschiedliche Ansätze für verschiedene Ausprägungen des Verkehrsplanungsproblems für zeitgetriggerte Datenströme vorgestellt und anhand einer großen Auswahl von synthetischen Szenarien mit prototypischen Implementierungen evaluiert. Diese Ansätze unterscheiden sich unter anderem hinsichtlich der zeitlichen Auflösung und dem Freiheitsgrad bei der Pfadberechnung sowie der Methode, die zur Berechnung des Verkehrsplans genutzt wird.

Zwei dieser Ansätze beruhen auf Constraint Programming. Konkret bedeutet das, dass die unterschiedlichen Verkehrsplannungsbedingungen für die Zeitpläne und Pfade von zeitgetriggerten

Datenströmen als Integer Linear Programs mit linearen (Un-)Gleichungen formuliert werden. Diese beiden Ansätze werden für die Planung von sogenannten Complemental Flows erweitert. Als Complemental Flows werden hierbei Datenströme bezeichnet, die aus einem zeitgetriggerten Teil und einem sich ergänzenden ereignisgetriggerten Teil bestehen. Weiter werden zwei auf Konfliktgraphen basierende Verfahren zur Verkehrsplanung vorgestellt. Im zuerst vorgestellten Verfahren wird dabei der Konfliktgraph schrittweise aufgebaut und die Verkehrspläne für statische Szenarien werden effizient mit einer Kombination von einem exakten Algorithmus und einer Heuristik berechnet. Als zweites wird der Konfliktgraphansatz für dynamische Szenarien erweitert. Die sich dabei ergebenden zusätzliche Herausforderungen werden identifiziert und Mittel zur Quantifizierung und Steuerung der Dienstgüte bei Verkehrsplanaktualisierungen vorgestellt.

Im zweiten Teil dieser Arbeit wird die Analyse von Netzwerkelementen mit zeitgetriggerten Dienstunterbrechungen im Network Calculus-Framework betrachtet. Zeitgetriggerte Dienstunterbrechungen treten beispielsweise in Netzwerkbrücken auf, die die Zeitpläne, die im ersten Teil dieser Arbeit betrachtet wurden, mit Hilfe eines Sperrmechanismus' durchsetzen. Dieser Sperrmechanismus wirkt sich so aus, dass ein Teil des Datenverkehrs für bestimmte Zeiträume nicht weitergeleitet wird. Ein ähnlicher Effekt ergibt sich auch in Netzwerkelementen, die aus Gründen der Energieeffizienz eine Leistungsabschaltung durchführen. Deshalb wird das verallgemeinerte Problem der zeitgetriggerten Dienstunterbrechung, bei dem Netzwerkelemente nach einem festgelegten Zeitplan den Dienst für bestimmte Datenströmen unterbrechen, untersucht. Eine Voraussetzung, um die Latenzgrenzen und den maximal belegten Zwischenspeicher in diesen Szenarien mit Network Calculus zu berechnen, ist zuerst einmal eine formale Beschreibung dieser Systeme. Deshalb werden zwei Grundformen von Netzwerkelementen mit zeitgetriggerter Dienstunterbrechung identifiziert, die jeweils auch beide unterschiedlich in der Analyse behandelt werden müssen. Für diese Grundformen werden zeitvariante und zeitinvariante Servicekurven herausgearbeitet. Eine Servicekurve beschreibt dabei das Weiterleitungsverhalten der untersuchten Netzwerkelemente und kann für die Analyse von zusammengesetzten Systemen mit Network Calculus verwendet werden. Diese Servicekurven werden in Hinblick auf die Unterabschätzung des angebotenen (Weiterleitungs-)Dienstes ausgewertet und Einflussfaktoren sowie Einschränkungen diskutiert.

1 Introduction

Timely delivery is of ubiquitous importance, be it the delivery of physical goods, or, more abstractly, the delivery of information. In this thesis, we address the latter, and in particular, we target the problem domain of information flows in data networks. Usually, a data network does not exist for its own sake but implements a communication service to facilitate the exchange of information-carrying data between components of an application which are spread throughout the data network. We call such an application a networked application. If we were to wish for the properties of such a communication service, we would like instantaneous delivery of arbitrary amounts of data at any time for zero cost. Undoubtedly, centuries' worth of engineering and research have brought us closer to that ideal. But yet, such an ideal communication service is unheard of, and, for all that we know about the laws of nature, this is how it will likely stay. Therefore, it is no surprise that we have to find acceptable trade-offs and compromises which very much depend on the specifics and requirements of the networked applications.

1.1 Time-Sensitive Communication

One entity that is fundamentally involved in many of these trade-offs and compromises is time. There exist many concepts that effectively trade in some time to compensate non-ideal behaviors of a communication service. For example, we can store the data for some time and then send it again if it appears that it was not delivered successfully. This allows to compensate data loss, which might occur in a non-ideal communication service, and practically this is implemented with techniques such as acknowledgments, time-outs, and retransmissions. Similarly, data can also be stored temporarily to improve the average throughput. This is also known as buffering and is commonly used.

The peculiarity is that trade-offs or compromises involving time are one-sided: many properties of a communication service can be improved at the expense of time, but we are not aware of anything that allows us to create more time. Even worse, time is very much outside of our control. We cannot stop it, and we lose time even if we cannot use it in a meaningful way.

This matters once we look at networked applications that require data to arrive in time and where data not arriving when needed or not arriving at all incurs unacceptable costs. Examples of such applications can be found in many domains, for example, vehicular onboard networks, industrial manufacturing, or power-grid automation. In more abstract terms: we find such networked applications in the domain of cyber-physical systems where computational processes directly interact with the physical world. An important class of cyber-physical systems are networked control systems. In networked control systems, one or multiple parts of the control loop are closed via the data network and the controller design incorporates assumptions about the communication service. In cyber-physical networked applications, the physical part of the application oftentimes much more imposes its rules on the cyber part of the application, and if it is the other way around, then it is many a time not in the original interest of the user—think of motion controllers commanding actuators to ignore physical obstacles. To ensure that a networked control system is able to control a physical process in a meaningful way, we need not only sufficient information about the state of this process, but we need to have that information available at the right time, and we need our commands to be applied at the right time—it does not really help to issue the command to open an overload valve after it blew up.

It takes efforts throughout the whole networked application to get the timing right. Even if the communication service was ideal and delivers information instantaneously, it is to no avail if the computation of the control command takes arbitrary time and vice versa. In this thesis, we consider what happens in the network, and—to some degree—draw the line at the network interface boundary.

Implementations of data networks offer in many regards only limited resources for the delivery of data. Most notable among these limitations is probably how much data can be transmitted in a certain period of time, and how much data can be temporarily buffered in the network. But these limitations also extend to more subtle aspects such as what computational power does the data network possess? For example, on which level can the data network differentiate between data? If the data network can buffer data, which algorithms can it execute on that data, for instance, to decide which data to deliver next, or which data to modify or discard? We do not want to go into the specifics of these limitations here, but we want to point out a more general consequence of these limitations. It is rarely the case that we can make substantial statements about the communication service by only looking at the properties of the data network alone. Instead, due to the aforementioned resource limitations—whatever shape they may have—the communication service also depends on the traffic, that is, the data flows in the network.

In the context of communication services that support the timely delivery of data, there is no single dominating approach. Yet again, what exactly we mean by timely delivery, what we know about the data network and the networked applications, what is subject to our control, and what is imposed externally, makes certain approaches more sensible than others. For example, if we have a global view on the data network and the networked applications therein and some amount of influence on the networked applications, then a well-established paradigm to achieve real-time communication is the combination of time-triggered transmissions with a network-wide coordinated configuration of routes and schedules. Then the challenge lies in traffic planning that synthesizes these configurations with the required properties.

If we have less influence or less knowledge about the networked applications, another way to go is to perform an analysis using the information we have and decide whether we can accept a certain scenario. This is applied, for example, in admission control for scenarios where information about the network and networked applications is only revealed during runtime, or when we only have stochastic information about the traffic generated by a networked application.

In this thesis, we will investigate both sides. We start with the synthesis of network configurations, which means that we present different approaches for *traffic planning*. Traffic planning is the key enabler of time-triggered real-time communication in distributed systems, and it is known to be notoriously hard. Then, we turn towards analysis and investigate how to *model service intermittence* in the elements of a data network.

1.2 Structure of this Thesis

The technical content of this thesis is organized into two main parts.

The first part (Chap. 2 – Chap. 5) focuses on *traffic planning*. This means we concern ourselves with different methods and modeling approaches for the *constructive* task of computing a traffic plan for a set of flows in a network. Or, in other words, in the first part of this thesis, we are given a system and some requirements, and we have to create (or synthesize) the rules of operation for the application and the network—the so-called traffic plan. We present the technological background and introduce a reference system model that we will use throughout the first part of this thesis. Then we give an overview of the traffic planning problem and common constraints and requirements before delving into the details of the different approaches and methods.

In the second part (Chap. 6), the perspective flips from *synthesis* to *analysis*. This means we are provided with a system and its operational rules, and our task is to derive certain properties of interest which allow, for example, to estimate the worst-case backlog that occurs during runtime. Our work in this second part (Chap. 6) is done in the Network Calculus framework. Network Calculus already describes methods and means for the composition of systems from fundamental building blocks such as the service curve models which we derive in the second part of the thesis. Therefore, we simultaneously take a step back from the strongly technologically motivated system model and network-wide perspective from the first part and zoom in on the behavior of a single network element in this second part.

1.3 Scientific Contributions

The overarching theme in this thesis is deterministic communication with time-triggered mechanisms. As explained in the previous section, we present concepts for computing traffic plans which provide service guarantees, and we present concepts for modeling network elements with time-triggered service intermittence to facilitate the analysis with Network Calculus.

The contributions in this thesis are mainly based on [Fal+18] by Falk, Dürr, and Rothermel, [Fal+19a] by Falk, Dürr, Linsenmayer, Wildhagen, Carabelli, and Rothermel, [Fal+20] by Falk, Dürr, and Rothermel, [Fal+22] by Falk, Geppert, Dürr, Bhowmik, and Rothermel, and [Fal+19b] by Falk, Dürr, and Rothermel:

1. We provide two different integer linear programs (ILPs) for traffic planning of arbitrary isochronous time-triggered flows with zero-queuing. Here, traffic planning refers to a joint scheduling and routing problem. If an ILP solver can produce a feasible solution to the ILP, we can obtain routes and schedules from the solution of the ILP, which could be used, for example, to configure the hosts and bridges.

    - In the ILP formulation for joint scheduling and route computation, each individual link is modeled with the intent to facilitate the computation of a valid route for each data flow via dedicated routing constraints in the ILP. This variant is "holistic" in that we encode the whole problem using the integer linear framework, and it provides us with a formal description of the traffic planning problem. However, it is possible to end up with ILP instances that take several days or even weeks to solve.
    - Therefore, we present an alternative ILP formulation for joint scheduling and path selection. In this variant, the computation of a set of candidate paths for each data flow is performed a priori. Consequently, the routing constraints in this ILP express the need to decide upon which path to select as the route. This means that the entities considered for the routing part are candidate *paths* instead of individual links. While—in theory—for each flow, every possible candidate path could be included in the ILP, we can restrict the number of candidate paths to influence the size of the ILP. On the other hand, the ILP formulation for joint routing and path selection uses a more powerful method to model the time when links are occupied by scheduled transmissions.

    The ILP formulation for joint scheduling and route computation was published in [Fal+18]. The author of this these contributed an estimated 95% of the content of [Fal+18]. The ILP formulation for joint scheduling and path computation was published as part of [Fal+19a].

2. We extended the two different ILP formulations in [Fal+19a] to optimize traffic plans for complemental flows. Complemental flows consist of both, a deterministic real-time part

and a complemental, non-time-triggered part (referred to as opportunistic part). The deterministic part has to be delivered deterministically within bounded latency and follows a fixed time schedule, whereas the opportunistic part is transported with relaxed or no latency guarantees. The basic idea is that the deterministic part of the flow provides the strictly mandatory guarantees to *safely* operate the system, whereas the opportunistic part just improves the system performance beyond the mandatory minimum.

The concept of complemental traffic flows has been jointly developed in the context of the DFG project "Integrated Controller Design Methods and Communication Services for Networked Control Systems (NCS)" [Deu], namely, this involved (in alphabetical order) Allgöwer, Carabelli, Dürr, Falk, Linsenmayer, Rothermel, and Wildhagen. The formalization of the optimal routing and scheduling problem for complemental flows, which includes the specific modeling of opportunistic transmissions and traffic metrics, using (mixed-)integer linear programs for the publication in [Fal+19a] was mainly done by the author of this thesis who contributed an estimated 85% to the content in [Fal+19a].

3. The next contribution is a conflict-graph-based approach for traffic planning, which was published in [Fal+20]. We show how to model the traffic planning problem using a conflict graph and establish the relation between traffic planning for time-triggered flows and the independent vertex set problem in the conflict graph. In addition to the theoretic relation between traffic planning and independent set search, we present an algorithm for conflict-graph-based traffic planning and evaluate its proof-of-concept implementation.

   The author of this thesis contributed an estimated 95% of the content in [Fal+20].

4. Building upon the conflict-graph-based modeling, we present a novel approach addressing traffic planning for time-triggered flows with real-time requirements in scenarios with dynamic changes in the flow set. Our approach takes advantage of the conflict-graph modeling where the conflict graph itself embeds a large portion of the knowledge about the solution space in a way that is mostly reusable for the computation of a new traffic plan. This work was published in [Fal+22], and the contributions are two-fold:

   - It introduces a novel approach for dynamic traffic planning for isochronous and/or cyclic-synchronous traffic flows using the zero-queuing principle. Our approach allows for reconfigurations of some active flows when adding new flows. This is referred to as offensive planning. Compared to defensive planning, which takes the configuration of active flows for granted and only uses the remaining network resources for routing and scheduling new flows, offensive planning allows for better utilization of network resources. While offensive planning obviously has to guarantee deterministic communication in the long run, it may introduce possible short-term degradations [Li+19] in the quality of service (QoS). Therefore, offensive planning makes only sense if we can control, both, the degree and duration of a QoS degradation, and it requires applications that can tolerate controlled fluctuations of QoS, for example, provide some level of jitter tolerance.

Our approach supports offensive traffic planning where the transitions from the old traffic plan to the new traffic plan, that is, traffic plan updates, do not require artificial pauses of sender nodes or dropping packets of active flows. Additionally, we can limit the QoS degradation during updates according to per-flow user-specified bounds. In particular, this allows a mixed-operation mode where a subset of active flows (for example, of jitter-resilient applications) can be reconfigured while the remaining active flows (for example, jitter-sensitive applications) are exempt from reconfigurations.

- The second contribution is a novel heuristic for a variant of the independent colorful vertex set problem with weighted colors, which is the underlying optimization problem for the computation of a new traffic plan.

The contributions concerning the "networking side" of the problem are from the author of this thesis, whereas the novel heuristic for solving the independent colored vertex set problem variant is a contribution of Heiko Geppert. The estimated contribution of the author of this thesis to the content in [Fal+22] amounts to 50%.

5. In the analysis part of this thesis, we address fundamental questions regarding the modeling of network elements with time-triggered service intermittence in the Network Calculus (NC) framework. We identify two archetypes of time-triggered network elements with intermittent service. We present time-variant approaches for deriving service curves in the NC framework which—in absence of specific restrictions on the service process of these network elements—differ depending on the archetype. We present approaches for deriving time-invariant service curves for these two archetypes of network elements, too. In particular, we show that a leftover service-curve approach is not applicable to obtain valid time-invariant service curves for both types and provide an alternative approach to model time-triggered network elements with intermittent service with time-invariant service curves. We evaluate the differences between time-variant and time-invariant service curves with respect to the overestimation of worst-case backlog and worst-case delay, and we identify schedule properties that influence the tightness of the derived bounds.

   This contribution was published in [Fal+19b]. The author of this thesis contributed estimated 95% to the content in [Fal+19b].

In addition to the work covered in this thesis, the author coauthored and contributed to [Fal+19c; Hel+20a; Hel+20b].

2 Background: Moving Packets Through Space and Time

Technological developments and academic research often mutually influence each other. Our research was, for example, in some parts motivated by the efforts of the industry and standardization bodies to equip the popular IEEE Std. 802.1Q [IEE18b] compliant Ethernet [IEE18a] networks with real-time communication capabilities in the context of the Time-sensitive Networking (TSN) initiative. A key mechanism in TSN networks is time-aware shaping as standardized by IEEE Std. 802.1Qbv[IEE16] (which later got merged into IEEE Std. 802.1Q-2018 [IEE18b]). Time-aware shaping introduces support for time-triggered (scheduled) forwarding operations in the bridges in the network according to a cyclic time schedule.

While these technological developments provide some of the context for our work on traffic planning, we adopt a more general perspective in the following. To this end, we abstract away some technology-specific implementation details with little qualitative influence to the core ideas of our traffic planning approaches and give an overview of the fundamental components, behaviors, and principles that form the backdrop of our traffic planning approaches in this part of the thesis.

2.1 Why Nodes?

When we usually think of a network, we think of devices such as switches, bridges, routers, and computers that are somehow connected by links.

All these aforementioned devices are what we refer to as nodes: A node is some kind of automaton or computing machine which may perform arbitrary operations using locally available information inside itself. A node can include clocks or timers which allow nodes to execute certain actions at specific points in time or after a specific time interval has elapsed. A node can also have other non-network inputs and outputs, for example, user-interfaces, sensor or actuator interfaces, but our interest lies in the interaction of nodes with the network.

For a node to access information that resides on a different node or to invoke a function provided by another node, it needs to communicate with that node using its network interface. For example, one node may have local access to a particular sensor whose sensor data is requested by another node to compute the command for its local actuator. If the nodes are in a local area

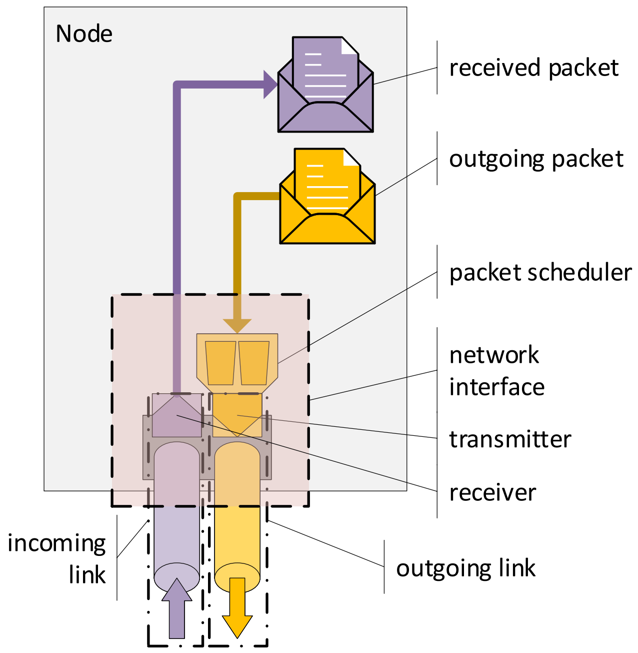


FIGURE 2.1
A node can send and receive packetized data to and from a directly connected node.

network (LAN), the visible part of the network interface could be, for instance, an RJ45-socket or an SFP-slot.

Looking at real-world examples, the predominant reasons to divide an information processing system into distinct entities are composability and space. In particular, how the nodes are distributed "in space" influences the constraints that shape the communication service.

2.2 Transmitting Data Packets via Links

Moving information from one node n_a to another node n_b is only possible if there is a link from the network interface of node n_a to the network interface of node n_b. We consider a link to be a physical system that allows the directed transmission of a data-carrying signal from one network interface to another network interface, for example, an Ethernet cable and attached circuitry or an optical transmitter and optical receiver attached to an optical fiber. This means a link in the physical layer corresponds to a physical, information-carrying channel in the real world.

If both, the link from n_a to n_b and the link from n_b to n_a, connect the network interfaces of nodes n_a and n_b, we have a full-duplex link that allows bi-directional communication. In this case, we call n_a and n_b directly connected. Network interfaces connected by a full-duplex link contain both, a transmitter and receiver, for the data-carrying signal.

Given two connected nodes, how is data transmitted from one node to another connected node? In this part of the thesis, data does not liquidly flow through the data layer. Instead, data is encapsulated in discrete chunks, so-called *packets*, see Fig. 2.1. The total amount of data in each packet, the packet size, is commonly expressed in bits. If node n_a sends a packet to node n_b via the link (n_a, n_b), node n_b will not have the packet available immediately, but it takes some time to transmit the packet via the link.

In this part of the thesis, there is in general no interleaved transmission of bits of different packets nor parallel transmission of multiple packets over one link. Then, at any time, there can be at most one packet in transmission on a single link. Consequently, we would consider physical links that can carry multiple packets at once, for example, an optical fiber in combination with wavelength division multiplex, as multiple (parallel) links. In this sense, the packet transmission over a link is a strictly serialized process, and, by default, we disallow the preemption of packet transmissions. If applicable, we will briefly point out possible relaxations of this restriction later in this thesis.

In practice, the transmission process that is used to transmit a packet from one node to another node often operates by mapping data bits to symbols that are represented, for example, by certain electrical signals. The circuitry that generates and detects these symbols does not operate arbitrarily fast. Therefore, the amount of data that can be transferred via a link from one node to another node in a given time interval is limited and is often expressed by a link property called bandwidth with unit $\left[\frac{\text{bit}}{\text{s}}\right]$. Since we assume nodes to be spatially separated, we also have to account for the fact that the signal which carries our symbols does not propagate arbitrarily fast, and it takes some time to cover the distance to the other node.

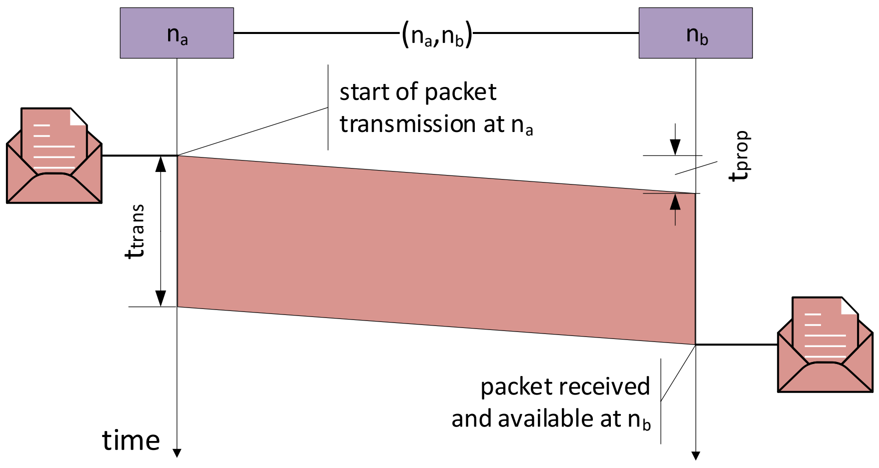
We use two components to model the delay that describes the time after which a packet sent by node n_a is available at node n_b.

transmission delay The transmission delay t_{trans} is defined as the time it takes the sender node to generate the signal which fully represents the packet. Consequently, it depends on the packet size and the bandwidth of the link via the relation

$$t_{\text{trans}} = \frac{\text{packet size } [\text{bit}]}{\text{bandwidth } \left[\frac{\text{bit}}{\text{s}}\right]}. \tag{2.1}$$

propagation delay The propagation delay t_{prop} is defined as the time it takes the signal generated at the sender node to propagate across the link to the receiver node. The propagation delay t_{prop} depends on the physical properties of the transmission process and is independent of the packet size.

Fig. 2.2 illustrates the temporal relation between the start and end of packet transmission and the delay components.



n_a
(n_a,n_b)
n_b
start of packet transmission at n_a
t_{prop}
t_{trans}
packet received and available at n_b
time

FIGURE 2.2 In the temporal domain, we describe the transmission of a packet between two nodes connected by a link with the transmission delay t_{trans} and propagation delay t_{prop}.

If, for some reason, node n_a has multiple packets ready for transmission at the same time, the packet transmissions are serialized, and packets are queued up—stored intermediately—until they are selected for transmission over the link. We assume that links start with the transmission of the next available packet as soon as the previous transmission is completed. Since links are directed, there can be simultaneous packet transmissions in progress in each direction on a full-duplex link.

Figuratively speaking, we can think of a full-duplex link as two packet pipes attached to a network interface: one pipe can be used to send packets to the respective neighboring node, while packets are received from the other pipe. It takes a certain time (t_{trans}) until the whole "packet" has been pushed into the pipe, and, depending on the length of the pipe, the packet will emerge after a certain time (t_{prop}) on the other side. Similar to the real-life example of physical pipes and packets, we assume that the packet has to be fully received before nodes access a packet's content or "do something"—like forwarding—with the packet. In other words, packets are treated as discrete entities by the network.

It is important to point out that we use the term packet in the generic sense as in "packet-switched networks". Therefore, one must not confuse our generic notion of a packet with a service data unit (SDU), for example, an IP packet, which can be encapsulated by another lower layer protocol in the classical layer architectures [Bra+87; Bra89; ISO+96]. Instead, packet refers to the basic unit of operation that occupies the link for a duration of t_{trans} time-units.

2.3 Selecting Packets for Transmission

If there is more than one packet ready to be transmitted over a link, the question arises which one to select and to transmit next. We call the component responsible for this task the packet scheduler of the outgoing link.

The packet scheduler operates like a packet storage from which the link obtains the next packet to transmit. We call this packet that will be retrieved next by the link the next eligible packet. Depending on the features provided by the packet scheduler different rules determine the next eligible packet.

First-In-First-Out Packet Scheduling In the simplest case, the packet scheduler reduces to a single FIFO queue. In this case, the link will transmit the packets in the same order in which they have been passed to the packet scheduler. This means always the head-of-queue packet is eligible for transmission.

Strict-Priority Packet Scheduling A packet scheduler with priority queuing has multiple FIFO queues. Priority levels are mapped to these FIFO queues, and packets are inserted into the appropriate queue according to their priority. The priority of a packet can be embedded in the packet itself in the form of some datum in the packet's metadata, or it can be assigned by the node according to some rules. With strict-priority queuing, the packet scheduler always returns the head-of-queue packet from the priority queue with the highest—that is, the most important—priority level.

Time-Aware Shaping A packet scheduler with the time-aware shaping feature has two additional elements: a shaper module and a controller that changes the gate states according to a schedule, see Fig. 2.3.

We can think of the shaper module as a packet *gate*, which is located in-between a packet queue and the link. If the gate is closed, it interrupts the data path between its packet queue and the link. As long as the gate is shut, the link cannot retrieve and transmit a packet from the packet queue behind the closed gate. When the link idles and a packet is inserted into a queue behind a closed gate, it is still not eligible (available for transmission). If the packet scheduler contains multiple queues, we can have per-queue gates which can open and close independently.

The *gate controller* instructs the individual gates when to close and open according to a cyclic schedule which is also referred to as *gate control list*. Starting at a given point in time, the gate controller applies a certain gate control vector, which describes the state for each gate, by opening and closing the gates accordingly. Each gate control vector is valid for a certain time. After the corresponding time interval has elapsed, the controller proceeds to the next

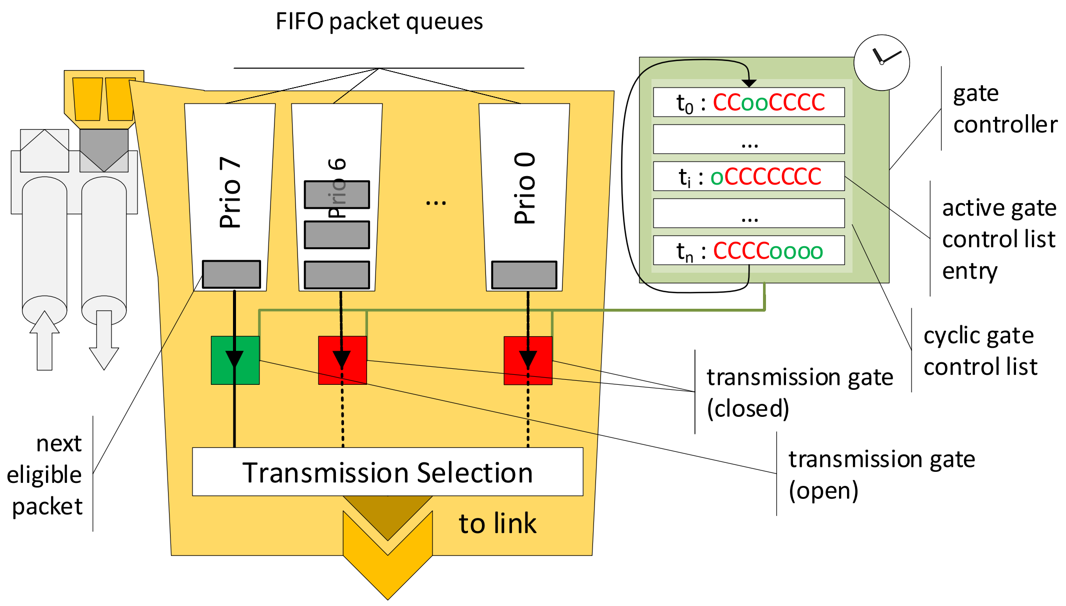


FIGURE 2.3 The packet scheduler determines which packet is eligible for the next transmission. In this example, the packet scheduler is composed of multiple packet queues for different priority levels with time-aware shaping.

entry in the gate control list. Having reached the end of the gate control list, it loops back to the beginning. Therefore, the gate-opening and gate-closing operations repeat cyclically.

If time-aware shaping and strict-priority queuing are combined, the gating can override the priority levels. For example, if the gate of a non-empty queue with high priority level is closed, the link can still transmit packets from a (non-empty) queue with lower priority level with an open gate. If multiple gates are open at the same time, the packets are retrieved from those queues with open gates according to the priority levels.

2.4 Bridges

So far, we discussed how directly connected nodes communicate. In practice, for example, due to reasons of scalability, nodes often communicate indirectly. Instead of connecting every node to every other node, there are specialized nodes that work as intermediaries and forward packets to other nodes. We refer to such nodes as bridges. If two nodes communicate indirectly via bridges, the end-to-end properties of the packet delivery depend on how these bridges operate.

A simplified bridge structure is depicted in Fig. 2.4.

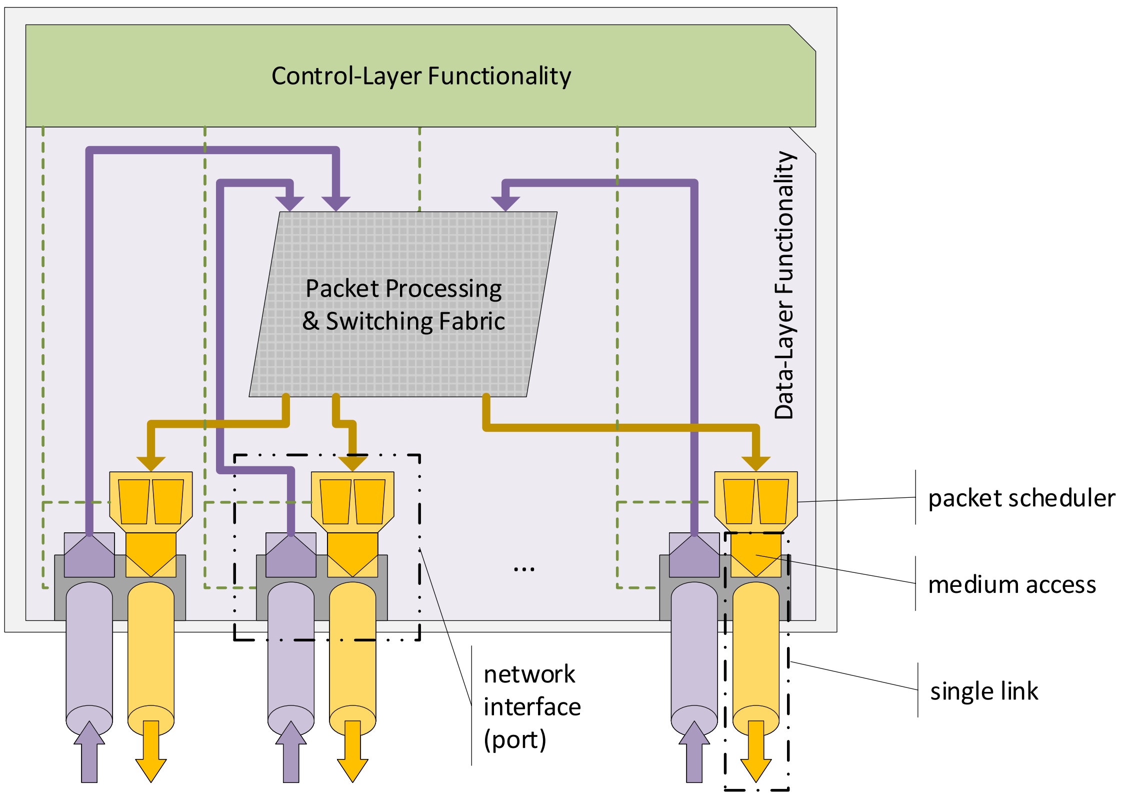


FIGURE 2.4 Bridges are specialized nodes that enable indirect communication between nodes by relaying packets.

In principle, a bridge operates by passing received packets to the packet scheduler of some appropriate outgoing link over which the packet then is transmitted towards the destination node. Packets may be temporarily stored inside the packet schedulers' queues before they are transmitted on the corresponding outgoing link. In this context, the network interface of a bridge is also referred to as *port*.

The bridge evaluates some priorly installed rules to decide which of its outgoing links it shall use to forward the packet. These rules are, for example, encoded in routing table entries or forwarding table entries. In practice, evaluating the forwarding rules usually results in a simple lookup or table-match operation, for instance, on the identifier of the packet's destination node. We assume that these operations are either implemented in hardware or with hardware support to the effect that the only variable part of the total time it takes a packet to travel through a bridge is the time the packet has to wait inside the packet scheduler before transmission. Consequently, the time between the reception of a packet and the start of the transmission on the outgoing link can be expressed as $t_{\mathrm{queue}} + t_{\mathrm{proc}}$, with the constant processing delay t_{proc}, and a variable queueing delay t_{queue}. The processing delay t_{proc} subsumes the time required to perform all the processing steps in the bridge. The queueing delay t_{queue}, in contrast, is the time a packet is stored in a queue of the packet scheduler at the output link, and its value depends on the runtime situation at a specific outgoing link.

In the following, we default to the common bridge model with store-and-forward behavior and output queuing. That means bridges need to have received the complete packet before they can start sending it.

For completeness' sake, we want to point out that there exists an optimization referred to as cut-through switching. With cut-through behavior, a bridge already forwards the so-far received bits of a packet as soon as it is possible to decide which outgoing link to use. This means with cut-through switching a packet can pass a bridge "in-flight" with the reception on the incoming link and the transmission on the outgoing link overlapping in time.

2.5 Hosts

Hosts are nodes that serve as communication endpoints and have identifiers in the network. These identifiers are often included in the packets. In practice, packets often include the identifying information of the source node and destination node. Well-known examples of such identifiers are MAC addresses or IP addresses.

From a networking point of view, hosts act as traffic sources and traffic sinks, respectively. This means hosts create packets and inject them into the network. Similarly, hosts can consume packets that they receive from the network—at least those packets which are targeted at them. In this context, packet consumption at the sink node means that the data from the packet is used inside the node itself and does not circulate in the network anymore.

For the traffic planning approaches, we assume that traffic sources can control the point in time when they release a packet into the network. Current commodity network interfaces already provide hardware support for this feature. Alternatively, depending on the required timing precision, current operating systems provide software implementations to achieve similar behavior.

2.6 Clocks and Time-Synchronization

Time-aware shaping is a time-triggered mechanism with each gating event being triggered at a certain, scheduled point in time. Similarly, packet creation or packet transmission in hosts can be triggered by a time-driven mechanism, for instance, in networked control applications with periodic sampling.

Especially in the context of cyber-physical systems, a globally common notion of time is preferable where on every node the duration of a certain time interval measured with the local clock will result in the same amount of elapsed time measured on an ideal clock. Similarly, at a certain point in time according to an ideal clock, the current time on all local clocks shall be the same. Figuratively speaking, we prefer that the hands of all local clocks are in the same position and move in unison. To achieve this, nodes can run a clock synchronization protocol.

The problem of clock synchronization and clock drift elimination is not in the scope of this thesis. Instead, we refer to clock synchronization protocols such as IEEE 802.1AS [IEE20a] or the Precision Time Protocol [IEE20b].

2.7 Time-Sensitive Networked Applications

A particular challenge arises when networked applications require certain (here: mostly temporal) constraints to be met by the network with regard to packet delivery.

One such important constraint is that the time between packet transmission at the source node and packet reception at the (final) destination node, the so-called end-to-end delay t_{e2e}, is bounded. We refer to a networked application that requires a bounded end-to-end delay as a real-time networked application. In this thesis, we consider real-time networked applications that rely on periodic transmissions with hard real-time guarantees, that is, an end-to-end delay lower than a threshold and bounded jitter. Jitter refers to the variance with regard to the time it takes to deliver a packet from its source node to its destination node.

In the following, we primarily focus on end-to-end delay bounds. This has two reasons: 1) our traffic planning approaches practically eliminate jitter by design, and 2) since we assume a common notion of time in the network, a time-synchronized play-out buffer can conceptually compensate jitter for a real-time networked application by releasing each packet in a node at the latest point in time that meets the end-to-end deadline.

2.8 Architecture: Organizing the Network Functionality

If the network is to provide a communication service for networked applications, nodes usually need to do more than simply sending a packet out via the network interface, especially if the support of intermediary nodes is required to deliver a packet to its final destination. We have hinted at this already in the previous chapter. For example, bridges need to know which outgoing link to forward a packet on, clocks need to be synchronized, gate schedules need to be configured, and so on.

To implement time-sensitive communication in the network, we use knowledge about and extensive coordination of the network functionality that is mostly associated with the Physical Layer and the Data Link layer of the OSI Reference Model [ISO+96]. In IEEE Std. 802-based

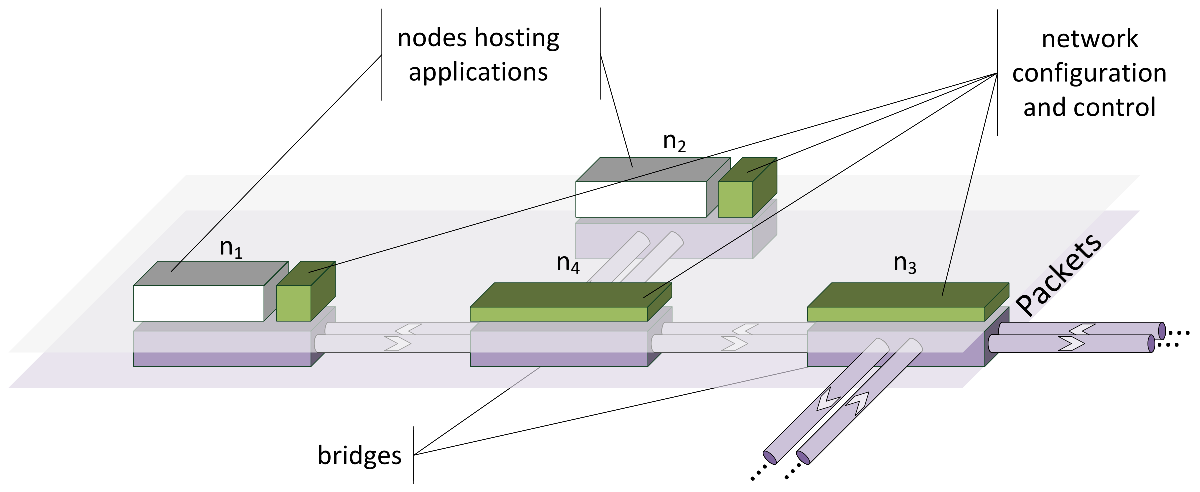


FIGURE 2.5 Network architecture: Networked applications are executed on hosts. All nodes (hosts and bridges) are equipped with network control and configuration functionality, which can be used to influence how packets are handled.

Ethernet networks, the payload is (probably encapsulated in multiple higher level protocols such as IP, UDP, etc.) sent in an Ethernet frame.

However, instead of looking at the network architecture through the lens of hierarchical protocol layers, where each layer consumes the service offered by the subjacent layer, we prefer an SDN-inspired perspective: We consider the network to be organized in three major elements, a data layer, network control and configuration functionality, and networked applications, see Fig. 2.5.

The data layer is responsible for sending and receiving packets between nodes, and we use the control functionality to configure the behavior of the data layer, for example, to install forwarding rules and scheduling rules. The application components on hosts produce and consume the payload for the packets transmitted in the data layer and also interact with the control functionality, for example, to request flows with certain guarantees.

Obviously, each of these elements, data layer, network control and configuration functionality, and applications, can be detailed further. For example, in practice, we usually do not just "pass the packet as-is from the wire to the end-users application code". Instead, the application's payload data is obtained with the help of the network stack of the host's operating system from within multiple nested protocol levels inside the packet.

Analogously, there are multiple alternatives regarding the network control and configuration functionality. For example, if the network topology and the networked applications are known at design time, the traffic planner may exist only during design time, and bridges and hosts can be configured once when building the network. For more dynamic scenarios, a more SDN-like implementation [Ope16] is needed with network control and configuration functionality available at runtime. For example, we may implement the traffic planner as a powerful network controller that communicates with the nodes in the network during runtime, for instance, to receive requests from applications and to update the behavior of nodes via some network configuration protocol.

The specifics of each of these elements and the involved protocols—be it for application payload, or network control and configuration—are necessary to implement and integrate a concrete instance of such a network and the networked applications. However, here, we are more interested in the underlying, more general principles of traffic planning and therefore continue with our more abstract view of the network.

2.8.1 Data Layer

The basic unit of operation of the data layer are *packets*, and the data layer is an abstraction for all functions directly involved in sending, receiving, and forwarding packets between the nodes.

Consequently, the data layer is by and large shaped by the actual, physical devices that form the network. For example, a link between two nodes in the data layer corresponds to some physical connection between the two nodes in the real world.

We assume that the data layer functionality is mostly cast in hardware, and thus we treat the capabilities, properties, and mechanisms provided by the data layer as given.

2.8.2 Network Control and Configuration Functionality

Another important element is the set of mechanisms, protocols, and functionality to configure and manage the data layer to "steer" how the packets move through the network. For example, we can install the forwarding information in bridges to achieve that packets between two nodes are forwarded along a certain sequence of bridges. Another part of this network control and configuration functionality is the ability to compute these paths, which may be subject to certain additional constraints. In networks where bridges have packet schedulers with different priority queuing levels, we assume that most packets will include a priority indicator field that is evaluated in the bridges to enqueue packets accordingly. In such scenarios, we expect to be able to instruct hosts or network ingress nodes to include a certain priority indicator meta-datum in the packets.

While the functionality in the data layer is tightly linked to physical devices, the functionality for network control and configuration is provided on a logical level. However, we assume that there is no out-of-band communication that can "magically" outpace our payload data in order to control the data flow through the network. In the style of SDN, we also refer to the totality of all components that provide network control and configuration functionality as the control plane.

2.8.3 Application Components

Finally, from an end user's perspective, the most important elements are networked applications. Each networked application consists of multiple application components that are distributed across multiple hosts in the network. The networked application shall provide some utility to the end-user, for example, control of a physical plant. To this end, the data layer and the network control and configuration functionality are mere means whose purpose is to offer a communication service to transport the application's payload data. We can therefore say that the raison d'être of the network is to transport payload data between networked applications on different nodes.

In this model, the payload data is generated and ultimately consumed in the networked application's components. The components of the networked application also interact with the network control and configuration functionality, for example, to announce their presence and communication requirements, and in turn receive instructions—meaning, for example, when to send packets—such that their requirements can be met.

How do packets fit in this architecture? A packet is formed by complementing the application's payload data with additional metadata which is, for instance, used by the bridges to make a forwarding decision. This metadata usually includes identifiers of the source node and the destination node. If multiple different networked applications are located on the same node, packets should include additional metadata for the node-internal payload demultiplexing, too.

3 System Model and Traffic Planning Problem Overview

Building on Chap. 2, we provide a more formalized description of our consolidated system model and our assumptions in this chapter. Subsequently, we state the traffic planning problem in terms of the consolidated system model in Sec. 3.5. For most of the first part of this thesis (Chap. 2 – Chap. 5), we operate on the same system model. If we refine or otherwise deviate from the consolidated system model in this part, we will point this out explicitly in the respective chapters. In the same vein, we explicitly mention variations or extensions, such as additional objectives, with regard to the traffic planning problem.

3.1 Notational Conventions

$\mathbb{N}$ denotes the set of natural numbers $\{0, 1, 2, 3, \ldots\}$. We use calligraphic letters (for example, $\mathcal{N}, \mathcal{L}$) to denote sets. Bold letters are used for indexable data structures. Here, an indexable data structure is some sort of container which allows us to access a particular datum, which itself may be another indexable data structure, using an index. The index itself may consist of multiple components. For example, we consider matrices and vectors as such indexable data structures. The value of matrix $\mathbf{X}$ in row i and column j is then given by $\mathbf{X}[i, j]$. We use $*$ as a wild card character in expressions where an index $i \in \mathcal{I}$ may have any possible value from its domain. For example, $\mathbf{X}[i, *]$ refers to all values in row i.

3.2 The Network Graph

We model the network by means of a directed graph $\boldsymbol{G}_n(\mathcal{N}, \mathcal{L})$ where we denote the set of nodes by $\mathcal{N}$ and the set of links by $\mathcal{L}$. Network nodes are the vertices in $\boldsymbol{G}_n$. Generally, we expect that $\boldsymbol{G}_n$ is not disconnected. A single link $\ell \in \mathcal{L}$ represents the ability for directed data transfer from node n_a to n_b. If a pair of nodes $n_a, n_b \in \mathcal{N}$ is directly connected with a full-duplex link, we represent the two directed links between these nodes by two directed edges in $\boldsymbol{G}_n$. We define the two operators $\mathrm{n}_\mathrm{out} : \mathcal{L} \to \mathcal{N}$ and $\mathrm{n}_\mathrm{in} : \mathcal{L} \to \mathcal{N}$. The expression $\mathrm{n}_\mathrm{out}(\ell)$ denotes the node that transmits outgoing packets via link ℓ. Conversely, $\mathrm{n}_\mathrm{in}(\ell)$ designates the node that receives incoming packets via link ℓ.

If the network does not contain any parallel links (think “multiple cables” between the same nodes), $\boldsymbol{G}_n$ does not have two or more directed edges that point from the same node n_a to the same node n_b. Without parallel links, we then can equivalently represent each link ℓ as a tuple $\ell \leftrightarrow (\mathrm{n}_{\mathrm{out}}(\ell), \mathrm{n}_{\mathrm{in}}(\ell)) = (n_a, n_b) \in \mathcal{N} \times \mathcal{N}$, and $\{(n_a, n_b), (n_b, n_a)\}$ represents a full-duplex link.

We consider a network where all links operate with the same transmission speed. While it is not necessary for our approach that all nodes and links induce the same processing delay and propagation delay, respectively, we omit node and link identifiers for the network parameters t_{proc} and t_{prop} for ease of presentation. This corresponds to the assumption of a network consisting of homogeneous bridges and links.

3.2.1 Multi-Hop Packet Delivery

If there is no link from node n_a to node n_b, node n_a can still send packets to node n_b if there is a path from node n_a to n_b, and intermediary nodes forward the packets from node n_a to node n_b.

A path in the network is a sequence of k links $(\ell_1, \ell_2, \ell_3, \ldots, \ell_k)$ with the following properties:

- For each pair of subsequent links ℓ_i, ℓ_{i+1}, $i \in [1, k-1]$, it holds that $\mathrm{n}_{\mathrm{in}}(\ell_i) = \mathrm{n}_{\mathrm{out}}(\ell_{i+1})$. In other words, a path is a sequence of links that follows along adjacent nodes.
- We assume that every link and every node occurs no more than once on a path. This means a path does not contain any loops.

In absence of parallel links, a list of nodes $(n_1, n_2, \ldots n_{k-1})$ can equivalently represent a path.

3.3 Traffic Flows

So far, we introduced networked communication on the level of individual packets, stating that networked applications communicate by transmitting information encapsulated in packets. However, we assume that a networked application exhibits a sustained need for communication. Consequently, we expect that a host does not just send a single packet once to other hosts but repeatedly sends packets to a specific set of other hosts throughout its lifetime. To describe these repeated packet transmissions between specific pairs of nodes, we use the concept of traffic flows.

A single traffic flow f represents a directed stream of packets that traverse the network from the flow’s source node to the flow’s destination node. We restrict ourselves to unicasts where for each traffic flow there is one source node and one destination node. This means a flow is a directed communication channel from one source node to one destination node. Bi-directional communication between a pair of nodes is modeled by two traffic flows.

Packets that belong to a certain traffic flow f do not only share the same source node and destination node but they are also forwarded along the same path. We call a path that is associated with a specific flow the route of the flow. In this sense, a traffic flow (in short: flow) provides spatial information about a set of packets in a network: packets that belong to a flow will only appear on the links and nodes on the route of the packet's flow.

Defining the temporal properties of a set of packets is in general far more difficult, which is manifested in the multitude of different characterizations of traffic sources that we can find in the literature.

3.3.1 Time-Triggered Traffic Flows

Of particular interest to us are so-called time-triggered traffic flows. We use the term time-triggered flow as an umbrella term for traffic flows such as isochronous flows, or cyclic-synchronous flows [IEE21a] where packet transmissions are controlled by a time-triggered mechanism. We assume that time-triggered flows carry time-sensitive information, which requires deterministic QoS and lossless communication under normal operating conditions. That is, we consider external failures, such as power loss or physical disturbances, out of scope.

For each time-triggered flow, the transmission of packets follows a periodically repeating pattern where packet transmissions are restricted to certain scheduled time intervals on each link on the route. We refer to these time intervals as reserved windows and denote their length by t_{resv}.

For each flow, we allow only uniformly sized packets. The packet size or, equivalently, the corresponding transmission time t_{pkt} is a flow parameter. This means for all packets sent by the source node of f we have $t_{\mathrm{trans}} = t_{\mathrm{pkt}}$. In the simplest case, each source node sends one packet in the reserved window in each cycle, that is, $t_{\mathrm{resv}} = t_{\mathrm{pkt}} = t_{\mathrm{trans}}$. In general, a source node may send more than one packet for the same flow per cycle with

$$t_{\mathrm{resv}} = m \cdot t_{\mathrm{pkt}}, \qquad m \in \{1, 2, 3, \ldots\} \tag{3.1}$$

such that it fully utilizes the reserved window in each cycle, and packets of the same flow are transmitted back-to-back in the flow's reserved window. From the perspective of a link, a "train" of equally sized packets of f will traverse the link during the window reserved for f, see Fig. 3.1.

The cycle time t_{cycle} of a flow specifies for each flow the frequency of reserved windows and thereby the frequency of packet transmissions. We expect that $t_{\mathrm{resv}} \ll t_{\mathrm{cycle}}$ though strictly speaking t_{resv} might become as large as t_{cycle}. (If $t_{\mathrm{resv}} = t_{\mathrm{cycle}}$, the full link capacity is assigned to this one time-triggered flow.) Each flow can have a different, individual value for the cycle times t_{cycle}, but we restrict ourselves to finite integer values. Given a set of flows $\mathcal{F} = \{f_1, f_2, f_2, \ldots\}$, we define the hyper-cycle t_{hyper} as the least common multiple (lcm) of the cycle times of the flows in the flow set $\mathcal{F}$.

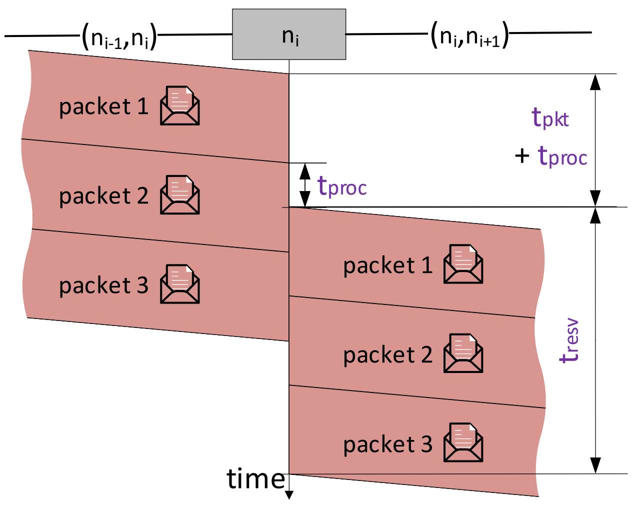


FIGURE 3.1
A reserved window may be used to transmit multiple packets from the same flow back-to-back, here $t_{\text{resv}} = 3 \cdot t_{\text{pkt}}$.

With an absolute time-origin t_0, t_{cycle} also induces a network-wide time-grid for each flow that divides the time-axis into discrete transmission cycles. In the following, we refer to the interval

$$[k \cdot t_{\text{cycle}} + t_0, (k+1) \cdot t_{\text{cycle}} + t_0] \tag{3.2}$$

with $k \in \{0, 1, 2, \ldots\}$ as the k-th transmission cycle of the respective flow.

Usually, packets of a time-triggered flow traverse several links until they reach the destination node. We can use the network-wide time-grid interpretation of the cycle time to “anchor” the position of the reserved window on each link on its route *relative* to the start of a new transmission cycle, see Fig. 3.2. For each flow f, we can specify the positions of the reserved windows for each link on the route of f in terms of a constant offset to the start of a new transmission cycle of this flow.

Of particular importance is the offset of the reserved window on the outgoing link of the source node, the so-called phase denoted by ϕ. This means the source node of a time-triggered flow starts to inject the respective amount of packets that it wishes to send in the reserved window of the k-th transmission cycle into the network at $k \cdot t_{\text{cycle}} + \phi + t_0, k \in \mathbb{N}$. We restrict the valid range of ϕ to $[0, t_{\text{cycle}} - t_{\text{resv}}]$ such that source nodes complete the transmission of packets from the first reserved window within t_{cycle}. We will detail the relation between the offsets and ϕ later.

To summarize, time-triggered flows are an abstraction that describes temporal and spatial properties of an arbitrarily large set of packets for an arbitrarily large time interval by a few parameters—the number of packets a flow sent during runtime then effectively depends on the lifetime of the flow.

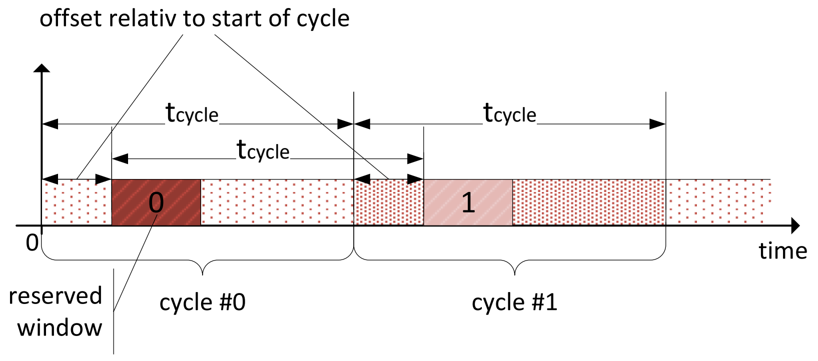


FIGURE 3.2 The schedule of a time-triggered traffic flow on a specific link specifies the offset of the reserved window relative to the start of a transmission cycle.

3.3.2 Establishing a Traffic Flow

In practice, it is rarely productive in terms of configuration overhead or network utilization to operate networks with per-packet individualized forwarding behavior. Instead, traffic flows are a convenient abstraction to describe large amounts of packets with specific properties. This property makes them a good candidate to convey the demands and requirements of a networked application, and they are a reasonable abstraction level for network control and configuration functionality.

Therefore, we assume that a networked application indicates its communication demand to the components responsible for network control and configuration functionality in terms of flows. To this end, networked applications issue a set of *flow requests*. A flow request usually contains a subset of flow properties for the requested flow which the networked application needs to be satisfied to function properly.

In turn, the control plane that receives such a flow request has to figure out whether the network can accommodate a flow with the given parameters. Provided that it can satisfactorily meet the networked application's flow requests—for example, by finding a short path between the source node and destination node with enough spare capacity—the control plane can establish the traffic flow by configuring the respective nodes in the data layer.

Requests for Time-Triggered Flows

If not noted otherwise, flow requests from networked applications specify at least the source node n_{src} and the destination node n_{dst}, the transmission cycle t_{cycle}, the time required for a single packet transmission t_{pkt}, the size of the reserved window t_{resv}, and the end-to-end delay t_{e2e}, see Tab. 3.1. Source, destination, t_{cycle}, t_{pkt}, t_{resv}, and end-to-end delay are flow parameters that do not change during a flow's lifetime.

TABLE 3.1 (Minimal) set of parameters in a request for a single time-triggered flow.

t_{cycle}	transmission period
t_{pkt}	$= t_{\text{trans}}$, time to transmit an individual packet of the time-triggered flow
t_{resv}	total temporal demand per cycle for packet transmissions
t_{e2e}	end-to-end delay bound
n_{src}	source node: the ingress node where packets of the flow enter the network
n_{dst}	destination node: the egress node where packets leave the network

As stated by Eq. (2.1), packet sizes and the time t_{pkt} are directly proportional. We default to specify the packet properties of time-triggered flows (and flow requests) in the temporal domain, since—given that all links in the network have the same bandwidth—the time that a packet occupies a certain link will be our major concern during traffic planning.

3.3.3 Regarding Different Types of Traffic Flows

In many cases, we might have different types of traffic flows in the network. Even though our primary focus lies on time-triggered flows, the network could be used by other types of flows, for instance, best-effort flows.

Here, we assume that time is our most precious commodity with the consequence that we want to protect time-sensitive flow types—in particular time-triggered flows—at the expense of other flow types. To give an example: reserving transmission windows for time-triggered flows has the potential to incur additional costs in terms of an increased storage time of packets of non-time-triggered flows inside the packet schedulers.

Depending on the features and configuration of the network, we can achieve a certain degree of separation between time-triggered flows and other flow types. If not noted otherwise, we use the following scheme:

We assume that all time-triggered traffic flows have the same priority level that is exclusively used for time-triggered flows. This means that there is one queue in the packet schedulers which contains exclusively packets of time-triggered flows, see Fig. 3.3. Additionally, we enforce the reserved windows by opening the gate of the queue with the priority level of the time-triggered flows during the reserved window while shutting the gates of all queues containing packets from other flow types.

Note that since we are ultimately interested in the time-triggered flows, we can relax the assumption of uninterruptible packet serialization for non-time-triggered packet transmissions provided that this does not affect time-triggered packets. For example, the transmission of a non-time-triggered packet may be put on hold during the reserved windows for time-triggered packets and resumed afterward.

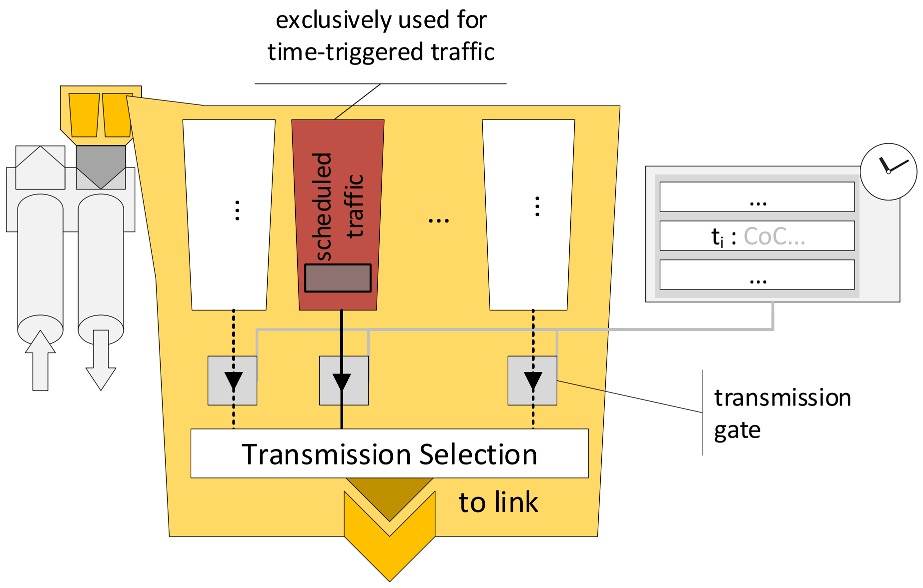
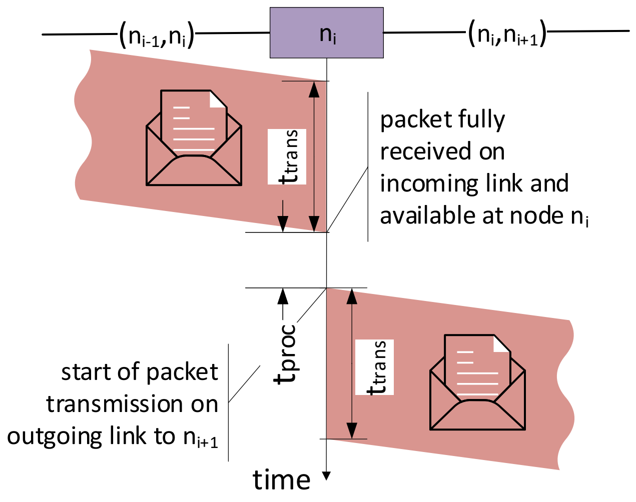


FIGURE 3.3 Time-aware shaper for one port of an IEEE Std 802.1Q compliant switch. We use one queue (traffic class) exclusively for time-triggered traffic.

3.4 Zero-Queuing Principle

In this section, we introduce a particular mode of operation of the network that we will exploit in our traffic planning approaches. Considering our system model for bridges, we see that the time t_{queue} a packet has to wait in a packet scheduler's queue is the only component that varies depending on the runtime (traffic) situation in the network. The amount of queuing delay depends on the number of packets that were enqueued up to this point in time and, even worse, possible future arrivals of packets with higher priorities. Having to track and consider this runtime variable causes a lot of headaches during traffic planning. Indeed, we will show in Chap. 6 that already analyzing and bounding these queuing delays amounts to a non-trivial problem. Therefore, we want to eliminate this runtime-dependent delay component on a packet's path when computing a traffic plan.

One method to achieve this is zero-queuing. With zero-queuing, packets encounter always empty packet scheduler queues while they traverse the network. If the queues of the packet scheduler are empty each time a packet is enqueued, we can eliminate the variable queuing delay ($t_{\text{queue}} = 0$) because each packet is immediately eligible for transmission. Therefore, the total amount of time it takes a bridge until it can start transmitting a packet on the corresponding outgoing link is fixed to t_{proc}. In theory, this makes the packet scheduler superfluous, and with zero-queuing packets of a flow are by definition oblivious to packets from other flows, see Fig. 3.5.



(n_{i-1},n_i)
n_i
(n_i,n_{i+1})
t_trans
packet fully received on incoming link and available at node n_i
t_proc
t_trans
start of packet transmission on outgoing link to n_{i+1}
time

FIGURE 3.4
With zero-queuing, no runtime-dependent packet retention in the packet scheduler occurs, and packets are immediately forwarded.

However, as discussed in Sec. 3.3.3, we protect the zero-queuing packet schedule by opening and closing the transmission gate in front of the time-triggered traffic queue at the start and end of the reserved windows, respectively. While this is transparent from the perspective of the packets of time-triggered flows, it can affect the other traffic types, see Chap. 6.

Time-sensitive networked applications profit twofold from zero-queuing: Firstly, with zero-queuing packets are delivered as quickly as possible and accumulate only the inevitable delay which is caused by the implementation of the forwarding process itself. Packets are not delayed by other traffic. Secondly, given that packet forwarding is a deterministic process itself, zero-queuing also eliminates queuing-delay induced jitter.

Zero-queuing turns out to be beneficial for network control and configuration functionality, too: the forwarding behavior of packets of a specific flow can be characterized based only on that flow's properties and the parameters of the network. In other words, we do not need to know the current traffic situation in the network to determine the progress a packet—or a packet-train—has made on its way to the destination node.

3.4.1 Per-Hop Delay

Zero-queuing allows us to describe all effects that add to the per-hop delay for packets of the same flow in the network by a single constant per-hop delay d_{hop}, see Fig. 3.5.

The per-hop delay d_{hop} can be interpreted as the time between the start of the packet transmission on link (n_{i-1}, n_i) and the start of the packet transmission on the next link (n_i, n_{i+1}) adjacent to (n_{i-1}, n_i). The per-hop delay of flow is given by

$$d_{\text{hop}} = t_{\text{prop}} + t_{\text{pkt}} + t_{\text{proc}} \tag{3.3}$$

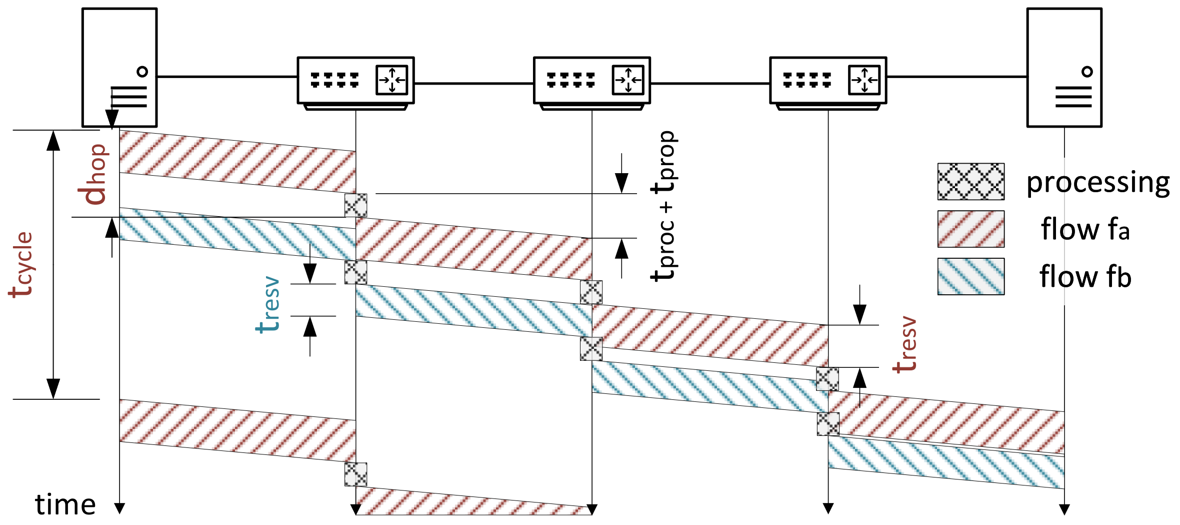


FIGURE 3.5 Zero-queuing illustrated for two flows with different values for t_{resv} and with the source node (on the left) and same destination node (on the right). We assume that processing delay is independent of the packet size.

with the flow property t_{pkt} and the network properties t_{prop} and t_{proc}. Note that d_{hop} depends on the time t_{pkt} required to transmit a *single* packet. Even if $t_{\text{resv}} > t_{\text{pkt}}$, the node can start to forward fully received packets already even the remaining packets of that reserved window are yet to be received, see Fig. 3.1.

We want to briefly point out that the zero-queuing principle can also be applied to cut-through switching in which case the transmission duration of a packet t_{pkt} in Eq. (3.3) has to be replaced by t_{header} which accounts just for the number of bits that have to be received before the transmission of the packet can start.

3.4.2 End-To-End Delay

With zero-queuing, packets accumulate only the (inevitable) delay which consists of the time between receiving a packet and completing the forwarding of the packet on the next link at each hop, see Fig. 3.5.

Thus, zero-queuing enables transmissions with bounded end-to-end delay

$$t_{\text{e2e}} = t_{\text{trans}} + t_{\text{prop}} + \text{len}\,(\text{path}) \cdot d_{\text{hop}} + t_{\text{src}} + t_{\text{dst}} \tag{3.4}$$

that depends on the number of hops len (path) in-between source and destination (len (path) is the number of bridges on the route) and processing delays of source and destination node (denoted by t_{src} and t_{dst}), respectively.

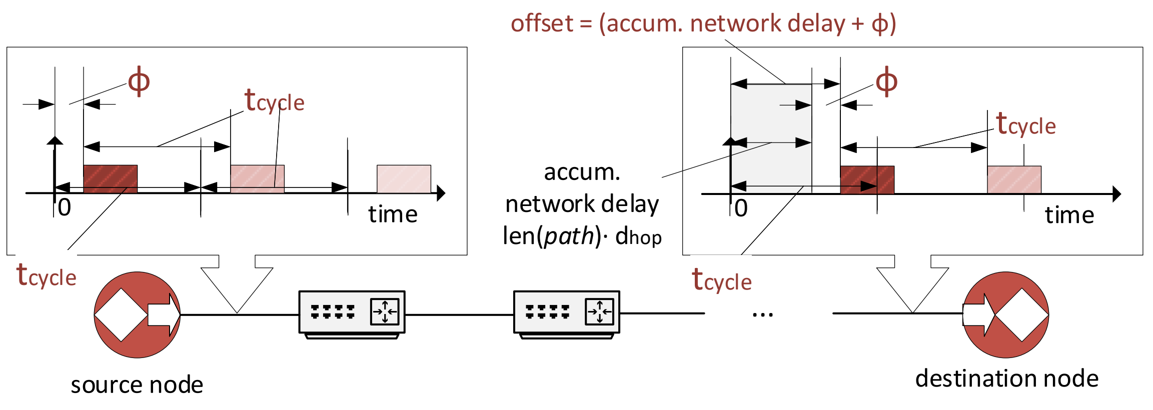


FIGURE 3.6 With zero-queuing, the offset on a particular link depends on the phase and amount of delay accumulated so far on the route to the destination.

Figure 3.6 also illustrates that we can derive the positions of the reserved windows on a particular link—the offsets—from ϕ by accumulating the per-hop delays along the path. For example, consider the following scenario with $t_{\text{resv}} = t_{\text{pkt}}$, (a single packet is sent per transmission cycle): assume the source node starts the transmission of the first packet in the 0th-cycle at $t_0 + \phi$. Then at $t_0 + \phi + t_{\text{pkt}}$, this transmission of the first packet on this link is complete. The transmission of the first packet on the next link then starts at $t_0 + \phi + t_{\text{pkt}} + t_{\text{proc}} + t_{\text{prop}}$, that is, the offset of the reserved window on this link is $\phi + d_{\text{hop}}$. The transmission on the following link then starts at $t_0 + \phi + d_{\text{hop}} + t_{\text{pkt}} + t_{\text{proc}} + t_{\text{prop}} = t_0 + \phi + 2 \cdot d_{\text{hop}}$, and so on. The source node then transmits the second packet at $t_0 + \phi + t_{\text{cycle}}$, etc.

This means knowing one phase value per flow is sufficient as this one phase value determines the offsets of the reserved window on the remaining links to the destination node.

3.5 Problem Overview

In general, it requires concerted coordination between the configuration of the network and the behavior of the source nodes to achieve zero-queuing. In other words, to establish a set of time-triggered flows in a given network with zero-queuing, we have to ensure that packets of time-triggered flows do not “interfere” with other packets and with each other. There are two basic methods to achieve this, namely, spatial isolation and temporal isolation. For spatial isolation, we assign concurrent packet transmissions to disjoint and independent network resources (here: links) to prevent interference. Spatial isolation ultimately requires deciding on the routes of time-triggered flows.

Temporal isolation assigns packet transmissions to disjoint time intervals and leads to a

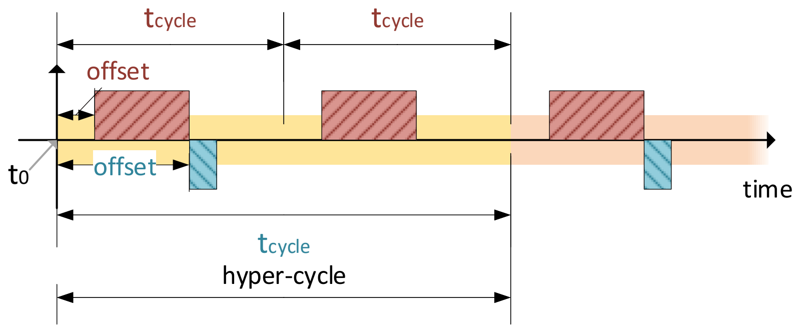


FIGURE 3.7 Example: Appropriately chosen offsets ensure temporal isolation for two flows with different cycle times. Note that link serialization is implicitly achieved by temporal isolation.

scheduling problem, see Fig. 3.7. Either spatial isolation or temporal isolation can in theory be used alone but with one significant drawback: the maximally achievable network utilization is low due to the restriction to disjoint routes or network-wide schedules. To increase the network utilization, we can combine the two dimensions, space and time. This means we are interested in schedules *and* routes, and we refer to a set of schedules and routes as a (global) *traffic plan*.

Note that the term scheduling is overloaded to a certain extent. The packet schedulers at each outgoing link decide locally at runtime which packet to transmit next. In the context of traffic planning, we have a global view of the network. Here, scheduling refers to computing the network-wide timetable of the reserved windows.

The task of computing such a traffic plan for a given set of requests for time-triggered flows—in short: *traffic planning*—will be at the core of the problems discussed in the following two chapters in the first part of this thesis. We consider the network, that is, the network graph and the network properties, and the flow requests as input for the traffic planning. With these inputs, we want to compute a traffic plan for the whole network that satisfies the following traffic planning constraints.

Packet-switching constraint The network topology constrains the movement of a packet through the network. Packets are transported between nodes only via links. Each movement of a packet consumes the corresponding amounts of time, for example, the time necessary for processing, transmission, and propagation of the respective packet.

Serialization constraint (or: Temporal isolation constraint) At any moment and every output port and link in the network, there is at most one packet (of a time-triggered flow) in transmission in progress. This means packet transmissions over links are serialized. From the perspective of traffic planning, this means that at any point in time there is at most one packet transmission scheduled on each link.

Routing constraint Packets of a specific flow enter the network at the source node and have to reach the destination network node where they leave the network. All packets of a flow are routed along the same route, and a route is a path between the source node and the destination node.

Delay constraint Every packet of a time-triggered flow has to reach its destination node within the end-to-end deadline.

Ordering constraint At every node, a packet with a certain priority level can only be transmitted on an outgoing link if all other packets with the same priority level which have arrived earlier at the same node and which have to be transmitted on the same outgoing link have been transmitted before (FIFO property).

Zero-queuing constraint When a packet of a time-triggered flow enters a queue in the packet scheduler of an outgoing link, it is transmitted immediately (zero-queuing).

These constraints will resurface more formally in the following two chapters, but we can make the following, general observations: with zero-queuing, the end-to-end deadline effectively restricts the length of the route. Remember that the zero-queueing constraints allow us for each time-triggered flow to derive the transmission schedule—or more precisely the position of the reserved window on the time axis—on each link from the phase ϕ.

An almost obvious approach for traffic planning is to try to formalize these constraints in a generic framework for constraint-based programming such as (integer) linear programming (ILP), SAT, SMT, and leave the computation of the traffic plan to an off-the-shelf solver. Indeed, this is what we will explore in Sections 4.2 to 4.4 with a focus on ILP. Following that, we explore a different, conflict-graph-based approach that provides different trade-offs.

Note that the traffic planning problem is NP-hard. Depending on how we model the traffic planning problem, we can find different well-known NP-complete problems that are easily related to a specific traffic planning approach. For example, [Dür+16] models the traffic planning problem as no-wait job-shop instance, with the no-wait job-shop problem being one of the established NP-complete problems [Gar+79; Mas+02]. For the ILP-based approach in Chap. 4, we sketch in Sec. 4.2.1 how to reduce Bin-Packing [Gar+79] to the traffic planning problem. The conflict-graph-based approaches from Chap. 5 can be related to the independent set problem (is there an independent vertex set with at least a specific number of vertices?) [Gar+79].

4 ILP-Based Traffic Planning

Recalling the numerous (verbally expressed) constraints on routes and schedules (see Sec. 3.5), it seems intuitive to formalize these traffic planning constraints using some kind of constraint-programming framework. Therefore, in this chapter, we approach the traffic planning task by formalizing the constraints introduced in Sec. 3.5 as a set of linear (in-)equalities or, more specifically, as an integer linear program (ILP).

Once we have constructed an ILP instance from our traffic planning problem instance, we can feed the ILP instance into any ILP solver and let the solver do the computational "heavy-lifting", though it is still in our responsibility to map the solution to the ILP instance back to our traffic planning problem. This means, provided that the solver has even found a feasible solution, we have to extract the traffic plan, that is, the routes and schedules from the ILP solution.

Using the outlined ILP-based approach has some advantages. Firstly, the ILP formulation doubles as a formal, declarative description of the traffic planning problem. Secondly, once we have expressed our problem in the form of an ILP, we may profit from future improvements in the field of integer-linear programming or computing that make their way into ILP solvers "for free". That is, we "only"[1] have to exchange the actual ILP solver implementation against an improved one to take advantage of the benefits of the ongoing research and development of these general-purpose "problem-solving" frameworks and solvers. Thirdly, we can extend the ILP formulations from pure constraint satisfaction problems (compute any valid traffic plan) to optimization problems (compute the best traffic plan).

We briefly introduce integer linear programming in Sec. 4.1. Then we present two different variants for how to express the traffic planning problem as constraint satisfaction problem with an integer linear programming formulation. In particular, we provide a formulation for joint scheduling and route computation in Sec. 4.2 and for joint scheduling and path selection in Sec. 4.3. We extend both variants with additional constraints and objectives for the computation of optimized traffic plans for complemental flows. Complemental flows and the extended ILP formulations are presented in Sec. 4.4. We then discuss related work on traffic planning in Sec. 4.5 and summarize the challenges of ILP-based traffic planning, which motivated our work on conflict-graph-based approaches (see Chap. 5) in Sec. 4.6.

[1]As experience has shown, in practice, swapping out one ILP solver against another often requires adapting a lot of the implementation for constructing the ILP instance from the traffic planning instance and interfacing with the ILP solver.

4.1 Integer Linear Programming

An (integer) linear program consists of an objective function and a set of linear constraints on a set of variables. Commonly, the ILPs are expressed in matrix-vector-notation.

$$\begin{aligned} \max \mathbf{c}^\intercal \underline{\mathbf{x}} & \qquad \text{(objective function)} \\ \text{s.t. } \mathbf{A} \cdot \underline{\mathbf{x}} \leq \mathbf{b} & \qquad \text{(constraints)} \end{aligned} \tag{4.1}$$

with

$$\begin{aligned} \mathbf{c} & \qquad \text{vector of objective function coefficients} \\ \mathbf{A}, \mathbf{b} & \qquad \text{matrix and vector, respectively, of constraint coefficients} \\ \underline{\mathbf{x}} \text{ with } \underline{x_i} \in \mathcal{S}_i & \qquad \text{vector of (decision) variables} \end{aligned}$$

We typeset variables underlined (variable $\underline{x_i}$) to distinguish them from the parameters. Parameters are known a priori, whereas we want to find an assignment of values for each variable $\underline{x_i}$ in $\underline{\mathbf{x}}$ from the respective domain $\mathcal{S}_i$. If variables of Eq. (4.1) are restricted to integral values, we speak of an integer linear program, which is NP-complete [Gar+79]. Some people further distinguish between integer linear programs and mixed-integer linear programs where the former contains only integer variables, and the latter has both, integers and rational variables, but we use the term integer linear program indiscriminately for any problem with some variables restricted to integer values.

An ILP can appear in two forms, either as an abstract ILP model/formulation or as a concrete ILP instance. In its abstract form, it describes for a set of problems with a particular structure how to instantiate an ILP instance for one specific problem. That is, we can think of an abstract ILP model/formulation as a set of rules for the construction of an ILP instance. In our traffic planning approaches, for example, we have to know how many flows there are in $\mathcal{F}$, and we need the numerical values of their parameters to compute the constraint coefficients to construct the corresponding ILP instance. Note that ILP instances for different problem instances that have been instantiated from the same ILP model can differ strongly in the number of constraints and variables. For example, think of one problem instance with few flows in a small network compared to another problem instance with many flows in a large network—the latter results in an ILP instance with more constraints and more variables.

In the context of our traffic planning approaches, we present ILP models that we use to translate traffic planning problem instances or parts thereof into ILP instances. The actual process of solving these ILP instances is performed by an ILP solver, which is a piece of software that takes an ILP instance as input, and returns, if it exists, a variable assignment that satisfies all the constraints, as well as an indication whether a solution is an optimal solution, too. From our perspective, the ILP solver and its inner workings are a black-box component, that is, we do not intervene in the solving process of the ILP solver.

An optimal solution of an ILP instance is an assignment of values to variables such that all constraints are satisfied, and the objective function assumes its maximum value. For the purpose of traffic planning, we are also interested in feasible solutions. A feasible solution is an assignment of values to variables such that all constraints are satisfied but the objective function need not assume its maximum value. If we are only interested in feasible solutions, we use integer linear programming as a constraint-programming tool and ignore its optimization functionality.

There can exist many solutions to a particular ILP instance. It is also possible that no solution exists for an ILP instance, that is, it is impossible to find an assignment of values to $\underline{\mathbf{x}}$ that satisfies all constraints, for example, if constraints are contradicting. We refer to an ILP instance for which no solution exists as infeasible.

Equation (4.1) is a concise way to present an ILP, but we use an alternative, more verbose format to expose the relations between the traffic planning problem and the constraints in a more comprehensible manner. To this end, we give many constraints in a row-oriented format

$$\sum_i a_i \cdot \underline{x_i} \leq \sum_k b_k. \tag{4.2}$$

Some aspects of the traffic planning problem are also much easier to understand if expressed with the help of logical operations:

Implication The implication that linear constraint 2 only applies if linear constraint 1 is satisfied is expressed with the if-operator in the form

$$\text{if (lin. constraint 1) then (lin. constraint 2)}$$

Disjunction The disjunction that it is sufficient to satisfy either linear constraint 1 or linear constraint 2 is expressed with the or-operator in the form

$$\text{(lin. constraint 1 or lin. constraint 2)}$$

These operators do not comply with the standard form Eq. (4.1) or the canonical form of ILPs. Nevertheless, expressions containing both of these operators can be transformed into a set of linear (in-)equalities by the “bigM”-technique if the involved variables are bounded—which they are for our problems (see Appendix A.1.1). However, since these transformations tend to obfuscate the constraints, we will sometimes favor presenting our constraints using these logical operators, and we defer the plain ILP formulations to the appendix.

The constraints as presented with the logical operators are not just easier to understand but can also be directly implemented with current tooling. State-of-the-art commercial solvers such as CPLEX and Gurobi support indicator constraints [Bel+16] to express these logical operators. Current ILP modeling environments such as zimpl [Koc04], GAMS, AMPL, or OPL support modeling problems with these operators, too, and can automatically transform expressions with these operators to a set of linear inequalities that can be processed by ILP solvers.

4.2 Joint Scheduling and Route Computation

The ILP described in this section models the traffic planning problem very close to the individual elements in the data layer. We express the requirements in terms of the forwarding behavior of packets of time-triggered flows on individual links in the network. We embed the constraints that express what we consider a valid route and a valid schedule in the ILP. Then the ILP solver jointly computes the routes and schedules that satisfy our constraints in one step. Therefore, the approach described in the following performs traffic planning by *joint scheduling and route computation.*

ILP Parameters

Next, we describe the traffic planning parameters.

We represent $\mathcal{N}, \mathcal{L}$, and $\mathcal{F}$ as subsets of the natural numbers $\mathbb{N}$ where each element (node, link, flow) is represented by one number. We define the notational shortcut

$$\mathcal{L}_{\text{out},n} = \{\ell \in \mathcal{L} \mid \mathrm{n}_{\text{out}}(\ell) = n\} \quad \text{(set of outgoing links at node } n\text{)} \tag{4.3}$$

$$\text{and} \quad \mathcal{L}_{\text{in},n} = \{\ell \in \mathcal{L} \mid \mathrm{n}_{\text{in}}(\ell) = n\} \quad \text{(set of incoming links at node } n\text{)} \tag{4.4}$$

to denote the set of all incoming and outgoing links, respectively, at a specific node n.

The different per-flow properties that constitute a flow request are represented by the integer components of vectors with each component corresponding to a flow f, see Tab. 4.1. We use f as an index to retrieve a specific parameter for flow f.

ILP Variables

Next, we introduce the variables for the ILP. Our ILP formulation follows a "constructive" approach where we can read the solution for the joint scheduling and routing problem directly from the ILP solution.

Consequently, we introduce a set of binary decision variables $\underline{\mathbf{u}_{\text{link}}}[f, \ell] \in \{0, 1\}$ with $f \in \mathcal{F}$

TABLE 4.1 ILP parameters for joint scheduling and route computation.

$\boldsymbol{G}_n$	network graph
$\mathcal{N} = \{0, 1, 2, \ldots\} \subset \mathbb{N}$	set of nodes
$\mathcal{L} = \{0, 1, 2, \ldots\} \subset \mathbb{N}$	set of links
$\mathcal{F} = \{0, 1, 2, \ldots\} \subset \mathbb{N}$	set of flows
$\mathbf{n}_{\text{src}}[f] \in \mathcal{N},\ f \in \mathcal{F}$	source nodes of flows
$\mathbf{n}_{\text{dst}}[f] \in \mathcal{N},\ f \in \mathcal{F}$	destination nodes of flows
$\mathbf{t}_{\text{cycle}}[f] \in \mathbb{N},\ f \in \mathcal{F}$	flow cycle times
$\mathbf{t}_{\text{pkt}}[f] \in \mathbb{N},\ f \in \mathcal{F}$	transmission time for any packet of flow f
$\mathbf{t}_{\text{resv}}[f] \in \mathbb{N},\ f \in \mathcal{F}$	reserved window sizes
$\mathbf{t}_{\text{e2e}}[f] \in \mathbb{N},\ f \in \mathcal{F}$	maximally allowed end-to-end delay
$\mathbf{d}_{\text{hop}}[f] \in \mathbb{N},\ f \in \mathcal{F}$	per-hop delay for packets of flow f
$t_{\text{hyper}} = \text{lcm}\,(\mathbf{t}_{\text{cycle}}) \in \mathbb{N}$	least common multiple of all cycle times

and $\ell \in \mathcal{L}$ with the interpretation

$$\underline{\mathbf{u}_{\text{link}}}[f, \ell] = \begin{cases} 1 \rightarrow \text{flow } f \text{ uses link } \ell \\ 0 \rightarrow \text{flow } f \text{ does not use link } \ell \end{cases} . \tag{4.5}$$

We can derive the routes of the flows from $\underline{\mathbf{u}_{\text{link}}}$ by following the used links from the source node after having solved the ILP.

Analogously, we introduce a set of bounded integer variables $\underline{\mathbf{t}_{\text{offset}}}[f, \ell] \in \mathbb{N}$ with $f \in \mathcal{F}$ and $\ell \in \mathcal{L}$. As the name insinuates, we interpret the value $\underline{\mathbf{t}_{\text{offset}}}[f, \ell]$ as the offset of the reserved window of flow f relative to the start of the flow's transmission cycle on link ℓ. We can also think of $\underline{\mathbf{t}_{\text{offset}}}[f, \ell]$ as the point in time modulo the flow period that describes the start of the reserved window for flow f if $\underline{\mathbf{u}_{\text{link}}}[f, \ell] = 1$.

The values of $\underline{\mathbf{t}_{\text{offset}}}$ are bounded with the constraints

$$\forall f \in \mathcal{F}, \forall \ell \in \mathcal{L} : \underline{\mathbf{t}_{\text{offset}}}[f, \ell] \geq 0 \tag{4.6}$$

$$\forall f \in \mathcal{F}, \forall \ell \in \mathcal{L} : \underline{\mathbf{t}_{\text{offset}}}[f, \ell] \leq \mathbf{t}_{\text{cycle}}[f] - \mathbf{t}_{\text{resv}}[f], \tag{4.7}$$

to ensure that on every link ℓ used by flow f the reserved window ends before the next transmission cycle starts, see Fig. 4.1. From the values of $\underline{\mathbf{t}_{\text{offset}}}[*, \ell]$, we can then construct the local schedule for link ℓ.

ILP Constraints

We proceed to explain the constraints that encode the requirements for a transmission pattern that separates the reserved windows in time (via scheduling) as well as in space (via routing).

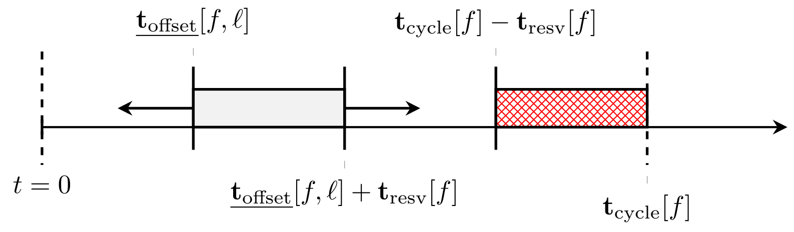

$\underline{\mathbf{t}_{\text{offset}}}[f,\ell]$
$\mathbf{t}_{\text{cycle}}[f]-\mathbf{t}_{\text{resv}}[f]$
$t=0$
$\underline{\mathbf{t}_{\text{offset}}}[f,\ell]+\mathbf{t}_{\text{resv}}[f]$
$\mathbf{t}_{\text{cycle}}[f]$

FIGURE 4.1 Equation (4.7) constrains $\underline{\mathbf{t}_{\text{offset}}}[f,*]$ such that the reserved window does not cross boundaries of a transmission cycle.

These constraints can be categorized in three groups: 1) constraints that ensure per-link compliance of reserved windows, 2) constraints that ensure loop-free routing from flow origin to flow destination, and 3) constraints that link the routing and scheduling constraints.

Here, we present the constraints using the logical operators as introduced in Sec. 4.1. For the sake of completeness, Appendix A.1 contains those constraints as plain ILP expressions.

Routing Constraints Constraints (4.8)-(4.10) express the properties of a valid route for each flow. The route of each flow f starts at the source node $\mathbf{n}_{\text{src}}[f]$. Therefore the constraint

$$\forall f\in\mathcal{F}:\sum_{\ell\in\mathcal{L}_{\text{out},\mathbf{n}_{\text{src}}[f]}}\underline{\mathbf{u}_{\text{link}}}[f,\ell]=1 \tag{4.8}$$

ensures that the route of each flow f contains one link that departs from the source node of f. Analogously, the constraint

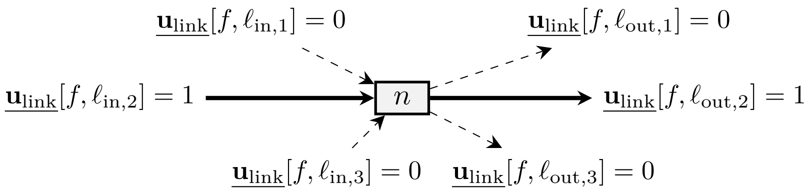
$$\forall f\in\mathcal{F}:\sum_{\ell\in\mathcal{L}_{\text{in},\mathbf{n}_{\text{dst}}[f]}}\underline{\mathbf{u}_{\text{link}}}[f,\ell]=1 \tag{4.9}$$

ensures that for each flow f one incoming link at the destination node of f is part of the route of f. The constraint

$$\forall f\in\mathcal{F},\forall n\in\mathcal{N}\setminus\{\mathbf{n}_{\text{src}}[f],\mathbf{n}_{\text{dst}}[f]\}:\sum_{\ell_{\text{out}}\in\mathcal{L}_{\text{out},n}}\underline{\mathbf{u}_{\text{link}}}[f,\ell_{\text{out}}]=\sum_{\ell_{\text{in}}\in\mathcal{L}_{\text{in},n}}\underline{\mathbf{u}_{\text{link}}}[f,\ell_{\text{in}}] \tag{4.10}$$

restricts the number of incoming links used by flow f to the number of outgoing links used by flow f at each node n.

So far, the routing constraints do not prohibit that the set of used links includes cycles. We



$\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in},1}] = 0$
$\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out},1}] = 0$
$\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in},2}] = 1$
n
$\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out},2}] = 1$
$\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in},3}] = 0$
$\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out},3}] = 0$

FIGURE 4.2 The routing constraints from Eq. (4.8)–Eq. (4.11) restrict the number of incoming and outgoing links at the nodes (here: n is a bridge on the path of flow f).

can rule this out by adding the constraint

$$\forall f \in \mathcal{F}, \forall n \in \mathcal{N} \setminus \{\mathbf{n}_{\text{src}}[f], \mathbf{n}_{\text{dst}}[f]\} : \sum_{\ell \in \mathcal{L}_{\text{in},n}} \underline{\mathbf{u}_{\text{link}}}[f, \ell] \leq 1 \tag{4.11}$$

that forbids using more than one incoming link at every node except the source node and the destination node. But even without this additional constraint, the constraints from the next section which link the routing and scheduling parts of the problem make it very unlikely that the ILP solution contains these cycles.

Scheduling Constraints The first scheduling constraint ensures that the reserved windows for any flows using a particular link do never overlap and is given by

$$\begin{aligned}
&\forall f_1 \in \mathcal{F}, f_2 \in \mathcal{F} : f_1 \neq f_2, \forall \ell \in \mathcal{L} : \\
&\forall a \in \mathcal{A}, \forall b \in \mathcal{B} : \\
&\quad \text{if } \big(\underline{\mathbf{u}_{\text{link}}}[f_1, \ell] + \underline{\mathbf{u}_{\text{link}}}[f_2, \ell] \geq 2\big) \text{ then} && (4.12)\\
&\quad\ (\underline{\mathbf{t}_{\text{offset}}}[f_1, \ell] + a \cdot \mathbf{t}_{\text{cycle}}[f_1] \geq \underline{\mathbf{t}_{\text{offset}}}[f_2, \ell] + b \cdot \mathbf{t}_{\text{cycle}}[f_2] + \mathbf{t}_{\text{resv}}[f_2] && (4.13)\\
&\quad\ \text{or } \underline{\mathbf{t}_{\text{offset}}}[f_2][\ell] + b \cdot \mathbf{t}_{\text{cycle}}[f_2] \geq \underline{\mathbf{t}_{\text{offset}}}[f_1][\ell] + a \cdot \mathbf{t}_{\text{cycle}}[f_1] + \mathbf{t}_{\text{resv}}[f_1]) && (4.14)
\end{aligned}$$

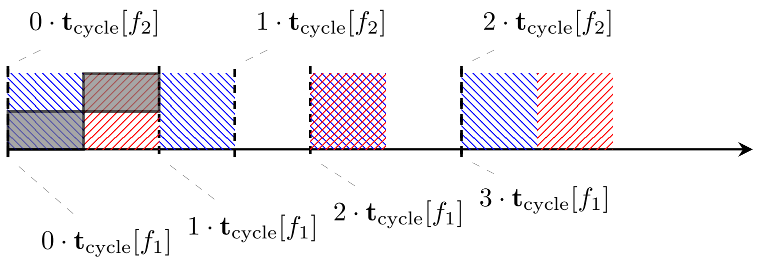
with

$$\mathcal{A} = \left\{ a \in \mathbb{N} : 0 \leq a \leq \frac{t_{\text{hyper}}}{\mathbf{t}_{\text{cycle}}[f_1]} \right\}, \tag{4.15}$$

$$\mathcal{B} = \left\{ b \in \mathbb{N} : 0 \leq b \leq \frac{t_{\text{hyper}}}{\mathbf{t}_{\text{cycle}}[f_2]} \right\}. \tag{4.16}$$

In the formulation of the scheduling constraints, we have to consider that neither do we know the routes of the flows a priori nor do we know the order of the reserved windows on a given link a priori.

This is reflected in an implication (if packets of f_1 and f_2 are routed over link ℓ, ensure



$0 \cdot \mathbf{t}_{\text{cycle}}[f_2]$
$1 \cdot \mathbf{t}_{\text{cycle}}[f_2]$
$2 \cdot \mathbf{t}_{\text{cycle}}[f_2]$
$0 \cdot \mathbf{t}_{\text{cycle}}[f_1]$
$1 \cdot \mathbf{t}_{\text{cycle}}[f_1]$
$2 \cdot \mathbf{t}_{\text{cycle}}[f_1]$
$3 \cdot \mathbf{t}_{\text{cycle}}[f_1]$

FIGURE 4.3 Scheduling constraints from Eq. (4.13) and Eq. (4.14): The third reserved window of flow f_1 overlaps with the second reserved window of flow f_2 in the hyper-cycle.

temporal isolation of reserved windows of f_1 and f_2), and a disjunction (ensure window for f_1 is reserved before window of f_2 or vice versa). The equivalent constraints using plain ILP primitives are given in Appendix A.1.2.

If the flows have different transmission cycles, then we have to ensure that the reserved windows do not overlap in the entire hyper-cycle. For example, in Fig. 4.3 the reserved windows only overlap in the third transmission cycle of flow f_1 and the second transmission cycle of flow f_2, respectively. To prohibit that any overlap occurs ever, we introduce auxiliary sets $\mathcal{A}$ and $\mathcal{B}$, where $a \cdot \mathbf{t}_{\text{cycle}}[f_1]$ denotes the start of the a-th transmission cycle of flow f_1, and $b \cdot \mathbf{t}_{\text{cycle}}[f_2]$ denotes the start of the b-th transmission cycle of flow f_2, respectively, in the hyper-cycle. We then check that for all combinations of $a \in \mathcal{A}$ and $b \in \mathcal{B}$ no overlap occurs.

Joint Scheduling and Routing Constraints The constraints presented up to this point either restricted the values of $\underline{\mathbf{u}_{\text{link}}}$ or the values of $\underline{\mathbf{t}_{\text{offset}}}$, respectively. Now we link the temporal and spatial domain of the problem. The constraint

$$
\begin{aligned}
&\forall f \in \mathcal{F}, \forall n \in \mathcal{N}: \\
&\forall \ell_{\text{in}} \in \mathcal{L}_{\text{in},n}, \forall \ell_{\text{out}} \in \mathcal{L}_{\text{out},n}: \\
&\quad \text{if}(\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in}}] + \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out}}] \geq 2) \,\text{then} && (4.17)\\
&\qquad (\; \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{out}}] = \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}] + \mathbf{d}_{\text{hop}}[f] && (4.18)\\
&\qquad \text{or}\, \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{out}}] + \mathbf{t}_{\text{cycle}}[f] = \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}] + \mathbf{d}_{\text{hop}}[f] \;) && (4.19)
\end{aligned}
$$

specifies that the position of the reserved window on the outgoing link ℓ_{out} at a node n on the path of a flow f is shifted by $\mathbf{d}_{\text{hop}}[f]$ compared to the incoming link ℓ_{in} on the path, see Fig. 4.4.

There are two different cases regarding the reservation on the upstream link ℓ_{out}.

In variant 1, see Fig. 4.5, the reservation on the incoming link ℓ_{in} is early enough so that the reservation on the outgoing link ℓ_{out} is in the same transmission cycle on both links.

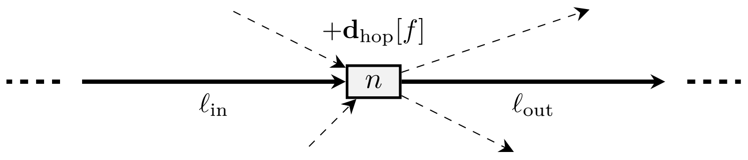
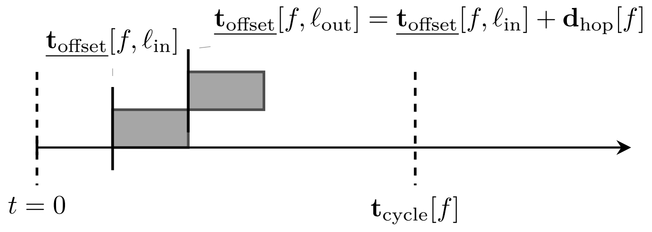


FIGURE 4.4 The constraints from Eq. (4.17)–Eq. (4.19) relate the reservations along the links of the path of flow f.



$\underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}]$
$\underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{out}}] = \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}] + \mathbf{d}_{\text{hop}}[f]$
$t = 0$
$\mathbf{t}_{\text{cycle}}[f]$

FIGURE 4.5 Illustration of the constraint from Eq. (4.18) where the reservation on the outgoing link ℓ_{out} is shifted by $\mathbf{d}_{\text{hop}}[f]$ compared to the reservation on the incoming link ℓ_{in} at node n.

Variant 2, see Fig. 4.6, occurs when the reservation on the incoming link ℓ_{in} is very close to the end of the transmission cycle. Then the packets which are transmitted in the reserved window via the incoming link ℓ_{in} in the transmission cycle k of flow f are transmitted on the outgoing link ℓ_{out} in the next transmission cycle with number $k + 1$. From a link-local perspective, the corresponding reservation on the outgoing edge ℓ_{out} is consequently earlier in the transmission cycle compared to the reservation on the incoming link ℓ_{in}.

The plain ILP equivalent of this constraint set is given in Appendix A.1.3.

Remark: Due to Eq. (4.17)–Eq. (4.19), the aforementioned cycles on the path which could possibly occur without Eq. (4.11) are such that it takes a packet an integer multiple of $\mathbf{t}_{\text{cycle}}[f]$ to traverse the circle. Hence, we can remove any such cycle in post-processing without jeopardizing the validity of the solution since the offsets of the reserved windows in each cycle remain the same.

Finally, to incorporate upper bounds on the per-flow end-to-end delay, we introduce the constraint

$$\forall f : \sum_{\ell \in \mathcal{L}} \mathbf{d}_{\text{hop}}[f] \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell] + \mathbf{d}_{\text{const}}[f] \leq \mathbf{t}_{\text{e2e}}[f] \tag{4.20}$$

with $\mathbf{d}_{\text{const}}[f]$ a constant delay component to account for the behavior of the source node and destination node of f. This constraint limits the number of links on the path of a flow f by

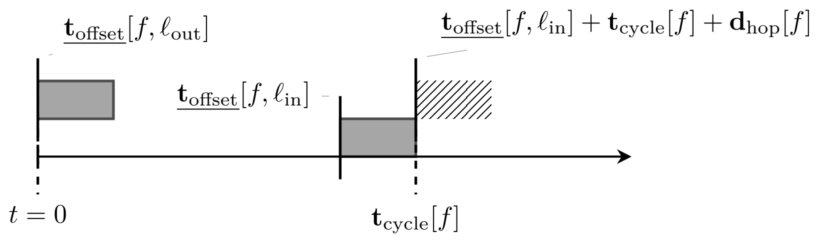

$\underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{out}}]$
$\underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}] + \mathbf{t}_{\text{cycle}}[f] + \mathbf{d}_{\text{hop}}[f]$
$\underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}]$
$t = 0$
$\mathbf{t}_{\text{cycle}}[f]$

FIGURE 4.6 Illustration of the constraint from Eq. (4.19) where the periodicity of the reservation has to be accounted for. The positions of the reservations relative to the start and end of the transmission cycle from the local perspectives of the respective link are filled in gray. The striped area right of $\mathbf{t}_{\text{cycle}}[f]$ is the position of the reservation on ℓ_{out} from the global perspective.

limiting the delay that can be accumulated. With $\mathbf{d}_{\text{const}}[*] = 0$, we can interpret Eq. (4.20) such that we “start the end-to-end delay timer” at the start of the reserved window and include a processing delay of t_{proc} at the destination node.

(Dummy) Objective In this section, our primary goal is to find a feasible solution, that is, a solution that satisfies all the constraints. Therefore, we do not have any objective for the ILP solver to maximize when computing such a solution, and we consider all feasible solutions for the joint routing and scheduling of the flows to be equally valuable.

Yet, in practice, the solver implementations or modeling frameworks expect an objective to be present in the ILP instances. Therefore, we introduce a placeholder objective function $\min \underline{q}$ on a bounded dummy variable $\underline{q}$. In Sec. 4.4.3, we will replace this dummy objective to actually influence the computation of the traffic plan.

4.2.1 Remark on the Complexity

We briefly sketch how we can reduce Bin-Packing [Gar+79] to the traffic planning problem. For any given Bin-Packing instance b with bin size $B \in \mathbb{N}$, finite set of elements $\mathcal{U}$ ($\forall u \in \mathcal{U} : s(u) \in \mathbb{N}$) with weight $s(u)$ for element u and K bins ($K \in \mathbb{N}$), we can create a special traffic planning instance j.

The network graph for j consists of source node n_{src} and destination node n_{dst}. With parallel links in the network graph, we can model each of the K bins by a link from n_{src} to n_{dst}. Then, for each element $u \in \mathcal{U}$, we add a flow request for a flow from n_{src} to n_{dst} with a reserved window $t_{\text{resv}} = s(u)$ and transmission cycle $t_{\text{cycle}} = B$. Iff j has a solution, then $\mathcal{U}$ can be partitioned into K disjoint sets $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K$ s.t. $\forall \mathcal{U}_i : \sum_{u \in \mathcal{U}_i} s(u) \leq B$. Note that we also could do without parallel links if we add K intermediate (auxiliary) nodes n_i and $2K$ directed edges (n_{src}, n_i) and (n_i, n_{dst}) with $i \in \{1 \ldots K\}$ between n_{src} and n_{dst}, that is, we replace each of the previously parallel links by a segment $n_{\text{src}}, n_i, n_{\text{dst}}$. However, then Eq. (4.7) has to apply only for the outgoing links at n_{src}, not for the intermediate nodes.

4.2.2 Evaluation

The ILP formulation in the previous section models the joint scheduling and routing problem in a fine-grained manner with per-flow per-link variables $\underline{\mathbf{t}_{\text{offset}}}$ and $\underline{\mathbf{u}_{\text{link}}}$. We evaluated a variety of problem instances to explore the scalability of this ILP-based formulation using synthetic joint scheduling and routing problem instances. A problem instance consists of a network graph and a set of flows defined by their parameters (source node, destination node, transmission cycle, packet size, reserved window size, and end-to-end deadline).

We explain the evaluation setup next and present some results regarding the solving performance which were originally published in [Fal+18].

Evaluation Scenarios

We generated the joint routing and scheduling problem instances in a two-step process. First, we generated a (random) network graph $\boldsymbol{G}_n$ according to either line, ring, scale-free, or random graph model followed by the generation of the flow set $\mathcal{F}$. We used the Python libraries graph-tool [Pei14] and networkX [Hag+08] to generate graphs according to the mentioned network models. For all topologies, we specified the number of nodes $|\mathcal{N}|$. The number of links $|\mathcal{L}|$ depends on the type of graph model.

The considered network models are of practical relevance (line graphs, ring graphs) and belong to fundamentally different classes of topologies (random scale-free graphs (Barabási-Albert network model), and random graphs (Erdős-Rényi network model)). Networks with a line or ring topology are often found in industrial scenarios, and the number of routes between two nodes is limited. The network topologies following the Barabási-Albert model [Pri07; Bar+99] have scale-free, power-law distributed vertex connectivity. In these tree-like structured graphs (see Fig. 4.7), there exists only one route between any two nodes. Since the graph generator for network graphs with Erdős-Rényi model does not guarantee to produce a connected random graph, we selected its largest connected component. Networks with this topology are usually meshed, and we usually cannot easily determine the exact number of routes between any two nodes.

The remaining network-related parameters are derived from a $1\,\frac{\text{Gbit}}{\text{s}}$ Ethernet network. For $1\,\frac{\text{Gbit}}{\text{s}}$ Ethernet networks, we set 1 time-unit in the ILP-formulation to correspond $1\,\mu\text{s}$. The numerical value for the propagation delay parameter t_{prop} is derived from links with lengths of $10\,\text{m}$ each. The propagation delay is given by $t_{\text{prop}} = 10\,\text{m}/\left(\frac{2}{3} \cdot c_{\text{light}}\right) = 0.05\,\mu\text{s}$. The propagation speed of two-thirds of the speed of light $\left(\frac{2}{3} \cdot c_{\text{light}}\right)$ is, for example, in the range

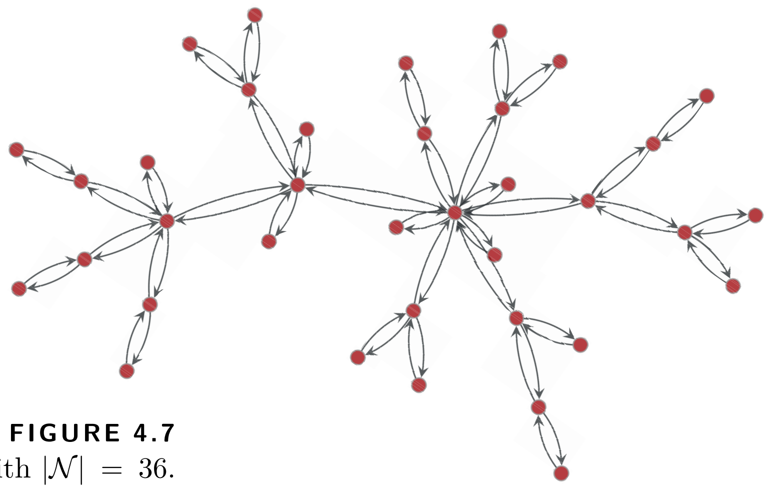


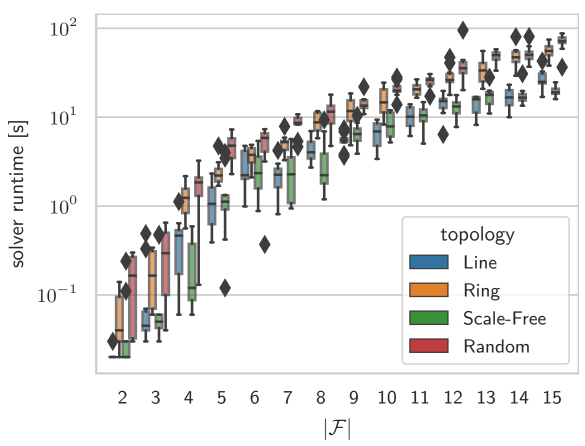
FIGURE 4.7
Example of a scale-free graph with $|\mathcal{N}| = 36$.

of CAT6-cables. The processing delay of the switches is set to $t_{\text{proc}} = 5\,\mu\text{s}$. This value for the processing delay can be expected from state-of-the-art Ethernet switches [Dür+14]. For all flows, t_{pkt} has the same value and is derived from a packet size of 368 B, which results in $t_{\text{trans}} = t_{\text{pkt}} = 368\,\text{B}/1\,\frac{\text{Gbit}}{\text{s}} = 2.944\,\mu\text{s}$. While this packet size can be considered small in terms of the Ethernet MTU (Ethernet allows for packets of roughly four times this size), closed-loop networked control applications or high-priority traffic [Ste+15] and traditional bus systems [PRO+16] often utilize smaller packet sizes. We chose a packet size of 368 B since it is big enough to accommodate a generous protocol overhead and a payload consisting of, for example, several timestamps and numerical sensor values. Consequently, the value of $\mathbf{d}_{\text{hop}}[f] = d_{\text{hop}}$ is the same for every flow and is set to $\mathbf{d}_{\text{hop}}[f] = d_{\text{hop}} = 8\,\mu\text{s} \approx t_{\text{pkt}} + t_{\text{prop}} + t_{\text{proc}}$.

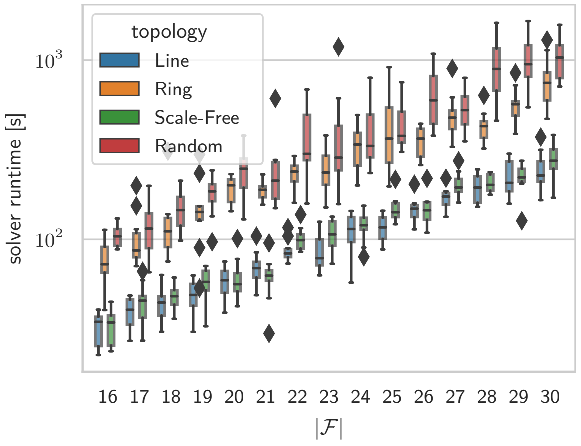
After generating a network instance, for each flow f in the $\mathcal{F}$, we select two nodes, the source node $\mathbf{n}_{\text{src}}[f]$ and the destination node $\mathbf{n}_{\text{dst}}[f]$, respectively, from the node set $\mathcal{N}$ uniformly at random. We assume that the networked applications request reservation windows t_{resv} which are approximately integer multiples of t_{pkt}. The reserved window size $\mathbf{t}_{\text{resv}}[f]$ for each flow f is drawn from $\{3, 6, 9\}\mu\text{s}$ uniformly at random. We assume transmission frequencies ranging from $125\,\text{Hz} = 1$ transmission/$8000\,\mu\text{s}$ to $20\,\text{kHz} = 1$ transmission/$50\,\mu\text{s}$. This means the values of $\mathbf{t}_{\text{cycle}}$ range from $50\,\mu\text{s}$ to $8\,\text{ms}$. The end-to-end delay is set depending on the network topology using the relation $\mathbf{t}_{\text{e2e}}[*] = a \cdot d_{\text{hop}} + 1$ with $a \in \{1, \ldots, |\mathcal{N}|\}$. By default, we used $\mathbf{t}_{\text{e2e}}[*] = |\mathcal{N}| \cdot d_{\text{hop}} + 1$ which permits routes that visit every node in the network.

Evaluation Setup

We separated the instantiation of a concrete ILP instance for a given problem instance, which is rendered to a file, and solving the ILP itself. The solver was invoked via the vendor-provided binary and then read the ILP instance from the file.



A 2-15 flows, runtime limit is set to 30 min.



B 16-30 flows, runtime limit is set to 60 min.

FIGURE 4.8 Box plot of solver runtimes for problem instances with varying number of flows, fixed number of nodes $|\mathcal{N}| = 8$, transmission cycles $\in \{1000, 2000, 4000, 8000\}$

For the measurements in this section, we used the Linux version of IBM ILOG CPLEX Optimization Studio 12.8.0 [IBM17], the modeling language OPL (using oplrun to render the ILP to .mps-files and .lp-files), and CPLEX as solver. With this tool-chain, the constraints from Sec. 4.2 involving the sets $\mathcal{L}_{\text{in},n}$, and $\mathcal{L}_{\text{out},n}$ are implemented using sparse matrices derived from the $\boldsymbol{G}_n$. In particular, we use a link-link adjacency matrix $\mathbf{A}_{LL}$ and a node-link incidence matrix $\mathbf{B}_{NL}$ so that we can express the routing constraints and the joint scheduling and routing constraints by summing over matrix components, see Appendix A.1.4. For the reasons mentioned, we also skipped the constraint from Eq. (4.11) in our ILP implementation.

The "compilation" and solving of the ILP instances were performed on a 4-socket compute node with four Intel Xeon E7-4850 v4 CPUs with a nominal clock speed of 2.1 GHz and a total amount of 1 TB RAM running Linux kernel version 4.15.15.

Evaluation Results

We evaluate the effects of various problem properties (number of flows, network graph, transmission frequency) on the solver runtime in the following.

Number of Flows For the first set of problem instances, we vary the number of flows and keep the number of nodes $|\mathcal{N}| = 8$ in the network fixed. The values for $\mathbf{t}_{\text{cycle}}[f]$ are drawn from $\{1000, 2000, 4000, 8000\}$ uniformly at random. For problem instance with 2-16 flows (see Fig. 4.8A), the solver had a runtime 30 min and always terminated within the given time limit.

In Fig. 4.8B, we plot a second batch of similar problem instances with more flows (16-30 flows), and the solver runtime limit was increased to 60 min. Despite the increased runtime lime,

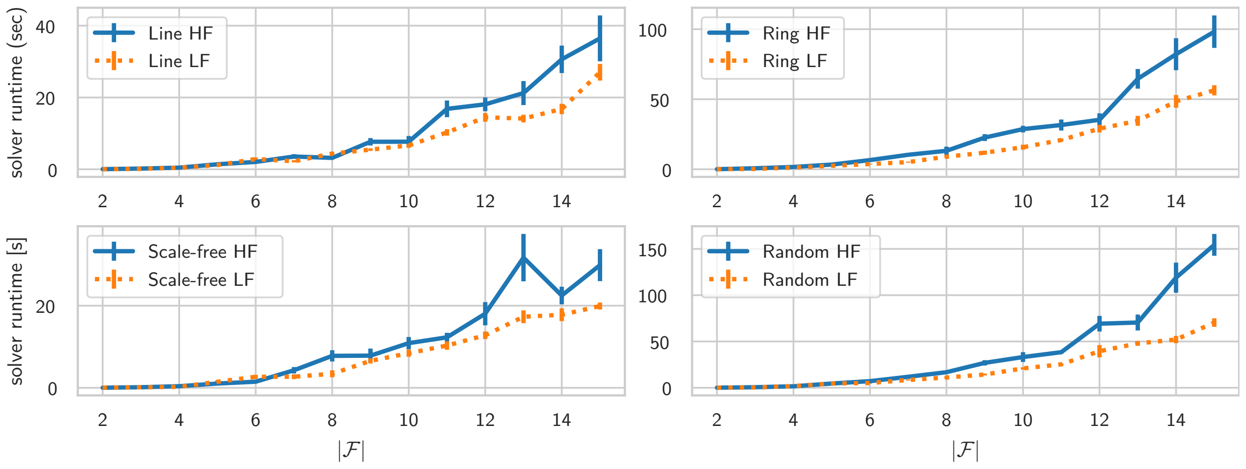


FIGURE 4.9 Comparison of runtimes for problem instances with varying number of flows, $|\mathcal{N}| = 8$, transmission cycles for HF $\in \{50, 100, 200, 400, 800\}$, and for LF $\in \{1000, 2000, 4000, 8000\}$, and runtime limit of 30 min.

we could observe two problem instances for networks with random topology where the solver did not produce a result before the timeout(at scenarios with 21 flows and 30 flows, respectively).

In both cases (Fig. 4.8A, Fig. 4.8B) we considered 10 problem instances per topology and number of flows. We observe an increase in runtime with an increasing number of flows for our problem instances. From a technical perspective, this is expected, because more flows will result in larger ILP instances. From the perspective of the traffic planning problem, this behavior conforms to our intuition, too, because we expect that putting more flows in the network while keeping the network size constant (in terms of nodes) increases the load and makes it harder to find schedules.

Transmission Frequency Regarding the latter aspect, we can also change the frequency of the transmissions to vary the expected load on the network. The frequency of the transmissions (the inverse of the transmission cycle $\mathbf{t}_{\text{cycle}}$) does not influence the number of the constraints. It only affects the numerical values in the constraints. Therefore, we evaluate a second set of problem instances with similar parameters as the problem instances presented in Fig. 4.8A, but this time the cycle times are much smaller and drawn from set $\mathbf{t}_{\text{cycle}}[f] \in \{50, 100, 200, 400, 800\}$ uniformly at random. In the following, we refer to these problem instances with $\mathbf{t}_{\text{cycle}}[f] \in \{50, 100, 200, 400, 800\}$ as high-frequency (HF) flows in contrast to the low-frequency (LF) flows with $\mathbf{t}_{\text{cycle}}[f] \in \{1000, 2000, 4000, 8000\}$.

The average runtimes for problem instances with low transmission frequencies (LF) are plotted with the dashed lines in Fig. 4.9. The solid lines in Fig. 4.9 are the average runtimes for the equally sized set of problem instances with higher transmission frequencies (HF). We

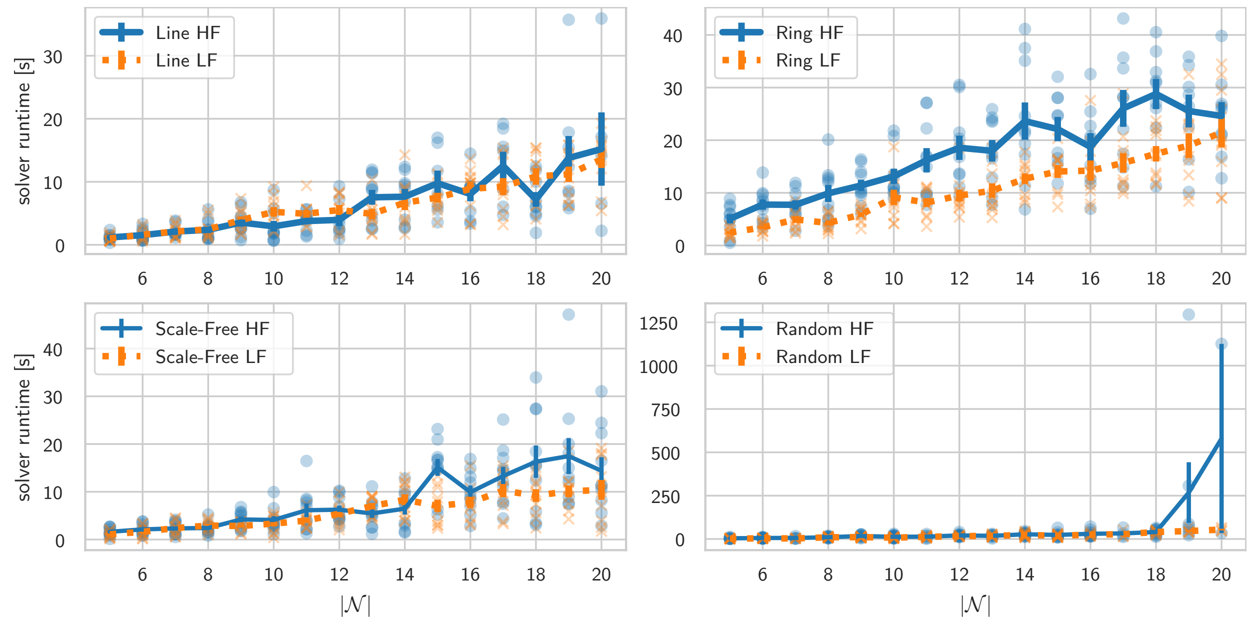


FIGURE 4.10 Comparison of runtimes for problem instances with varying number of nodes, $|\mathcal{F}| = 7$, transmission cycles for HF $\in \{50, 100, 200, 400, 800\}$ and for LF $\in \{1000, 2000, 4000, 8000\}$, and runtime limit of 30 min.

observe that the solver requires increasingly more time to solve problem instances with higher transmission frequencies compared to problem instances with lower transmission frequencies the more flows have to be considered.

Since the range of the size of the reserved windows is fixed ($\mathbf{t}_{\text{resv}}[*] \in \{3, 6, 9\}$) while the transmission cycle was varied, we effectively changed the fraction of the transmission period that is reserved. That supports the observations that a higher traffic demand increases the runtime since with a higher fraction of reservations per cycle there are fewer possibilities to place the reservations of different flows.

Network Graph Size Next, we keep the number of flows fixed $|\mathcal{F}| = 7$ and vary the number of nodes. As before, we also vary the transmission frequencies. In Fig. 4.10, we compare the average solver runtimes for the problem instances with low and high transmission frequencies. The dashed lines in Fig. 4.10 are the average runtimes for problem instances with low transmission frequencies (LF) with $\mathbf{t}_{\text{cycle}}[f]$ drawn from $\in \{1000, 2000, 4000, 8000\}$. Solid lines in Fig. 4.10 correspond to the averages of problem instances with higher transmission frequencies (HF) with $\mathbf{t}_{\text{cycle}}[f] \in \{50, 100, 200, 400, 800\}$

Overall, we again observed increasing runtimes for an increasing number of nodes in the network. There are a few irregularities, that is, decreasing runtimes, for line and scale-free graphs visible in Fig. 4.10. These may be introduced by the random placement of the source

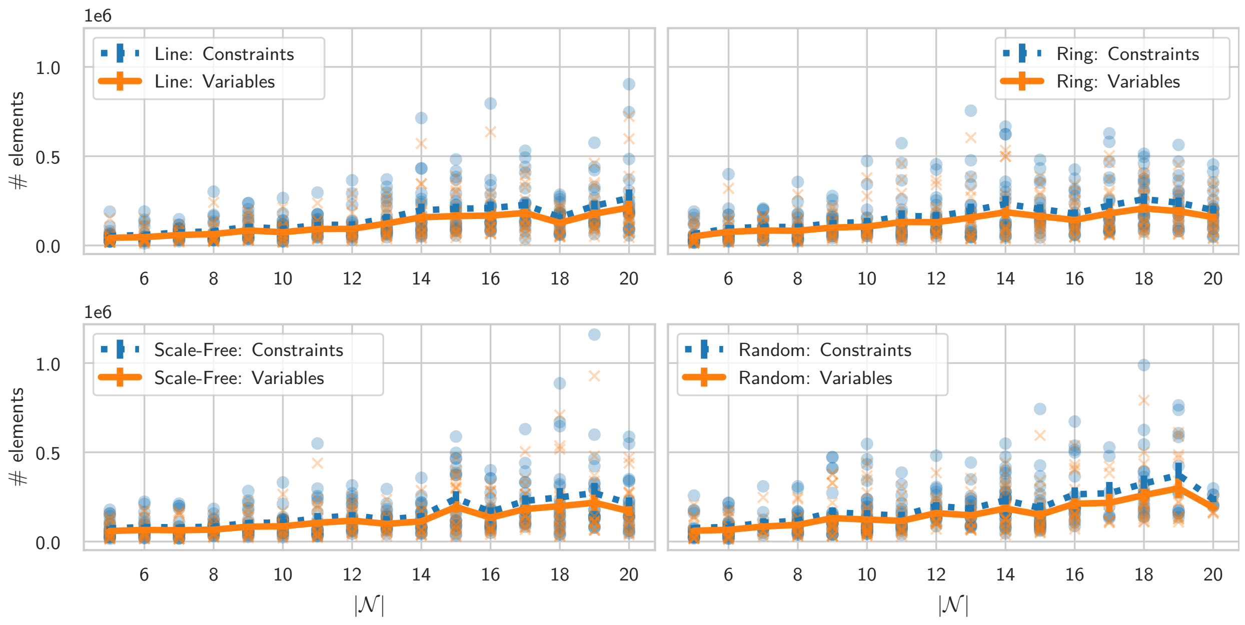


FIGURE 4.11 Average number of ILP constraints and ILP variables for problem instances with varying number of nodes (cf. Fig. 4.10).

nodes and destination nodes or due to scenarios where the solver quickly discovered infeasibility. Striking is the steep rise of the runtime in HF problem instances with random topology for 19 and 20 vertices where runtimes up to more than 1000 s were observed.

For reference, Fig. 4.11 shows the average size of the ILP instances.

As expected, the ILP instances become larger, since the number of constraints depends (via $\mathbf{B}_{NL}$ and $\mathbf{A}_{LL}$) on the size of the network graph $\boldsymbol{G}_n$. Additionally, in a larger graph with random placement of the flow origins and destinations, the routes for the flows can be longer which increases the number of edges where the scheduling constraints have to be applied.

We can observe that the high runtimes from Fig. 4.10 are unlikely to be caused by the size of the ILP alone, since, for example, ILP instances with 20 nodes resulted in overall comparable ILP dimensions, and yet runtimes are different. Instead, this behavior could be due to the NP-hard character of the problem where particular "difficult" problem instances require much more runtime to solve than the average case.

Topology In Fig. 4.8B (and to a lesser degree in the other plots) it is also noticeable that the runtime for ring and random topology almost consistently exceeds those of the remaining topologies, and the runtime of problem instances with random topology exceeds those of the ring topology.

In Fig. 4.12, we plot the average dimension of the ILP (in terms of the number of constraints and variables) for the problem instances from Fig. 4.8B and Fig. 4.9 where we varied the number

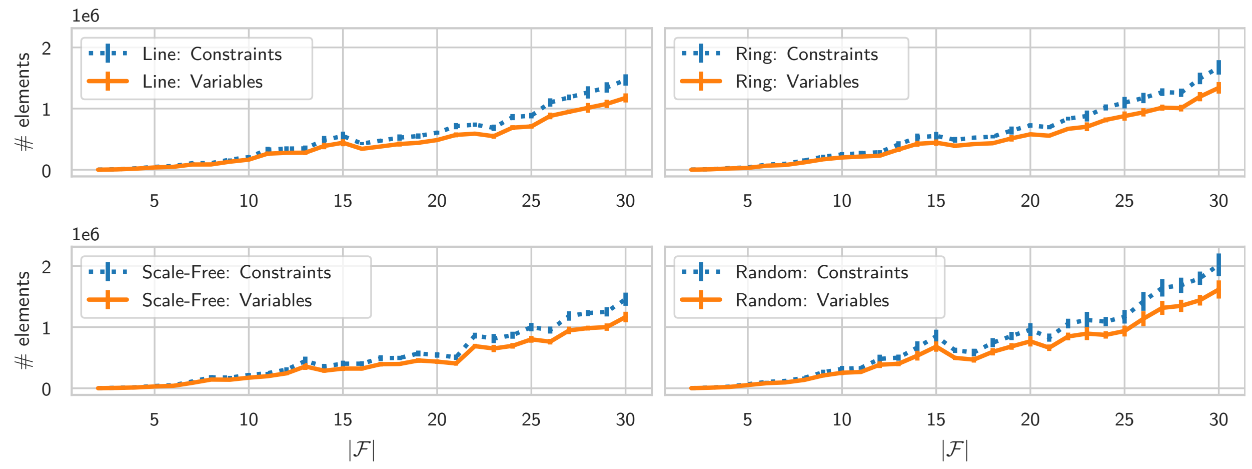


FIGURE 4.12 Average number of ILP constraints and ILP variables for problem instances with varying number of flows (cf. Fig. 4.8B and Fig. 4.9).

of flows.

Our results indicate that more routing options also increase the solver runtime since we observed that instances with ring topology took longer to solve compared to instances with line topology and scale-free topology in relation to the size of the ILP instances. Other than in ring topologies where we know that there exist two paths between any pair of nodes, there exists only a single path between any pair of nodes in line topologies and scale-free topologies.

We want to remark that we encountered a series of infeasible problem instances in these topologies without routing alternatives for HF problem instances with $\mathbf{t}_{\text{cycle}} \in \{50, 100, 200, 400, 800\}$ in our evaluations. In contrast, the solver produced a feasible solution for all ILP instances with ring topologies, and only a single problem instance with random topology was infeasible.

Evaluation Summary

In our evaluations, we made the following three main observations: Firstly, the solver runtime is more sensitive to an increase in the number of flows than to an increase of the network graph size. In our evaluations, additional flows increased the dimension (number of variables and number of constraints) more than increasing the network graph size which is one reason for this behavior. Secondly, the solver runtime is influenced by how much of the transmission cycle is occupied by the reserved window. Thirdly, the graph topology does impact the solver runtime. In our measurements, the solver runtime depended on how many paths exist between any two nodes which can be interpreted such that additional routing decisions slow down the solving process.

4.3 Joint Scheduling and Path Selection

In our evaluation scenarios from Sec. 4.2.2, the runtime to solve ILP instances varied for the different network topologies. This motivated another ILP formulation where we do not leave it up to the solver to *compute* a valid route on a per-link granularity. Instead, we pre-compute a set of so-called candidate paths for each flow and let the solver *select* a suitable one. Therefore, we have ILP-based *joint scheduling and path selection*. From an architectural perspective, we assume that this candidate path computation is performed as part of traffic planning. Alternatively, we could think of an architecture where the candidate paths are part of the flow requests and thus are provided by the issuer of the flow requests.

Originally, the ILP formulation for joint routing and path selection has been introduced in [Fal+19a] in conjunction with complemental flows. It was also re-used in [Fal+20], as an exemplary ILP-based reference to evaluate the conflict-graph-based traffic planning method. Therefore, we defer the evaluation of the ILP formulation for joint scheduling and path selection to the later sections, Sec. 4.4 and Sec. 5.2.

However, since the ILP formulation for joint scheduling and path selection provides an alternative for computing feasible traffic plans in its own right, we chose to apportion it with its own section.

4.3.1 Candidate Paths

Next, we explain candidate paths, which will also resurface in the context of conflict-graph-based traffic planning in Sections 5.1 to 5.3, in more detail.

Encoding just the constraints of a valid route as done in Sec. 4.2 with per-link granularity provides all the freedom with regard to the routing decision, but we have to pay for this with many variable assignments which the solver has to discard because they do not even result in valid routes. In contrast, with candidate paths, we include for each flow a set of sensible routing options, so-called candidate paths, in the traffic planning process. This means we take the burden of computing valid paths from the generic solver and use dedicated routing algorithms for the task of computing valid paths.

In theory, we could capture all routing options, for example, by using some breadth-first-search technique to compute for each flow a candidate path set that contains every loop-free path. Since the number of loop-free paths in a network of finite size is finite, there is only a finite amount of candidate paths for each flow, to begin with.

Without losing any feasible traffic plan, we can also safely discard all paths between the source node and the destination node of a flow that are "too long" in the sense that the end-to-end delay bound would be exceeded. As explained in Sec. 3.4, zero-queuing provides a simple relation between path length and end-to-end delay t_{e2e}, which we can exploit here. It follows that we do not need explicit end-to-end delay constraints if we ensure that all candidate paths are short enough to meet the end-to-end delay bound of the respective flow.

Yet, depending on the scenario, for example, in highly meshed networks and with large end-to-end delay bounds, we might still end up with impractical many candidate paths. Therefore, we usually upper-bound the number of candidate paths and use an algorithm, for example, k-shortest paths algorithms or path enumeration techniques such as breadth-first search with a path-length cut-off, which, by design, are more likely to produce “short enough” paths.

In summary, the idea of traffic planning with candidate paths is to pre-compute a set of sensible candidate paths for each flow such that only valid routing choices need are considered from that point on. Each candidate path must not only be loop-free but is also restricted in length by the end-to-end delay constraint for each flow.

4.3.2 Integer Linear Program

Next, we present a second ILP formulation with joint scheduling and path selection, which uses candidate paths.

Enhanced Modeling of Temporal Properties

Again, we use a discrete representation of time. For each flow, we represent each transmission cycle as a sequence of discrete cells. A cell denotes a time interval with length τ. To find a suitable mapping of cell length τ to seconds, we can for example use the greatest common divisor of all values for t_{pkt} and the delay parameters of the network.

We can use cells to model temporal properties by assigning certain attributes or properties to them. In this section, each cell has an attribute that represents whether a cell is part of a reserved window. We will make extended use of this additional expressiveness in Sec. 4.4 to show how to compute an optimized traffic plan for complementary traffic flows with time-varying traffic metrics.

ILP Parameters

Before we express the ILP for joint-scheduling and path-selection, we explain how we map the traffic planning parameters (see Tab. 4.2) to the ILP.

For each flow f, we may have a different number of candidate paths. Each candidate path is assigned a path index. We store the available path indices for flow f denoted by $\mathbf{n}_{\text{path}}[f] = (0, 1, 2, \ldots)$ with $f \in \mathcal{F}$. $\mathbf{P}_{\text{cand}}[f, \pi] = (\ell_1, \ell_2, \ell_3, \ldots)$ denotes the candidate path for flow $f \in \mathcal{F}$ with the path index $\pi \in \mathbf{n}_{\text{path}}[f]$. $\mathbf{P}_{\text{cand}}[f, \pi] = (\ell_1, \ell_2, \ell_3, \ldots)$ is a sequence of links $\ell_i \in \mathcal{L}$ from the source node to the destination node of flow f. As a notational shortcut for all links that appear on any candidate path of flow f, we define $\mathcal{L}_f = \bigcup_{\pi \in \mathbf{n}_{\text{path}}[f]} \mathbf{P}_{\text{cand}}[f, \pi]$.

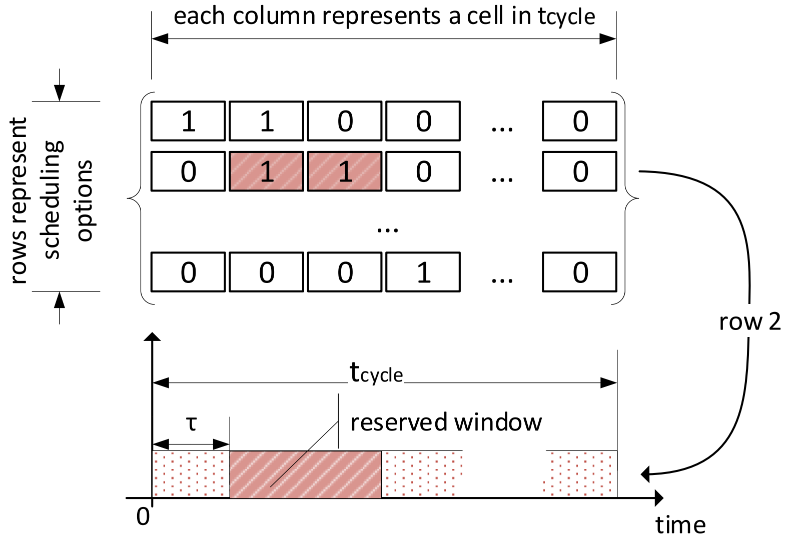


FIGURE 4.13 Reserved cells in the transmission cycle are expressed by reservation matrices where rows correspond to scheduling options and columns to different intervals in the transmission cycle.

This means once we know the candidate paths, we only need to consider the links in $\mathcal{L}_f$ when scheduling reserved time intervals for flow f.

We express reservations for each flow on a per-cell level with the help of reservation matrices. In the reservation matrix $\mathbf{m}_{\text{resv}}[f]$ for flow $f \in \mathcal{F}$, each row is associated with a specific scheduling option. Each column in $\mathbf{m}_{\text{resv}}[f]$ represents a cell in the transmission cycle where the column index is related to the position of that cell relative to the start of the cycle, see Fig. 4.13.

The components of the reservation matrix are binary values where $\mathbf{m}_{\text{resv}}[f][i, j] = 1$—the value in the ith row at the jth column—indicates that the jth cell is reserved if we choose the ith scheduling option. Conversely, a value of zero for $\mathbf{m}_{\text{resv}}[f][i, j]$ indicates that the jth cell is not reserved if we choose the ith scheduling option.

Theoretically, the reservation matrix for each flow supports arbitrary patterns. However, we restrict ourselves to quasi-feature-parity with the ILP from the previous section: each scheduling option corresponds to a specific offset—now in multiples of τ. This means each row represents a reserved window of length $\mathbf{t}_{\text{resv}}[f]$ (given in multiples of τ, too) for a different offset.

In this case, the reservation matrix $\mathbf{m}_{\text{resv}}[f] \in \{0, 1\}^{\mathbf{t}_{\text{cycle}}[f] \times \mathbf{t}_{\text{cycle}}[f]}$ for each flow $f \in \mathcal{F}$ can be constructed iteratively from the flow parameters with the roll-operator.

The operator $\text{roll}(\mathbf{x}, i)$ rotates the components of the row vector $\mathbf{x}$ by i positions to the right, overflowing components being appended on the left. The first $\mathbf{t}_{\text{resv}}[f]$ cells in $\mathbf{m}_{\text{resv}}[f][0, *]$ have value 1—indicating the reserved window—and the remaining cells have value 0.

The remaining rows are then defined as

$$\mathbf{m}_{\text{resv}}[f][c_{\text{offset}}, *] = \text{roll}\left(\mathbf{m}_{\text{resv}}[f][0, *], c_{\text{offset}}\right) \tag{4.21}$$

TABLE 4.2 Parameters for ILP formulation for joint scheduling and path selection.

$\boldsymbol{G}_n$	network graph
$\mathcal{N} = \{0, 1, 2, \ldots\} \subset \mathbb{N}$	set of nodes
$\mathcal{L} = \{0, 1, 2, \ldots\} \subset \mathbb{N}$	set of links
$\mathcal{F} = \{0, 1, 2, \ldots\} \subset \mathbb{N}$	set of flows
$\mathbf{n}_{\text{src}} \in \mathcal{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{n}_{\text{src}}[f]$ is the source node of flow f
$\mathbf{n}_{\text{dst}} \in \mathcal{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{n}_{\text{dst}}[f]$ is the destination node of flow f
$\mathbf{t}_{\text{cycle}} \in \mathbb{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{t}_{\text{cycle}}[f] \cdot \tau$ is the cycle time for flow $f \in \mathcal{F}$
$\mathbf{t}_{\text{pkt}} \in \mathbb{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{t}_{\text{pkt}}[f] \cdot \tau$ is the time it takes to transmit a packet of f
$\mathbf{t}_{\text{resv}} \in \mathbb{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{t}_{\text{resv}}[f] \cdot \tau$ is the length of the reserved window for flow f
$\mathbf{t}_{\text{e2e}} \in \mathbb{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{t}_{\text{e2e}}[f] \cdot \tau$ is the maximal end-to-end delay for flow f
$\mathbf{d}_{\text{hop}} \in \mathbb{N}^{\lvert\mathcal{F}\rvert}$	$\mathbf{d}_{\text{hop}}[f] \cdot \tau$ is per-hop delay for packets of flow f
$t_{\text{hyper}} = \text{lcm}\left(\mathbf{t}_{\text{cycle}}\right)$	least common multiple of all cycle times in multiples of τ

with

$$c_{\text{offset}} \in \left\{0, 1, 2, \ldots, \left(\mathbf{t}_{\text{cycle}}[f] - 1\right)\right\}.$$

ILP Variables

We use a binary variable $\underline{\boldsymbol{\pi}_{\text{path}}}[f, \pi] \in \{0, 1\}$ to indicate whether path $\pi \in \mathbf{n}_{\text{path}}[f]$ is selected as route for flow $f \in \mathcal{F}$. We also use the binary variable $\underline{\mathbf{u}_{\text{link}}}[f, \ell] \in \{0, 1\}$ with $f \in \mathcal{F}$ and $\ell \in \mathcal{L}$. Although this time, $\underline{\mathbf{u}_{\text{link}}}[f, \ell]$ is more of a helper variable that is used to map the selected candidate path to the respective links on that candidate path.

The variables and the method which we use to encode the scheduling decision in the ILP with joint scheduling and path selection differ from Sec. 4.2. Instead of directly representing the offset for each flow in an (integer) variable, we use binary variables $\underline{\mathbf{s}}[\ell, f, c] \in \{0, 1\}$ to represent the selected scheduling option with $\ell \in \mathcal{L}$, $f \in \mathcal{F}$, and c is in the range of row-indices of $\mathbf{m}_{\text{resv}}[f]$.

If $\underline{\mathbf{s}}[\ell, f, c] = 1$, then the c-th scheduling option–which in turn corresponds to the c-th row of $\mathbf{m}_{\text{resv}}[f]$—is selected for flow f on link ℓ, see Fig. 4.14. In our case, each row in $\mathbf{m}_{\text{resv}}[f]$ (each scheduling option) corresponds to a specific offset of the reserved window. If we consider

$$\underline{\mathbf{s}}[\ell, f, *] = \begin{bmatrix} \underline{\mathbf{s}}[\ell, f, 0] & \underline{\mathbf{s}}[\ell, f, 1] & \underline{\mathbf{s}}[\ell, f, 2] & \cdots \end{bmatrix}$$

as another row vector, then the offset for the reserved window of flow f on link ℓ is equal to the (zero-based) index c of that variable $\underline{\mathbf{s}}[\ell, f, c]$ which has value 1 multiplied by τ.

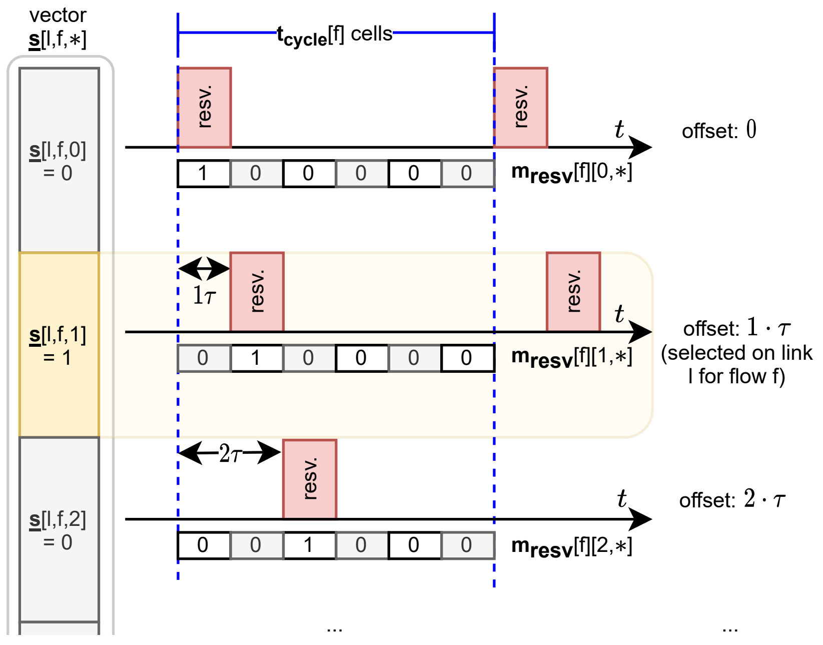


FIGURE 4.14 Rows of the reservation matrix $\mathbf{m}_{\text{resv}}[f]$ represent different positions of the reserved window. Variable $\underline{\mathbf{s}}[\ell, f, c]$ indicates whether offset $c \cdot \tau$ is selected for flow f on edge ℓ.

Path Selection Constraints

Since we have candidate paths, the path selection constraint

$$\forall f \in \mathcal{F} : \sum_{\pi \in \mathbf{n}_{\text{path}}[f]} \underline{\boldsymbol{\pi}_{\text{path}}}[f, \pi] = 1 \tag{4.22}$$

ensures that for each flow one candidate path out of the set of all candidate paths is selected. Consequently, we can easily retrieve the route of f from the ILP solution.

Next, we want to set $\underline{\mathbf{u}_{\text{link}}}[f, \ell]$ to one for all links on the selected candidate path of flow f. We define the constraint

$$\forall f \in \mathcal{F} : \forall \ell_f \in \mathcal{L}_f : \underline{\mathbf{u}_{\text{link}}}[f, \ell_f] = \sum_{\pi \in \mathcal{P}_{\text{cand},\ell_f}} \underline{\boldsymbol{\pi}_{\text{path}}}[f, \pi] \tag{4.23}$$

with

$$\mathcal{P}_{\text{cand},\ell_f} = \{\pi \in \mathbf{n}_{\text{path}}[f] | \ell_f \in \mathbf{P}_{\text{cand}}[f, \pi]\}. \tag{4.24}$$

In Eq. (4.23), ℓ_f is a link that appears in at least one candidate path for flow f—different candidate paths may share identical sub-paths—, and $\mathcal{P}_{\text{cand},\ell_f}$ is the set of all path indices where the associated candidate path contains link ℓ_f.

Scheduling Constraints

As in Eq. (4.7), we restrict the selectable offsets by forcing some entries of $\underline{\mathbf{s}}[\ell, f, c]$ to zero with

$$\begin{aligned} &\forall f \in \mathcal{F} : \forall \ell_f \in \mathcal{L}_f : \\ \forall c \in \{c \in \{0, 1, 2, \dots (\mathbf{t}_{\text{cycle}}[f] - 1)\} | c > (\mathbf{t}_{\text{cycle}}[f] - \mathbf{t}_{\text{resv}}[f])\} &: \underline{\mathbf{s}}[\ell_f, f, c] = 0. \end{aligned} \tag{4.25}$$

If we use $\mathcal{L}_f = \bigcup_{\pi \in \mathbf{n}_{\text{path}}[f]} \mathbf{P}_{\text{cand}}[f, \pi]$, Eq. (4.25) is equivalent to the restriction from Eq. (4.7) where the reserved window may not cross the transmission cycle bounds on every link that is part of any candidate path of flow f.

However, the representation of the reserved windows with the reservation matrix allows replacing $\mathcal{L}_f$ in Eq. (4.25) with

$$\mathcal{L}'_f = \{\ell \in \mathcal{L} | \, \mathrm{n}_{\text{out}}(\ell) = \mathbf{n}_{\text{src}}[f]\}. \tag{4.26}$$

With $\mathcal{L}'_f$ we restrict only the phase. In other words, only on the outgoing links of the source nodes of the flows, the reserved windows must not cross over the transmission cycle boundaries.

Next, we present the scheduling constraint in matrix-notation

$$\forall \ell \in \mathcal{L} : \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \leq \sum_{f_\ell \in \mathcal{F}_\ell} (\mathbf{M}_{\text{resv}}[f_\ell]^\intercal \cdot \underline{\mathbf{s}}[\ell, f_\ell, *]^\intercal) \leq \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{4.27}$$

with

$$\mathcal{F}_\ell = \{f \in \mathcal{F} | \exists \pi \in \mathbf{n}_{\text{path}}[f] : \ell \in \mathbf{P}_{\text{cand}}[f, \pi]\} \tag{4.28}$$

denoting the set of all flows whose candidate paths contain link ℓ. The matrix $\mathbf{M}_{\text{resv}}[f_\ell]$ is defined as

$$\mathbf{M}_{\text{resv}}[f_\ell] = \underbrace{\begin{bmatrix} \mathbf{m}_{\text{resv}}[f_\ell] & \cdots & \mathbf{m}_{\text{resv}}[f_\ell] \end{bmatrix}}_{\frac{t_{\text{hyper}}}{\mathbf{t}_{\text{cycle}}[f_\ell]} \text{ times}}. \tag{4.29}$$

This means we can interpret $\mathbf{M}_{\text{resv}}[f_\ell]$ as the extension of $\mathbf{m}_{\text{resv}}[f_\ell]$ to cover the whole hyper-cycle by means of horizontal concatenation.

The term $\underline{\mathbf{s}}[\ell, f_\ell, *] \cdot \mathbf{M}_{\text{resv}}[f_\ell]$ yields a row vector indicating which cells in the hyper-cycle are reserved for flow f_ℓ on link ℓ and the selected phase which is encoded in $\underline{\mathbf{s}}[\ell, f_\ell, *]$. This idea is illustrated in Fig. 4.15 for two flows. Since element-wise inequalities are more commonly expressed for column-vectors, we use the relation $(AB)^\intercal = B^\intercal A^\intercal$ and sum over all flows on link ℓ. The expression $\sum_{f_\ell} (\mathbf{M}_{\text{resv}}[f_\ell]^\intercal \cdot \underline{\mathbf{s}}[\ell, f_\ell, *]^\intercal)$ thus returns a column vector which indicates for

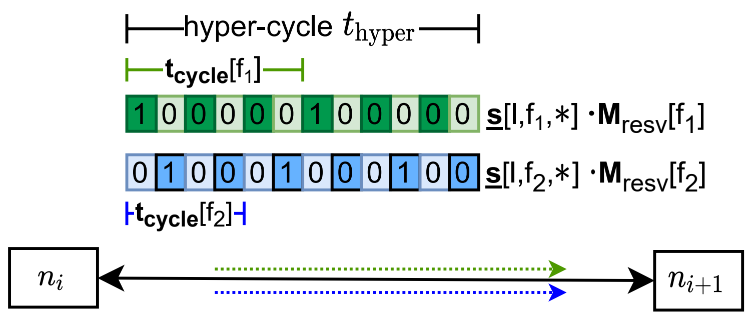

hyper-cycle t_{hyper}
$t_{\text{cycle}}[f_1]$
1 0 0 0 0 0 1 0 0 0 0 0
$\underline{s}[l,f_1,*] \cdot M_{\text{resv}}[f_1]$
0 1 0 0 0 1 0 0 0 1 0 0
$\underline{s}[l,f_2,*] \cdot M_{\text{resv}}[f_2]$
$t_{\text{cycle}}[f_2]$
n_i
n_{i+1}

FIGURE 4.15
Each cell in the hyper-cycle can be reserved for at most one flow.

each cell in the hyper-cycle how many flows would reserve it. The right-hand side of Eq. (4.27) adds the constraint that for link ℓ each cell in the hyper-cycle is reserved for at most one flow. Note that the link-local schedule for the reserved windows at ℓ can be recovered from the solution by evaluating this sum term from Eq. (4.27).

Equation (4.27) ensures temporal isolation for arbitrary patterns of reserved cells. More specifically, wrap-arounds of the reserved window at transmission cycle boundaries are supported "for free" by the ILP formulation in this section. In contrast, the approach from Sec. 4.2 would require more complex scheduling constraints due to the additional disjunctions introduced by non-continuous reserved intervals.

Joint Scheduling and Path Selection Constraints

We need one offset for the reserved window on each link on the route of a flow in the traffic plan.

Consequently, $\underline{\mathbf{s}}[\ell, f, *]$ contains at most one component with $\underline{\mathbf{s}}[\ell, f, c] = 1$. If ℓ is a link on the selected candidate path, exactly one entry in $\underline{\mathbf{s}}[\ell, f, *]$ has value one. If ℓ is not a link on the selected candidate path, all entries of $\underline{\mathbf{s}}[\ell, f, *]$ have value zero. This is expressed with the constraint

$$\forall f \in \mathcal{F} : \forall \ell_f \in \mathcal{L}_f : \sum_{c_{\text{shift}} \in \mathbf{n}_{\text{prd}}[f]} \underline{\mathbf{s}}[\ell_f, f, c_{\text{shift}}] = \underline{\mathbf{u}_{\text{link}}}[f, \ell_f]. \tag{4.30}$$

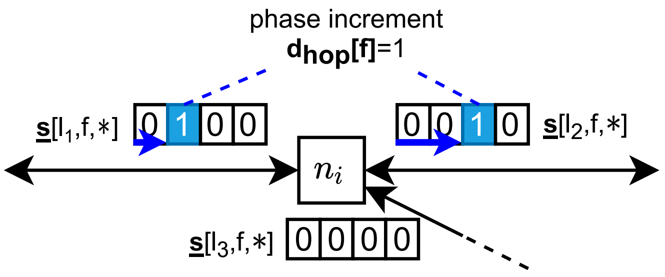
To express the relation of the selected phases on subsequent links of the candidate path, we first define a helper matrix $\mathbf{m}_{\text{shift}}[f] = \text{roll}\left(\mathbf{I}_{\mathbf{t}_{\text{cycle}}[f] \times \mathbf{t}_{\text{cycle}}[f]}, \mathbf{d}_{\text{hop}}[f]\right)$ where $\text{roll}(\mathbf{I}, x)$ rotates the columns of the identity matrix $\mathbf{I}$ with $\mathbf{t}_{\text{cycle}}[f]$ rows and columns by x positions to the right. The constraints

$$\forall f \in \mathcal{F} : \forall \pi \in \mathbf{n}_{\text{path}}[f] : (\ell_{\text{in}}, \ell_{\text{out}}) \subseteq \mathbf{P}_{\text{cand}}[f, \pi] :$$

$$\mathbf{m}_{\text{shift}}[f]^\intercal * \underline{\mathbf{s}}[\ell_{\text{in}}, f, *]^\intercal - \left(\mathbf{1} - \mathbf{1} \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in}}]\right) \leq \underline{\mathbf{s}}[\ell_{\text{out}}, f, *]^\intercal - \left(\mathbf{1} - \mathbf{1} \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out}}]\right) \tag{4.31}$$

$$\mathbf{m}_{\text{shift}}[f]^\intercal * \underline{\mathbf{s}}[\ell_{\text{in}}, f, *]^\intercal + \left(\mathbf{1} - \mathbf{1} \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in}}]\right) \geq \underline{\mathbf{s}}[\ell_{\text{out}}, f, *]^\intercal + \left(\mathbf{1} - \mathbf{1} \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out}}]\right) \tag{4.32}$$

FIGURE 4.16
The offset of the reserved window is adjusted by $\mathbf{d}_{\text{hop}}[f]$ from hop to hop on the route used by f. On unused links (here: ℓ_3), all components of $\underline{\mathbf{s}}[\ell_*, f, *]$ have value 0.



adjust the position of the reserved cells along the path according to the per-hop delay for flow f. In Equations (4.31) and (4.32), $(\ell_{\text{in}}, \ell_{\text{out}})$ are the pairs of subsequent links of the candidate path $\mathbf{P}_{\text{cand}}[f, \pi]$, and $\mathbf{1}$ is a column vector with $\mathbf{t}_{\text{cycle}}[f]$ ones. Again, there is no need for a disjunction, since the wrap-around of the offset is implicitly accounted for by the roll-operation.

To break down Equations (4.31) and (4.32), let us first ignore the routing aspect. If the route of a flow f is fixed, then Equations (4.31) and (4.32) simplify to

$$\forall f \in \mathcal{F} : \forall \text{ subsequent edges } \ell_{\text{in}}, \ell_{\text{out}} \text{ on the route of } f :$$
$$\underline{\mathbf{s}}[\ell_{\text{in}}, f, *] \cdot \mathbf{m}_{\text{shift}}[f] = \underline{\mathbf{s}}[\ell_{\text{out}}, f, *]$$

or, equivalently, $\mathbf{m}_{\text{shift}}[f]^\intercal * \underline{\mathbf{s}}[\ell_{\text{in}}, f, *]^\intercal = \underline{\mathbf{s}}[\ell_{\text{out}}, f, *]^\intercal$. This means the offset encoded in $\underline{\mathbf{s}}[e, f, *]$ is advanced along the links on the chosen path for flow f according to the respective per-hop delay as depicted in Fig. 4.16.

But since we solve the scheduling problem and allow path selection, Equations (4.31) and (4.32) contain the additional auxiliary terms $\left(\mathbf{1} - \mathbf{1} \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_*]\right)$ to satisfy the constraints if not both links, ℓ_{in} and ℓ_{out}, are part of the chosen candidate path.

(Dummy) Objective

As described for the ILP formulation for joint scheduling and routing in Sec. 4.2, we can use a dummy objective on an uninvolved dummy variable (maximize or minimize $\underline{q}$) if we are only interested in obtaining any valid traffic plan.

We replace this placeholder objective in Sec. 4.4.4, where we want the ILP solver to compute optimized traffic plans.

4.3.3 Comparison and Relation to Joint Scheduling and Routing

Compared to the ILP with joint scheduling and routing from Sec. 4.2, the ILP formulation presented in this section makes different trade-offs in the modeling of the traffic planning problem.

If we take a step back, we make the following observation: In the spatial domain—with a path being a sequence of links—the basic unit of operation (path) in the ILP formulation in this section is a higher-level abstraction compared to the basic unit of operation in the ILP formulation from Sec. 4.2, which is the individual link. Similarly, in the temporal domain, if we think of a reserved window as composed of a set of one or more adjacent reserved cells, the basic unit of operation (reserved window) in the ILP formulation from Sec. 4.2 is a higher-level abstraction compared to the basic unit of operation in the ILP formulation in this section, which is the cell.

As a consequence of using a basic unit of operation on a higher level of abstraction for the ILP formulation, we can move decisions from the phase of actually solving the traffic planning problem to the phase of modeling and creating the problem instance. For example, while it is the responsibility of the ILP solver to select links that form a path from source node to destination node in the ILP formulation for joint scheduling and routing, it is our responsibility to ensure that only valid paths are presented as candidate paths to the solver. Conversely, we could possibly construct a reservation matrix $\mathbf{m}_{\text{resv}}$ with "gaps" in the reserved window and present it to the solver—this is not possible with the ILP formulation from Sec. 4.2.

4.4 Traffic Planning for Complemental Flows

From Sec. 4.2 and Sec. 4.3, we now have two different ILP formulations available for computing feasible traffic plans. Since we have a rather trivial dummy objective in place in Sec. 4.2 and Sec. 4.3, it is reasonable to expect that the solver returns the first solution that satisfies all constraints. That is, the ILP solver will compute "some" solution that corresponds to a traffic plan that *satisfies* all of our traffic planning constraints—provided that there exists a solution to the traffic planning problem, and the solver does not run out of memory (or we run out of patience).

Now, we want the solver to search among the set of traffic plans that satisfy all constraints for (one or more) traffic plans that are the best with regard to some objective. In our case, the objective function is not defined just in terms of the time-triggered traffic flows. Instead, we build on top of the ILP formulations from the previous section to compute *optimal* traffic plans for *complemental flows*.

Compared to time-triggered traffic flows (see Sec. 3.3.1), complemental traffic flows equip applications with more freedom with regard to when they are allowed to send a packet. To this end, complemental flows consist of a deterministic real-time part and a complemental non-time-triggered part. For the deterministic real-time part, the application has to comply with the time schedule prescribed by the traffic plan when sending deterministic packets. In turn, the network delivers the deterministic real-time part deterministically with bounded latency. In contrast, the complemental non-time-triggered part is transported with relaxed or no latency guarantees and therefore is called the opportunistic part in the following. The relaxed guarantees "buy" the application the freedom to transmit opportunistic packets at will, that is, at any time when no deterministic packet is scheduled already. The basic idea of this scheme is that the deterministic part of the flow provides the strictly mandatory guarantees to *safely* operate the networked application, whereas the opportunistic part improves the networks application's performance beyond the mandatory minimum. In other words, the deterministic part ensures that nothing "bad" will ever happen, whereas the opportunistic part is "nice to have" and used to further optimize the performance.

Networked control systems are a prime example of a class of networked applications that would benefit from such complemental flows as, for example, outlined in [Lin+19]. In [Lin+19], the deterministic part is used to guarantee the stability of the control system by enforcing latency bounds for a base rate of periodically transmitted sensor values or actuator commands. The opportunistic part transports additional sensor values and actuator commands that allow the controller to improve the control system performance, for example, by staying closer to the set-point, whenever there is extra bandwidth available.

It is important to see that the value of the opportunistic part is not completely independent of the deterministic part. For instance, an opportunistically transmitted sensor value transmitted immediately after a deterministically transmitted sensor value might carry only little additional information since both values are almost the same. In contrast, an opportunistic value with a greater temporal distance to the previous deterministic value might provide really new(er) information. This example shows that complemental flows should be considered when it comes to planning schedules and routes if we strive for maximum performance. More generally, we assume that the deterministic traffic part and the opportunistic traffic part of complemental flows are coupled by some internal state of the sending application.

Since existing routing and scheduling approaches for real-time networks in general do not have a notion of complemental flows, they miss the opportunity of optimizing opportunistic parts in relation to deterministic parts during scheduling and routing, ultimately resulting in lower application performance. Therefore, we propose joint routing and scheduling algorithms for networks explicitly supporting complemental flows.

Next, we introduce the concept of complemental flows consisting of time-triggered, periodic transmissions with deterministic real-time requirements and interspersed opportunistic transmissions. We propose different generic models to model opportunistic transmissions which serve as bases for defining optimization objectives. We then formalize the optimal routing and scheduling problem for complemental flows using the ILPs from Sections 4.2 and 4.3 as stepping stones.

4.4.1 Complemental Flows and Application Model

In this section, we describe the concept of complemental flows in the context of real-time networked applications.

Deterministic and Opportunistic Messages

We consider real-time applications that, on the one hand, require the time-triggered, cyclic transmission of packets that need to be delivered within deterministic time bounds and limited jitter. We refer to these packets in the following as deterministic packets. The deterministic communication pattern can be found in many real-time applications where timely delivery is a safety requirement. Consider, for example, the diverse applications found in modern cars where controllers, networked sensors, and actuators (servo motors) form several control loops. Here, sensor values need to be delivered from the sensors to the controller within time bounds. The same is required for actuator commands from the controller which have to be delivered to the actuators in time, else the car might be damaged or passengers' lives endangered. Depending on the specifications of the car (engine power, speed, accuracy of sensors, etc.), some minimum update (packet) rate and cycle time can be defined a priori guaranteeing that the car will be operational with (at least) a minimum required quality (safety condition).

On the other hand, additional updates might increase the accuracy and efficiency (quality) of the operation beyond the safe minimum. Here, quality is a performance metric that should be optimized, and quality is related to transmitted packets.

The problem is that network bandwidth is a shared resource since different applications might use the same onboard-network at the same time. Reserving more bandwidth beyond the minimum required bandwidth for one application makes it unavailable for other applications. Moreover, not all applications might require additional bandwidth at all times. Therefore, reserving excessive bandwidth in a time-triggered fashion at all times seems to be unjustified. Instead, we propose to use residual network resources opportunistically—when available—to optimize performance. We refer to packets transmitted outside the a priori defined time-triggered window for the deterministic traffic as opportunistic packets.

Our model distinguishing between deterministic and opportunistic packets has relevance beyond this example. For instance, it can be used to implement the well-known class of weakly-hard real-time (WHRT) systems. WHRT systems [Ber+01; Gaï+08] can tolerate a certain number of packets missing their deadline within a time window. We can ensure this using deterministic packets at a minimum rate, such that enough deterministic packets are part of each time window, and all other packets within the window are sent as opportunistic packets.

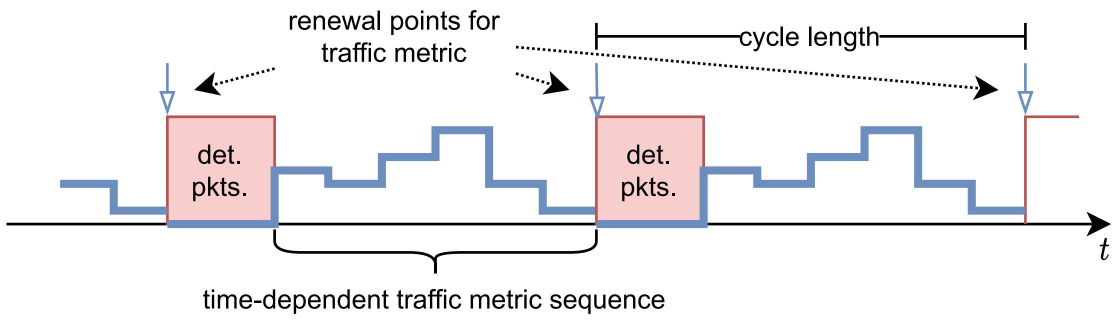

renewal points for traffic metric
cycle length
det. pkts.
det. pkts.
t
time-dependent traffic metric sequence

FIGURE 4.17 A time-dependent traffic metric allows to assign different traffic metric values to different time intervals. The reference point for the time-dependent traffic metric sequence is the transmission of the deterministic packet.

Optimizing Application Performance: Traffic Metric

Complemental flows consist of the aforementioned deterministic packets and opportunistic packets that guarantee safety, for instance, stability on the one hand, and optimize performance on the other hand. Considering the performance optimization through opportunistic packets, a traffic metric is required that relates opportunistic packets to their utility. The term traffic metric is therefore always tied to the opportunistic part of a complemental flow, and traffic metric refers to the traffic metric for opportunistic packets in the following. Based on this metric, we would like to schedule and route complemental flows such that performance is maximized—resulting in an optimization problem. Here, we strive for an offline scheduling and routing approach. Thus, we need to define traffic metrics that also can be estimated offline. These metrics are application-specific, therefore we cannot provide an exhaustive list of concrete metrics. Instead, we focus on the generic properties of metrics.

We distinguish between *time-independent* metrics and *time-dependent* metrics. For time-independent metrics, the utility of an opportunistic packet is not related to the schedule of the deterministic traffic. This means the temporal distance between the emission of an opportunistic packet and a deterministic packet does not influence the utility of the opportunistic packet. For instance, the average bandwidth (rate) of opportunistic packets is such a time-independent traffic metric following the reasonable assumption: the more packets, for example, containing sensor values, the higher the performance.

However, this simplifying assumption is neglecting the fact that not all packets might have equal utility, but the utility of a packet also depends on its value and time. For instance, a sensor value deviating significantly from the previous deterministically reported value might be more useful than sending the same value again. The problem is that for an offline approach, it is impossible to predict the concrete sensor values transmitted at runtime—which obviously depend on the situation (otherwise, one would not need to send the predicted messages at all).

Therefore, depending on the networked application, it is reasonable to make some assumptions. These assumptions draw from the particular characteristics of the networked application, the meaning of the data that is sent, and therefore require domain knowledge. For instance, in a network control application sensor readings taken immediately after each other might have almost the same value depending, for example, on the controlled plant's inertia. Here, opportunistic packets sent at a larger time distance to the previous or next deterministic packet might provide more useful information than packets sent immediately after or before deterministic packets. As another example, we could also think of applications where each deterministic packet contains strictly necessary information, but can possibly be followed by "a trailer" of a varying number of opportunistic packets which carry an enhanced set of information. In this case, the value of each subsequent opportunistic packet would be decreasing after each deterministic transmission with the gradient depending on, for example, the expected lengths of the "opportunistic trailers" and the relative occurrences of different "trailer lengths". That is to say, for different networked applications, the relation between the point in time of the transmission of a deterministic packet and the point of time of the transmission of an opportunistic packet can differ strongly.

This motivates the introduction of time-dependent traffic metrics that incorporate the relation of opportunistic packets to the time schedule of deterministic packets. In other words, a time-dependent metric can be interpreted as an a priori given function over time defining for each point in time the traffic metric value of an opportunistic packet sent at that time. Since we assume that utility is defined in relation to the previous and/or next deterministic packet, this function only needs to be defined over the time period of one cycle of the deterministic traffic and then repeats for every cycle, see Fig. 4.17.

4.4.2 System Model Amendments and Specifics

For the implementation of complemental flows, we pick up on the multi-paradigm nature of converged networking. Converged networks support multiple communication paradigms ranging from time-triggered real-time communication to simple best-effort communication with class-based priority scheduling.

Network and Bridges

For complemental flows, we explicitly require bridges to provide both, strict-priority packet scheduling and time-aware shaping.

By assigning different priority values to opportunistic packets and deterministic packets, each bridge uses FIFO queuing for the aggregate of deterministic traffic as well as for the aggregate of opportunistic traffic, see Sec. 2.3. Hence, the packet schedulers in the bridges need not distinguish between the individual complemental flows.

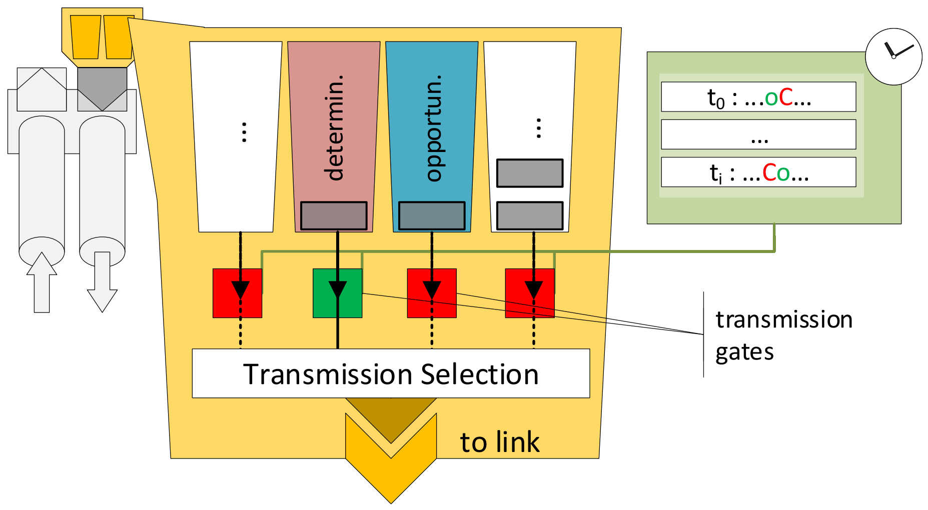


FIGURE 4.18 Output port with strict-priority scheduling and time-aware shaping. Deterministic and opportunistic packets are enqueued separately.

Complemental Flows

If a networked application issues a flow request for a complemental flow, we need additional information for the traffic planning. In particular, flow requests have to be extended with the traffic metric for opportunistic packets for the computation of the optimal traffic plan.

Both parts of a complemental flow are forwarded through the network along the same static route. Hosts, or rather the networked applications themselves, are aware of the distinction between deterministic and opportunistic packets and annotate packets in a suitable manner which allows the bridges to enqueue the packets in the corresponding output queue, see Fig. 4.18. For example, in IEEE Std 802.1Q, this can be achieved by setting different PCP values for deterministic and opportunistic packets.

It is easy to see that the “hard” requirements and constraints for the deterministic parts of the complemental flows correspond to reserved windows as previously introduced for time-triggered flows. With the equivalence deterministic packet $\leftrightarrow$ reserved window, we can map the scheduling problem for the deterministic packets to the computation of suitable offsets relative to the start of the transmission cycles, see Fig. 4.19. This means for each flow, we again use the start of a transmission cycle (see Eq. (3.2)) as reference point for the computation of the transmission schedule for deterministic packets. For each flow, the time intervals following the transmission of the deterministic packets can be used for the transmission of opportunistic packets.

We yet have to detail the traffic metric for the opportunistic packets. We expect that for a given traffic planning problem instance, the traffic metric is given in the same “unit” or measure for all complemental flows, or can be converted to a suitable proxy metric. For instance, the traffic metric could be equal to the average bandwidth for the opportunistic traffic part. Obviously, if we cannot relate the traffic metrics of different flows, for example, if one metric is given in terms of network load, the other metric in terms of (electrical) energy, we have a hard time computing an optimized traffic plan.

As an additional restriction, we expect traffic metrics that are additive in nature, that is to say, the addition of the traffic metric values of different complemental flows "makes sense". For example, a categorical traffic metric, which consists of labels ("unimportant traffic", "medium urgency", "priority", "express"), is not suitable as is (it would have to be mapped to an algebraic metric with concrete numerical values). Additive traffic metrics are also an established concept to describe traffic flows in the context of quality-of-service, for example, in admission control schemes [Jam+97].

Additionally, we require that the traffic metric value does not change significantly from hop to hop. In other words, the traffic metric should be chosen such that it can remain stable throughout the network, and is not changed, for example, by cross-traffic or the bridges themselves. To give a counter-example, a bridge with an aggressive traffic limiter that drops lots of packets could change the time-invariant traffic metric average bandwidth of a flow passing through this limiter and therefore violates this requirement.

This implies that the traffic metric is actually a property of the networked application or, to be more exact, a property of the process generating opportunistic packets. This has the practical advantage that the traffic metrics could be obtained for example by analyzing the application's sending behavior either analytically, or with a priori runtime measurements.

These requirements for the traffic metric also have a more pragmatic background. They facilitate solving the problem with ILPs, which require that we can model our problem only with linear inequalities and a linear objective function.

To model the time-independent traffic metric of complemental flows, we use a single *scalar value* that contains the aggregated information about the opportunistic transmissions for flow f.

For the time-dependent traffic metric, we again discretize the transmission cycle into a sequence of cells and assign the respective traffic metric value to each cell.

Note that depending on the actual implementation and the actual meaning of the traffic metric, the queuing of opportunistic packets has the potential to influence the temporal relation of the traffic metric. Here, we assume that this influence is negligible in relation to the overall uncertainty regarding the generation of opportunistic packets. We would like to remark that there are approaches in the literature to limit message queuing for opportunistic transmissions, for example, Controlled Load Services [Wro97] or bufferless statistical multiplexing [Rei+02].

In summary, a complemental flow models a stream of both, deterministic packets and opportunistic packets, that is generated from a shared, application-internal state of a networked application. For the time-independent traffic metric, the modeling of a complemental flow, for example, eliminates the possibility that the networked application simultaneously generates a deterministic packet and an opportunistic packet. Similarly, this shared state of the networked application manifests itself in the temporal linkage between the time-dependent metric and the phase of the deterministic packets.

It is a valid alternative to treat the deterministic packets and opportunistic packets of a complemental flow separately. We could, for example, replace a complemental flow with a pair of flows consisting of a deterministic flow and an opportunistic flow. However, this would add complexity to the traffic planning: we would have to consider twice as many routing decisions, and we would have to explicitly model the dependencies between deterministic and opportunistic packets, for example, matching the length of the routes and matching the offsets with additional constraints.

Therefore, we extend the ILPs from Sections 4.2 and 4.3 for traffic planning for complemental flows. We use a min-max objective function that minimizes the highest accumulated value of the traffic metric in the network. The ILP formulation from Sec. 4.2 lends itself for traffic planning in scenarios with time-independent traffic metric, whereas the ILP formulation from Sec. 4.3 is the starting point for traffic planning with time-dependent traffic metric.

4.4.3 ILP: Route Computation and Time-Independent Traffic Metric

The ILP formulation in this section is based on the ILP with route computation from Sec. 4.2. We have to incorporate the time-independent traffic metric and replace the dummy objective with the optimization objective for complemental flows.

Remember, the ILP formulation most notably uses the two variables $\underline{\mathbf{u}_{\text{link}}}[f, \ell]$ and $\underline{\mathbf{t}_{\text{offset}}}[f, \ell]$. From $\underline{\mathbf{u}_{\text{link}}}[f, \ell]$ (meaning: flow f uses link ℓ)) we can obtain the route a flow, and $\underline{\mathbf{t}_{\text{offset}}}[f, \ell]$ denotes the offset of the deterministic packet transmission of flow f on link ℓ relative to the start of the transmission cycle of f, see Fig. 4.19.

The time-independent traffic metric value of flow f is represented by an additional parameter $\mathbf{m}_{\text{trf,t.i.}}[f] \in \mathbb{R}_+$ with $f \in \mathcal{F}$.

Analogously, we introduce the set of variables $\underline{\mathbf{a}_{\text{t.i.}}}[\ell] \in \mathbb{R}_+$ to capture the accumulated value of the traffic metric on a particular link $\ell \in \mathcal{L}$. With the constraint

$$\forall \ell \in \mathcal{L} : \sum_{f \in \mathcal{F}} \mathbf{m}_{\text{trf,t.i.}}[f] \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell] = \underline{\mathbf{a}_{\text{t.i.}}}[\ell] \tag{4.33}$$

the variable $\underline{\mathbf{a}_{\text{t.i.}}}[\ell]$ is set to the sum of the traffic metric values of all flows which are routed via link ℓ.

For example, if the time-independent traffic metric corresponds to the expected average

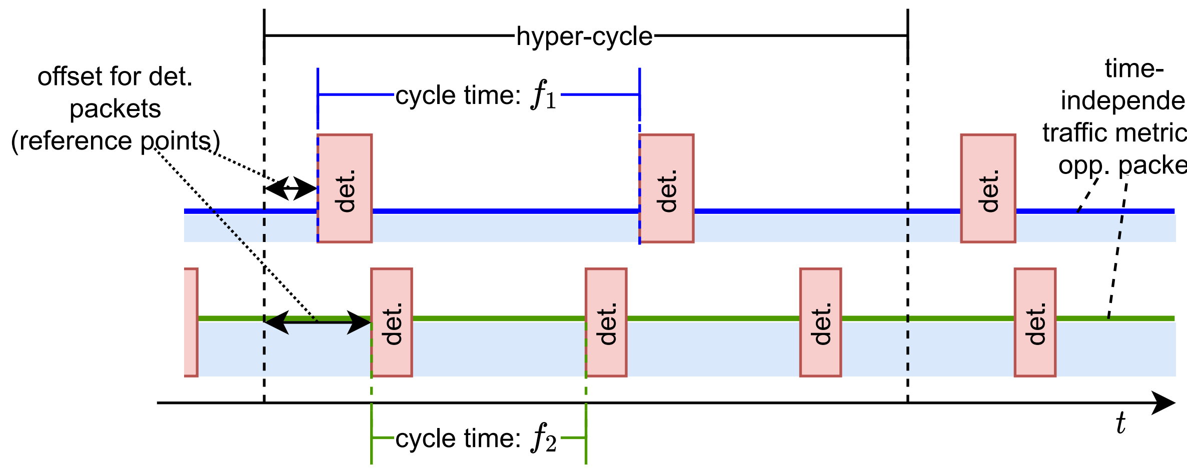

hyper-cycle
offset for det. packets (reference points)
cycle time: f_1
time-independent traffic metric for opp. packets
det.
cycle time: f_2
t

FIGURE 4.19 Example with time-independent (scalar) traffic metric: Computing feasible offsets for the periodic, deterministic packets of a complemental flow corresponds to the scheduling of reserved windows.

bandwidth consumed by opportunistic packets, we have to add $\mathbf{m}_{\text{trf,t.i.}}[f]$ to the expected average bandwidth consumed on every link that is part of the route of f, and $\underline{\mathbf{a}_{\text{t.i.}}}[\ell]$ captures the total bandwidth consumed by opportunistic packets on a specific link ℓ in the network.

To formulate the optimization objective, we additionally use a scalar variable $\underline{a_{\text{obj,t.i.}}} \in \mathbb{R}_+$. To avoid an extremely uneven distribution of the opportunistic traffic metric, we can replace the dummy objective from Sec. 4.2 with a min-max objective. To this end,

$$\forall \ell \in \mathcal{L} : \underline{\mathbf{a}_{\text{t.i.}}}[\ell] \leq \underline{a_{\text{obj,t.i.}}} \tag{4.34}$$

in combination with the objective

$$\text{minimize } \underline{a_{\text{obj,t.i.}}} \tag{4.35}$$

ties the auxiliary variable $\underline{a_{\text{obj,t.i.}}}$ to the globally highest value of the accumulated traffic metric on any individual link which then shall be minimized. This combination of constraints for the traffic metric reduces the highest value of the accumulated values of the traffic metric in the whole network. Due to flow conservation—packets are only created or consumed by the source and destination nodes—, this can distribute the traffic more evenly as long as the constraints for the deterministic packets remain satisfied.

We want to remark that the ILP can easily be adapted using the exposed variables and parameters. For example, if our traffic metric relates to the average bandwidth, we could impose bandwidth restrictions for the opportunistic traffic parts by adding constraints in the form $\underline{\mathbf{a}_{\text{t.i.}}}[\ell] \leq \text{bandwidth limit}$.

4.4.4 ILP: Candidate Path Selection and Time-Dependent Traffic Metric

Next, we extend the ILP formulation from Sec. 4.3 to optimize traffic plans for complemental flows with a time-dependent metric for the opportunistic part. Remember, the ILP formulation uses $\underline{\boldsymbol{\pi}_{\text{path}}}[f, \pi]$ and $\underline{\mathbf{u}_{\text{link}}}[f, \ell]$ to indicate path selection and used links, respectively, and variable $\underline{\mathbf{s}}[\ell, f, c]$ is used in the constraints for the selection of a scheduling option (= offset).

Here, we re-apply the concept of cell matrices, previously introduced to indicate the reserved cells ($\mathbf{m}_{\text{resv}}$) to represent the time-dependent traffic metric. In the time-dependent traffic metric matrix $\mathbf{m}_{\text{trf,t.d.}}[f] \in {\mathbb{R}_+}^{\mathbf{t}_{\text{cycle}}[f] \times \mathbf{t}_{\text{cycle}}[f]}$ with $f \in \mathcal{F}$, each row $\mathbf{m}_{\text{trf,t.d.}}[f][c_{\text{offset}}, *]$ describes the corresponding traffic metric sequence for scheduling option c_{offset}.

In the context of complemental flows, $\mathbf{m}_{\text{resv}}[f]$ and $\mathbf{m}_{\text{trf,t.d.}}[f]$ have the same number of rows and columns for a given flow f, and each scheduling option (row) corresponds to a different offset. In other words, both matrices use the same time-discretization of the cycle with the same cell duration of τ. The c_{offset}-th row in both matrices $\mathbf{m}_{\text{resv}}[f][c_{\text{offset}}, *]$ and $\mathbf{m}_{\text{trf,t.d.}}[f][c_{\text{offset}}, *]$ shall refer to the same actual value of the offset. This can be implemented again by using the roll-operator to construct $\mathbf{m}_{\text{trf,t.d.}}[f]$ analog to the construction of the reservation matrix as described before in Eq. (4.21):

$$\mathbf{m}_{\text{trf,t.d.}}[f][c_{\text{offset}}, *] = \text{roll}\left(\mathbf{m}_{\text{trf,t.d.}}[f][0, *], c_{\text{offset}}\right) \tag{4.36}$$

with $c_{\text{offset}} \in \{0, 1, 2, \dots, (\mathbf{t}_{\text{cycle}}[f] - 1)\}$. To summarize, if flow f is scheduled with offset c_{offset}, then $\mathbf{m}_{\text{resv}}[f][c_{\text{offset}}, c_{\text{index}}] = 1$ if the corresponding cell c_{index} is reserved for deterministic packets, else $\mathbf{m}_{\text{resv}}[f][c_{\text{offset}}, c_{\text{index}}] = 0$. The value of the traffic metric in cell c_{index} is $\mathbf{m}_{\text{trf,t.d.}}[f][c_{\text{offset}}, c_{\text{index}}]$, see Fig. 4.20.

Similar to the variables introduced in Sec. 4.4.3, we introduce the variables $\underline{\mathbf{a}_{\text{t.d.}}}[\ell, c_{\text{hyper}}] \in \mathbb{R}_+$. Variable $\underline{\mathbf{a}_{\text{t.d.}}}[\ell, c_{\text{hyper}}]$ describes the accumulated values of the traffic metric for the link $\ell \in \mathcal{L}$ in a specific cell c_{hyper} in the hyper-cycle, that is, $c_{\text{hyper}} \in \{0, 1, 2, \dots t_{\text{hyper}} - 1\} \subset \mathbb{N}$.

The constraint

$$\forall \ell \in \mathcal{L} : \sum_{f_\ell \in \mathcal{L}_f} \left(\mathbf{M}_{\text{trf,t.d.}}[f]^\intercal \cdot \underline{\mathbf{s}}[\ell, f_\ell, *]^\intercal\right) = \underline{\mathbf{a}_{\text{t.d.}}}[\ell, *]^\intercal \tag{4.37}$$

computes the value of the accumulated value of the traffic metric for each cell c_{hyper} in the hyper-cycle on link ℓ by summing over the contributions of each flow f using a candidate path that includes ℓ. In Eq. (4.37) $\mathcal{L}_f = \{\ell \in \mathcal{L} | \ell = \text{n}_{\text{out}}(n_{\text{src}} \text{ of } f)\}$ as before in Eq. (4.26). Structurally, Eq. (4.37) is similar to Eq. (4.27), but we consider the traffic metric instead of

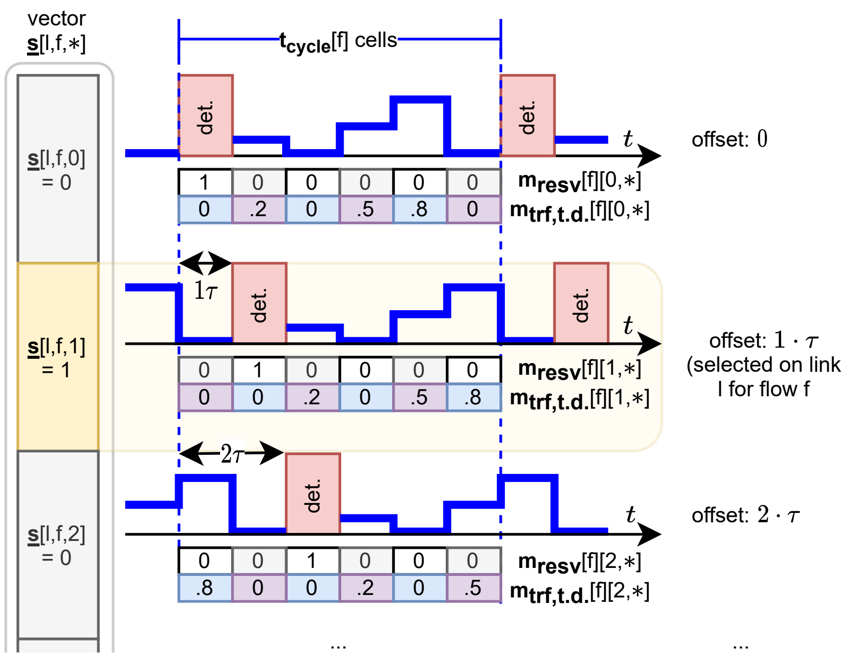


FIGURE 4.20 Rows of the matrices ($\mathbf{m}_{\text{resv}}[f]$, $\mathbf{m}_{\text{trf,t.d.}}[f]$) represent the complemental traffic parts for different offsets. Variable $\underline{\mathbf{s}}[\ell, f, c]$ indicates whether offset $c \cdot \tau$ is selected for the deterministic part of flow f on link ℓ.

reservations. Consequently, $\mathbf{M}_{\text{trf,t.d.}}[f]$ is constructed by

$$\mathbf{M}_{\text{trf,t.d.}}[f_\ell] = \underbrace{\begin{bmatrix}\mathbf{m}_{\text{trf,t.d.}}[f_\ell] & \cdots & \mathbf{m}_{\text{trf,t.d.}}[f_\ell]\end{bmatrix}}_{\frac{t_{\text{hyper}}}{\mathbf{t}_{\text{cycle}}[f_\ell]} \text{ times}} \tag{4.38}$$

with the same horizontal concatenation of the per-cycle traffic-metric matrices.

As was the case in Sec. 4.4.3, we introduce a scalar variable $\underline{a_{\text{obj,t.d.}}} \in \mathbb{R}_+$ and give the constraints and the objective function for the min-max-optimization. The main difference is that we want to minimize the maximum value $\underline{a_{\text{obj,t.d.}}}$ *in any cell* on any link via

$$\forall \ell \in \mathcal{L} : \forall c_{\text{hyper}} \in \{0, 1, 2, \ldots t_{\text{hyper}} - 1\} : \underline{\mathbf{a}_{\text{t.d.}}}[\ell, c_{\text{hyper}}] \leq \underline{a_{\text{obj,t.d.}}} \tag{4.39}$$

$$\text{minimize } \underline{a_{\text{obj,t.d.}}} \tag{4.40}$$

Remark: Variable $\underline{\mathbf{a}_{\text{t.d.}}}[\ell, c_{\text{hyper}}]$ exposes the accumulated values of the traffic metric, which facilitates the adaption of the objective function or the addition of extra constraints to account for traffic-metric related bounds, for instance, to limit the accumulated time averages on any link or to limit per-cell peak-values, etc.

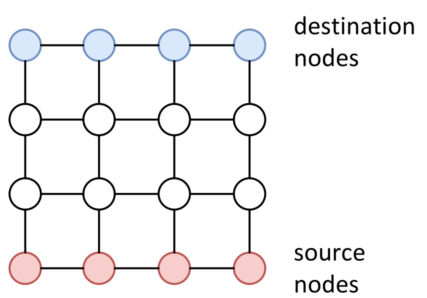


FIGURE 4.21
Topology and placement of source nodes and destination nodes used for the evaluations.

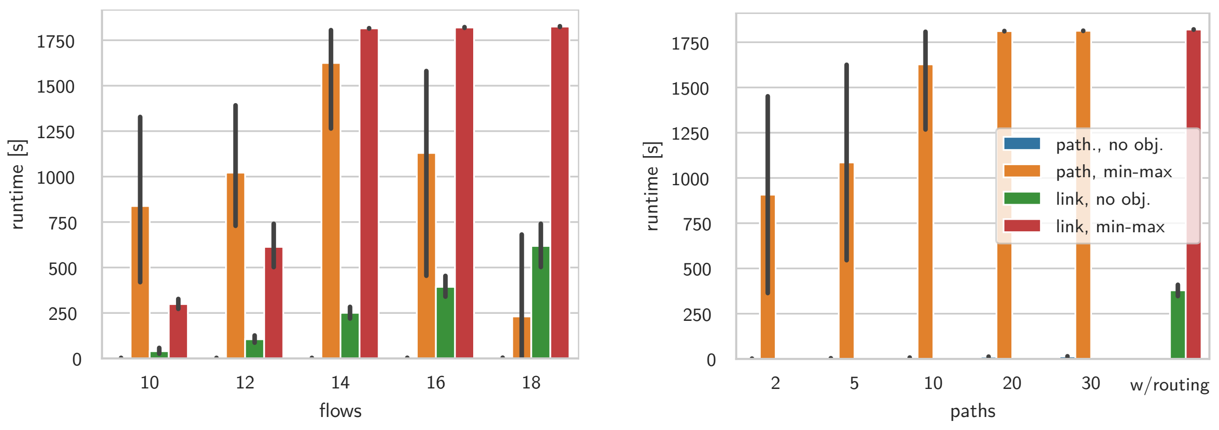
4.4.5 Evaluation

In Sec. 4.2 we evaluated one particular ILP formulation on a wide range of evaluation scenarios. In this section, we want to evaluate and compare different ILP formulations against each other. Therefore, we deliberately limit the variations regarding evaluations scenarios.

Evaluation Scenarios and Setup

Each evaluation scenario consists of a network (graph) and a set of flows. We use a 4-by-4 grid (see Fig. 4.21) as underlying network topology for all the measurements throughout this section. In this network topology, we can be sure that there exist multiple paths for each source-destination pair. For all flows, the source nodes are located on one side, and the destination nodes are located on the opposite side of the grid. To achieve an approximately similar number of flows per node on either side of the grid, we assign flows to the nodes on the source and destination side in a round-robin fashion, and each node may serve as source or destination for multiple flows, respectively. In other words, the source and destination nodes are placed deterministically in the network. The values of propagation delay and processing delay are 1 cell length for all links and nodes, respectively. For each flow, the number of cells per transmission cycle is randomly drawn from the set $\{10, 20, 40\}$, transmissions of deterministic messages require a reservation of 1 cell length, and the deadline (length of pre-computed paths, respectively) is such that up to two-thirds of all vertices may be traversed by each flow.

Due to the different granularity of routing and the traffic metric descriptions in the ILPs from Sec. 4.4.3 and Sec. 4.4.4, we have "pseudo-equivalent" twins for every flow in the evaluation scenarios. For each flow $f_{\text{t.i.}}$ in the set of flows for the route computation approach, there is a "pseudo-equivalent" twin $f_{\text{t.d.}}$ in the set of flows for the candidate-path selection approach which differs only with respect to routing and the traffic metric. The twins $f_{\text{t.i.}}$ and $f_{\text{t.d.}}$ have the same source node and destination node, but we pre-compute a set of candidate paths for $f_{\text{t.d.}}$ using the graph-tool library (version 2.27) [Pei14]. Similarly, for each flow $f_{\text{t.d.}}$ with time-dependent traffic metric, we created random sequences by assigning the absolute values of normally distributed



A Varying number of flows. **B** Varying number of precomputed paths.

FIGURE 4.22 Average solver runtimes with and without traffic-metric objective.

random numbers to the cells of $\mathbf{m}_{\text{trf,t.d.}}[f_{\text{t.i.}}][0,*]$. Then, the time-independent traffic metric value $\mathbf{m}_{\text{trf,t.i.}}[f_{\text{t.i.}}]$ of the corresponding twin $f_{\text{t.i.}}$ is set to the average of $\mathbf{m}_{\text{trf,t.d.}}[f_{\text{t.d.}}][0,*]$.

In addition to the ILP with route computation and time-independent traffic metric from Sec. 4.4.3 (abbr.: *link, min-max*) and the ILP with candidate-path selection and time-independent traffic metric from Sec. 4.4.4 (abbr.: *path, min-max*), we additionally use a version of both ILPs without objective. The ILP with route computation and time-independent traffic metric and no objective function (abbr.: *link, no-obj.*) lacks Constraint (4.34). Similarly, the ILP with candidate-path selection and time-dependent traffic and no objective (abbr.: *path., no-obj.*) metrics lacks Constraint (4.39). Note that the ILPs without objective effectively correspond to the ILPs from Sections 4.2 and 4.3.

We implemented the ILPs with Pyomo [Har+11; Har+17] and used Gurobi 8.1.0 [Gur19] to solve the ILPs for our evaluation scenarios in a containerized environment on a computing node (4× Intel Xeon E7-4850, 2.1 GHz, 1 TB RAM) running Linux 4.19.4.

In our evaluations, we limit the solver runtime to 30 min. For the evaluated scenarios, this did not affect the schedulability, only the optimality of the routes and schedules.

Evaluation Results

Figure 4.22A and Fig. 4.23 show the results for evaluation scenarios where the number of flows ranges from 10 to 18 flows with four candidate paths per flow for the ILPs with candidate-path selection.

In Fig. 4.22A, we plot the average runtime in seconds for evaluation scenarios with different numbers of flows. For 10 and 12 flows the ILPs with route computation and time-independent traffic metrics are solved faster on average, even though the sizes of the ILPs with route

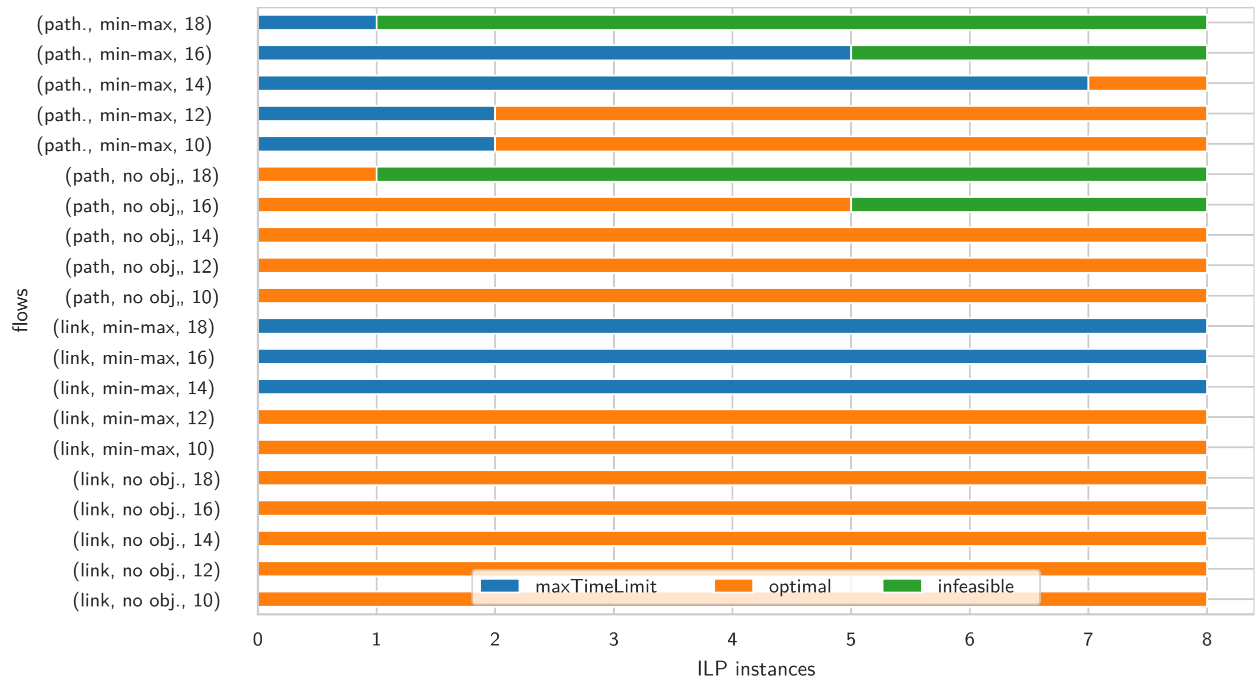


FIGURE 4.23 Exit status of solver, varying number of flows.

computation are in general a magnitude larger compared to the ILPs with candidate path selection in our evaluation scenarios (route computation: from $\sim 1.1 \cdot 10^5$ constraints and $\sim 1.3 \cdot 10^5$ variables for 10 flows up to $\sim 3.4 \cdot 10^5$ constraints and $\sim 3.5 \cdot 10^5$ variables for 18 flows; candidate-path selection: from $\sim 2 \cdot 10^4$ constraints and $\sim 8 \cdot 10^3$ variables for 10 flows up to $\sim 2.7 \cdot 10^4$ constraints and $\sim 1.1 \cdot 10^4$ variables for 18 flows). However, the solver hits the run-time limit of 30 min for two instances of evaluation scenarios with only 10 flows for the candidate-path selection min-max ILP, whereas all instances of evaluation scenarios with 10 and 12 flows for the route computation min-max ILP are solved optimally in less than 30 min.

In Fig. 4.22A, we also observe an at a first glance counter-intuitive decrease of the average runtime of the candidate-path selection min-max ILPs for 16 and 18 flows. The reason why the solver appears to require less runtime to solve the presumably harder problems with more flows is indicated in Fig. 4.23, which shows the solver status for the different instances of the evaluation scenarios.

Starting at 16 flows, there are evaluation scenarios that result in infeasible ILPs with candidate-path selection (irregardless of whether or not we have an objective function). Since infeasibility is detected relatively quickly by state-of-the-art solvers, these ILP “solutions”–which actually indicate the absence of a traffic plan that satisfies all constraints—skew the runtime observations.

This interpretation is supported by the runtime results in Fig. 4.22B. For the evaluation scenarios depicted in Fig. 4.22B, we fixed the number of flows to 16 flows and varied the number

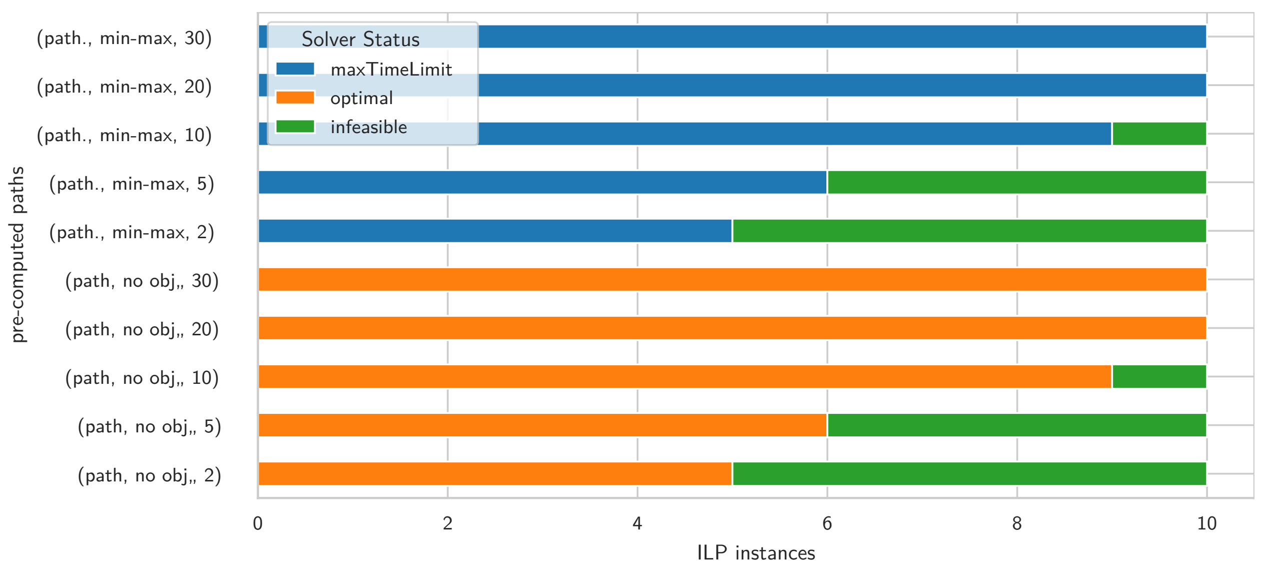


FIGURE 4.24 Exit status of solver, ILP with candidate-path selection for varying number of precomputed paths.

of precomputed paths. The corresponding solver statuses for these ILP with candidate-path selection are given in Fig. 4.24.

The required runtime for the candidate-path selection min-max ILPs for different numbers of pre-computed paths correlates with the number of infeasible evaluation scenarios since we observe a monotonically decreasing number of infeasible evaluation instances from five infeasible scenarios with only two precomputed paths to zero infeasible scenarios. That is, 10 out of 10 evaluation scenarios are schedulable for evaluation scenarios with 20 or more paths. Or, in other words, the schedulability of our evaluation scenarios increases (as expected) with the number of precomputed paths if we use the approach with candidate-path selection. As before (Fig. 4.22A and Fig. 4.23), all evaluation scenarios with the route computation variant of the flow parameters are schedulable with the ILPs with route computation. All ILPs with route computation and no objective terminate with a solution for these evaluation scenarios, and all route computation min-max ILPs yield a (possibly) non-optimal feasible solution within 30 min.

Considering only the ILPs without objective, then for our evaluation scenarios the candidate-path selection approach results not only in smaller ILPs but also much faster execution. In both, Fig. 4.22A and Fig. 4.22B, the ILPs with candidate-path selection without objective are solved in a few seconds (Fig. 4.22A: ≤ 4.1 s; Fig. 4.22B: ≤ 17.1 s for 30 paths) which is much faster compared to the average runtime of the ILPs with route computation and no objective (Fig. 4.22A: ≤ 99 s for 10 flows, ≤ 14.7 min for 18 flows; Fig. 4.22B: ≤ 12 min). In turn, the additional degrees of freedom in the route computation massively improve the schedulability.

For both (route computation and candidate-path selection) ILPs with no objective, we keep the

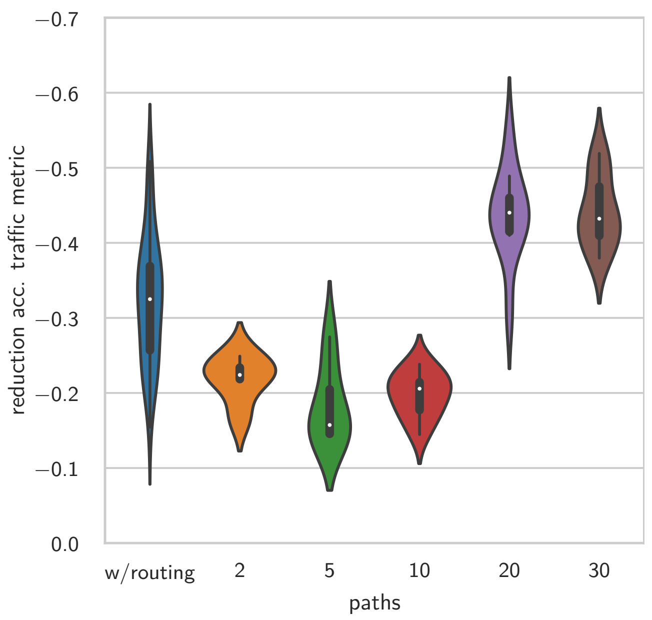


FIGURE 4.25
Average reduction of highest value of aggregated traffic metric for varying number of precomputed candidate paths.

constraints to compute the accumulated values of the traffic metric per link (Eq. (4.33)) or per cell per link (Eq. (4.37)). Despite our method for generating the traffic metrics, simply comparing the traffic metric values of the approach with route computation to those of the approach with candidate path selection is unfair, since they have different objectives (accumulated traffic metric value *per-link* vs. accumulated value *per-link per-cell*). Instead, we calculate the respective reduction of the objective value (highest value of the accumulated traffic metric values per link/per cell per link)

$$\text{reduction route computation, t.i.:} \left(\max_{\ell \in \mathcal{L}} \underline{\mathbf{a}_{\text{t.i.}}}[\ell] \right) / \underline{a_{\text{obj,t.i.}}} - 1 \tag{4.41}$$

$$\text{reduction candidate path-selection, t.d.:} \left(\max_{\substack{c \in \{0,1,2,\ldots t_{\text{hyper}}-1\} \\ \ell \in \mathcal{L}}} \underline{\mathbf{a}_{\text{t.d.}}}[\ell, c] \right) / \underline{a_{\text{obj,t.d.}}} - 1 \tag{4.42}$$

yielded by the min-max ILPs over the no-obj. ILPs.

In Fig. 4.25, the reduction is depicted for the evaluation scenarios with a varying number of paths from Fig. 4.22B and Fig. 4.24. In other words, Fig. 4.25 shows the improvement over the case where a traffic planning approach that ignores the opportunistic packets is used for traffic planning for complemental flows. The approach with route computation reduces the highest accumulated value of the traffic metric on any link on average by 32.4%, with the extrema ranging from a reduction of as little as 15.5% up to a reduction of 50.8%. The approach with candidate-path selection yielded an average reduction of the highest accumulated value of the traffic metric in any cell on any link in the range of 18.3% (for 5 paths) to 44.2% (30 paths).

The (feasible) evaluation scenarios from Fig. 4.22A and Fig. 4.23 yielded similar results with an average reduction of 23.9% for the approach with candidate-path selection and an average reduction of 30.8% for the approach with route computation.

Evaluation Summary

In the evaluations in this section, we performed a four-way comparison that addresses two different aspects.

On the one hand, we compared the effects of extending our traffic planning ILP formulations to ILP formulations for the computation of optimized traffic plans for complemental flows. We have seen in our evaluations that the integration of the opportunistic traffic metric and the traffic planning objective for complemental flows could generate noticeably improved traffic plans in terms of the traffic planning objective at the cost of longer runtimes of the ILP solver. For our evaluation scenarios, we also observed that there is often a noticeable improvement with regard to the traffic planning objective shortly after the solver has found the first feasible solution. Thus, even if we abort the traffic planning for complemental flows after the solver found a feasible (non-optimal) solution, these solutions usually compare favorably to pure constraint-based traffic planning, which ignores the opportunistic traffic part.

On the other hand, we also compared the ILP with route computation against the ILP path selection. Our evaluations showed that the ILP with path selection is an alternative to the ILP with route computation, which was oftentimes solved faster, and we could improve the schedulability and optimality of the results with an increasing number of candidate paths. Picking up on the earlier discussion from Sec. 4.3.3, the evaluations in this section underline that the modeling decisions play an important role for the ILP formulation with path selection. For example, if we use ILP formulation with path selection and consider too few candidate paths, we might prevent the solver from finding a feasible solution that the solver might have discovered if we had included more candidate paths or used the ILP formulation with route computation instead.

In other words, the ILP from Sec. 4.3 provides more explicit handles to influence the size of an ILP instance via modeling decisions. We can influence the temporal resolution (time-discretization) for both ILP formulations, but Sec. 4.3 explicitly requires us to specify the spatial granularity in the form of candidate paths when modeling the traffic planning problem. However, we have to keep in mind that a larger ILP instance does not automatically translate to "more work", and, in practice, may not automatically translate into higher runtimes, for example, if the ILP instance contains structures that can be efficiently exploited by pre-solving heuristics.

In general, we have to be aware and distinguish between the given parameters of the traffic planning instance, how we render this traffic planning instance to an ILP instance (which candidate paths to consider, time-discretization), and, finally, how we solve this ILP instance, if we want to compare traffic planning evaluations. Therefore, a direct comparison between the evaluations from Sec. 4.2.2 and this section may be misleading, since we used different network topologies, flow requests, and changed the ILP modeling framework and ILP solver.

4.5 Related Work

In general, the problem of "how to get packets from source to the destination in bounded time" reoccurs in different variations throughout the literature. In this thesis, we focus on approaches targeting multi-hop networks with directed links and deterministic, time-triggered packet-transmission schemes.

We organize the discussion of related work on traffic planning using different categories. In this section, we present the first part of our discussion of the related work on traffic planning with an emphasis on the different scopes regarding scheduling and route computation or path selection, and the various assumptions for different aspects of the system model. Since the next chapter continues with the overall theme of traffic planning, we defer the second part of the discussion of related work on traffic planning to Sec. 5.4.

A related approach may share some traits with our approaches in one category but differ in other categories. Therefore, we may reference the same work multiple times.

Scheduling Including Route Computation or Path Selection? We can differentiate traffic planning approaches depending on their "input" and the degrees of freedom during the computation of the traffic plan, especially with regard to routing.

On one end of the spectrum, routes are already fixed before the computation of the traffic plan, eliminating the spatial aspect. This reduces the task of traffic planning to a scheduling task. The scheduling problem, that is, the temporal component of traffic planning, is, for example, considered in [Ste10; Ste11; Dür+16; Cra+16b; Poz+16; Cra+17; Oli+18]. Unless the network provides but a single route between source node and destination node, including the routing decision in the computation of the traffic plan can improve schedulability, see Sec. 4.4.5.

There is some work on scheduling-aware routing [Nay+18b; Ata+19] where routes are computed a priori using heuristics that take the expected traffic load in form of the flow requests into account. The so-obtained routes are then used as input for a pure scheduling algorithm. Ideally, the effort spent on these scheduling-aware routing heuristics is more than compensated by a speed-up or improvement in solution quality in the following scheduling step.

The other end of the spectrum is occupied by joint scheduling and routing approaches where both, the transmission schedules and routes for the flows, are computed from scratch. Few approaches integrate the actual *route computation* on a per-link granularity and the scheduling in the same step as our approach from Sec. 4.2.

In one of the early approaches [Nay+16] (later extended in [Nay+18a] to support dynamic addition or removal of flows), the authors address scheduling of periodic transmissions in SDN networks. Their approach is limited to uniform transmission lengths, a network-wide base-period, and does not use the gating mechanism in the network. Instead, packets are scheduled by the source node for transmission in a time slot during which all links on the route of a packet are reserved simultaneously. In contrast, the ILP from Sec. 4.2 supports arbitrary lengths of reservations for transmissions, arbitrary periods of transmissions, and per-port schedules for the time-aware shapers in the bridges.

In [Smi+17], a 0-1 ILP formulation is presented for the joint message routing and scheduling problem which builds on an existing model of automotive communication networks. The authors do not consider the end-to-end delay of messages for the routing decision and employ a binary coding that directly relates the granularity of the schedules to the number of variables. The ILP formulation in [Smi+17] is targeted at a very specific use-case, since the authors aim to utilize their ILP as a component in a multi-objective optimization for automotive communication networks, which influences, for example, their choice of variables and encoding.

Another ILP formulation that addresses joint routing and scheduling is presented in [Sch+17]. The ILP in [Sch+17] shares many features with our approach from Sec. 4.2 such as support for per-flow transmission cycle-times but does not use zero-queuing. In a later work [Sch+20], the approach from [Sch+17] is enhanced to support joint route computation and scheduling for multicast flows, too.

Outside of constraint-programming approaches, there are heuristics which “interleave” the computation of routes and schedules. For example, [Pah+18; Pah+19b] describe a dedicated routing step that produces (as per the authors) *all* valid paths between source node and destination node of a flow which are then referred to during scheduling.

Interweaving scheduling constraints and routing constraints can quickly become very expensive in terms of computational resources. Similarly, “compiling” these routing constraints (if even possible) to a list of all valid routes may quickly exhaust the available memory. Therefore, some approaches try to cut down the degrees of freedom with respect to routing to achieve a compromise between the schedulability and computational effort.

With route computation on a per-link granularity as starting point, [Sch+17; Hel+21] propose various pre-processing techniques that reduce the search space for the route computation. For example, one idea is to ignore those links which will never be part of any path between source and destination node or to aggregate the constraints for line segments in the network.

Alternatively, instead of *cutting down* on routing decisions, some approaches start from the opposite side by exposing only a limited amount of routing decisions, to begin with. For example, pathsets routing is proposed in [Nay+16] for TSSDN to reduce the traffic planning runtime. Similar to the concept of candidate-path selection as used in Sec. 4.3, for each flow a number

(≥ 1) of valid routes between source node and destination node are pre-computed, and the decision of which candidate path is chosen is performed jointly with scheduling. A similar idea is described in [Ata+19] in the context of redundant paths where only a subset of sets of disjoint paths is taken into account during the computation of the global traffic plan.

All of our approaches, including the conflict-graph-based approaches from the next chapter, are designed to include the spatial aspect either in the form of route computation (constraints) or by offering routing decisions and thereby cover a wide range of the discussed spectrum for routing. Our approaches are also generic in the sense that we are not restricted to any specific network topology or network structure. We can easily adapt the approaches with path decision by swapping out the routing algorithm that produces the candidate paths. This also includes the conflict-graph-based approaches from Chap. 5 that reuse concept of candidate-path selection.

System Model From a bird's-eye-view, we compute a traffic plan for periodic, real-time transmissions between pairs of nodes in a packet-switched network where packets have to be fully received prior to being forwarded to the next node. This is commonly referred to as store-and-forward behavior. If we look closer, there are many, seemingly subtle differences in the system model that restrict the transferability of traffic planning approaches to another than the originally targeted scenario. Of particular relevance are queuing constraints and capabilities of the bridges: traffic plans where packets can be arbitrarily reordered in bridges may not work for networks with FIFO policies.

Nodes Focusing on the capabilities of a single node, it makes a difference, what—if anything at all—a node can do to control the point in time when a message is forwarded. For example, pre-TSN Ethernet and TSSDN [Nay+18a] bridges are work-conserving and will forward a frame as soon as it is at the head of the highest-priority non-empty FIFO queue at its outgoing port.

Nodes that support, for example, time-aware shaping as specified in [IEE18b] can block transmissions for some time on the level of traffic classes or priority levels. This is exploited by schemes where reserved windows are planned for multiple frames, possibly belonging to different flows [Oli+18; Hel+20a; Hel+20b]. Depending on the behavior of source nodes, these approaches may introduce some (bounded) jitter for individual frames caused by the shared gate event (opening, closing of the gate).

Closely related are models with cyclic queuing and forwarding [Kro+21]. Here, nodes also have a limited amount of outgoing queues which alternatingly become eligible. However, in these scenarios, the mapping frame-to-queue is usually performed on a per-node basis, which requires more management effort than a static priority-to-queue mapping.

Another commonly found assumption is that nodes have the ability to select any available frame from their internal buffers for transmission at a certain point in time. In this model, commonly found in the context of time-triggered traffic in TTEthernet [Ste+09; Ste+13], the arrival time and scheduled departure time of frames are effectively decoupled. “Newer” frames overtaking “older” frames is not bound to the number of outgoing priority queues but is limited by the available buffer space. This model is adopted for traffic planning, for example, in [Ste10; Cra+16a; Sch+17]. Traffic plans (or schedules) generated by these approaches therefore usually allow queuing but do not constrain the reordering of messages belonging to different flows in the bridges. Consequently, it is not guaranteed that the traffic plans are applicable to networks with TSN-style gating mechanism. The commonalities and differences between scheduling with gating mechanism and per-frame transmission scheduling are briefly pointed out in [Cra+17; Ste+18], too.

In contrast, zero-queuing is very attractive because it requires very little in terms of node capabilities. For example, in [Nay+15; Nay+16; Nay+18a], a method for deterministic traffic in plain SDN networks is presented where the source nodes are tasked with the time-triggered scheduling, that is, the schedule is the scheduling is performed at the network ingress.

But even for more “powerful” nodes, for example, with time-aware shaping, zero-queuing is attractive. There are multiple works on schedule synthesis for TSN networks with time-aware shaping which use the zero-queuing principle. In the context of TSN scheduling, an early approach is [Dür+16], which maps the temporal dimension of the traffic planning task to the no-wait job-shop problem. The heuristics from [Pah+18; Pah+19b] similarly do not allow buffering of frames once they have been sent by the source nodes. In [Ata+19], which adds the consideration of frame replication to the joint scheduling and routing problem, zero-queuing is used, too.

Similarly, all of our traffic planning approaches exploit the zero-queuing principle, too, making them comparably technology-agnostic.

Integration There are multiple works which—in contrast to us—consider task scheduling and traffic planning for networked real-time applications in conjunction [Cra+16a; Sye+19] or incorporate flow inter-dependencies originating from applications into the traffic planning [Pah+19b].

Another form of integration is the integration of time-triggered traffic and other traffic types which co-exist in the same network, often referred to as converged networking. A reoccurring theme are approaches to reduce the negative impact of the reservations for time-triggered traffic in terms of jitter and delay experienced by non-time-triggered traffic. For example, [Ste11] proposes to explicitly incorporate “gaps” into the transmission schedule for time-triggered messages. Integrating other, shaped traffic, for example, rate-constrained traffic or AVB streams into traffic planning for time-triggered flows is addressed in [Lau+16; Pop+16; Gav+18].

However, in the aforementioned approaches time-triggered and non-time-triggered traffic are treated as separate entities. In our approach for complemental flows in Sec. 4.4, we consider the inter-dependency of the two traffic parts with different criticality that compose a complemental flow. Other than from the traffic planning perspective, complemental flows themselves have been addressed in [Lin+19].

4.6 Challenges of ILP-Based Traffic Planning

In this chapter, we presented different ways to model and solve the traffic planning problem for time-triggered flows with integer linear programming. These ILP formulations allow to easily integrate various objective functions to compute not just any feasible (Sections 4.2 and 4.3) but an optimal traffic plan (Sec. 4.4), too. However, we also face some challenges with ILP-based approaches for traffic planning.

Firstly, we sometimes have to "work around" the limited expressiveness of integer linear programs, for example, to express traffic planning constraints that involve conditions (if link is used...) or alternatives (before or after reserved window...). Resolving these limitations, either by using notational shortcuts (if, or), or by using the "bigM"-technique directly, introduces additional auxiliary variables and constraints. We see this effect in the ILP formulation for joint scheduling and route computation, where we end up with several auxiliary variables (and constraints) in addition to $\underline{\mathbf{t}_{\text{offset}}}$ to express the scheduling constraints in the temporal domain. Similarly, in the ILP formulation for joint scheduling and path-selection, we have per-link variables $\underline{\mathbf{u}_{\text{link}}}$ (see Eq. (4.23)) in addition to $\underline{\boldsymbol{\pi}_{\text{path}}}$, even though the solver cannot make routing decisions on a per-link basis.

Secondly, the size of the constraint sets in these ILPs can also become a limiting factor, not only for the solver, but also during construction of the ILP instance itself. We have seen that the number of variables and constraints can increase strongly if the network size grows, or the cycle times get larger.

Thirdly, our ILP-based approaches suffer from the high degree of coupling between the route and schedule of an individual flow and the global (network-wide) traffic plan. This means that the requirements for a single flow are scattered over many individual constraints, for example, consider the scheduling constraints in Sections 4.2 and 4.3. This is not limited to our particular ILP formulations but applies too many similar approaches that use constraint-programming methods and operate on constraints formulated at the level of when to reserve a certain link for an individual flow. This high coupling makes it hard for the solver to find a valid traffic plan because a small variation of a variable for a single flow can result in a cascade of constraint violations that ripples through the whole constraint set.

In the next chapter, we meet these challenges by introducing conflict-graph-based approaches for traffic planning.

5 Traffic Planning with Conflict Graphs

In the previous chapter, we presented and evaluated ILP-based approaches for traffic planning, and we identified some challenges of ILP-based traffic planning in Sec. 4.6. Some of these challenges are due to the properties and limitations of the underlying method of integer linear programming. Other challenges result from formulating the traffic planning problem directly as a set of constraints on the routing variables and scheduling variables. In particular, the large constraint set and a high coupling of the traffic planning constraints make it hard to develop heuristics or to decompose the traffic planning problem at this level because we need to satisfy a very large set of constraints at once. Otherwise, we might end up with an invalid traffic plan where packets might not arrive at the destination node at all (if some routing constraints are not satisfied), or reserved windows are overlapping (if some scheduling constraints are not satisfied).

Therefore, in this chapter, we present conflict-graph-based approaches that use a different modeling of the traffic planning problem and also allow for replacing the ILP solver with heuristics. Conflict graphs have been employed in scheduling problems, for example, in the domain of rail traffic control [DAr+07], sensor networks [Hsu+09], and flow-shop scheduling [Tel+16]. For the purpose of traffic planning, we find conflict graphs also commonly applied to wireless scenarios [Dju+07; Li+10; Jia+10; Kon+18], where conflicts often model (spectral) transmission interference on links.

For our particular traffic planning problem, the basic idea is as follows: For each flow, possible schedule-route combinations, so-called flow configurations, and their relations to other flow configurations are represented as vertices and edges, respectively, in the (configuration-)conflict graph. We then compute a traffic plan by searching an independent vertex set in the conflict graph. An independent vertex set is a subset of vertices in the conflict graph where none of the vertices in the independent vertex set is connected by an edge to any other vertex in the independent vertex set. We will see that the conflict graph allows intuitive reasoning about the relation of individual (flow) configurations to the global traffic plan for all flows which makes it easier to develop heuristics. In simplified terms, we just have to pick the “right” flow configurations from the conflict graph to obtain a traffic plan. Additionally, conflict-graph-based traffic planning allows for iterative approaches with a progressively expanding search space.

We present the principles and fundamental concepts of our conflict-graph-based approach for traffic planning in Sec. 5.1. In Sec. 5.2, we then present a traffic planning algorithm built on these principles and evaluate a proof-of-concept implementation in comparison against ILP-based traffic planning. In our evaluations, our proof-of-concept implementation of the conflict-graph-based approach outperforms the ILP-based traffic planning and is more memory efficient making it a promising alternative to constraint-based traffic planning approaches. As in the previous chapter, we consider scenarios where all flows are known a priori, and schedule and route are not changed during the flow's lifetime in Sec. 5.2.

Then, we relax the restriction regarding a fixed set of a priori known flow requests. In Sec. 5.3, we consider scenarios where the set of currently active flows can change at runtime. This means we have to compute an updated traffic plan when flow requests arrive. We show, that conflict-graph-based traffic planning is suitable for dynamic scenarios and present our conflict-graph-based approach for dynamic, QoS-aware traffic planning in Sec. 5.3.

We continue the discussion on related work on traffic planning in Sec. 5.4, and sum up the features of conflict-graph-based traffic planning in Sec. 5.5.

5.1 Fundamental Concepts and Relations for Conflict-Graph-Based Traffic Planning

Remember that the input or parameters for the traffic planning consist of the network with its topology and specification (processing delay, transmission speed, propagation delay on a link) and the flow requests for a flow set $\mathcal{F}$. The degrees of freedom (variables) for the traffic planning are the transmission schedule (phases) and the routes of the flows.

In the following, we show how to translate the traffic planning problem into the problem of finding an independent vertex set problem in a conflict graph. Figure 5.1 visualizes the idea of the conflict-graph approach. We represent a "valid" configuration of a flow as vertex in the conflict graph, and edges between configurations in the conflict graph encode a violation of the traffic planning constraints. The traffic plan can then be obtained from an independent vertex set in the conflict graph. Every flow has to be represented by at least one configuration in the traffic plan if we want to have any chance of finding a traffic plan for all flows.

Next, we discuss and formalize these concepts in more detail.

Flow Configuration

Previously, the traffic plan—referring to schedules and routes that satisfy the traffic planning constraints for a set of flows—was laid out in full in the ILP solution. For each flow, we could directly obtain the information which links to use at which time from the variable assignment produced by the ILP solver. While this is convenient, it is also redundant: with zero-queuing, we can derive the complete transmission schedule for each flow from the value of phase ϕ at the source node provided that we know the path, see Sec. 3.4.

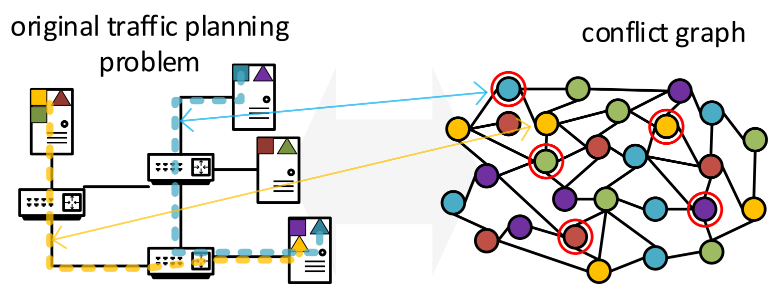


FIGURE 5.1 An independent vertex set in the conflict graph is a solution to the original traffic planning problem. Vertices in the conflict graph correspond to flow configurations.

In the spatial domain, Sec. 4.3 introduced the idea of candidate paths where we pre-compute and label possible routes for each flow. This allows us to capture the routing decision on a path-granularity with a path index variable π. Ultimately, due to the traffic planning constraints, for each flow, the phase value ϕ, and path index π fully determine when and where the packets will be transmitted. With this in mind, we arrive at a more concrete definition of a flow configuration:

DEFINITION 5.1 (Flow Configuration)
The tuple (f, ϕ, π) represents a flow configuration.

In other words, a flow configuration for flow f is the tuple of all scheduling variables and routing variables that fully define the source node behavior and the network behavior for packets of flow f such that all constraints on the routes and schedules of the traffic planning problem are satisfied in the *empty* network. Or, we could also say that a flow configuration for flow f is a valid solution for the traffic planning problem for a single flow. To solve the actual traffic planning problem for $\mathcal{F}$, we then have to find a combination of flow configurations which for each flow maintains "the illusion" that there is no other traffic.

Similar to Sec. 4.3, for each flow f we *once* pre-compute a set of candidate paths from source node to destination node subject to the routing constraints and the delay constraints. For each flow, paths that are too long such that packets traversing them do not reach the destination node within their flow's t_{e2e} must not be candidate paths. Remember, zero-queuing allows us to discard such paths by checking the hop-count, see Sec. 3.4. It follows that we only need a finite amount of path indices—in our case, path indices are integer numbers—to capture the "routing decision".

Since a continuous-time variable ϕ yields infinitely many flow configurations, we make some practical assumptions. As before, we use discrete time with appropriately mapped granularity, for example, with 1 µs resolution. This means that a flow configuration is nothing but a tuple of

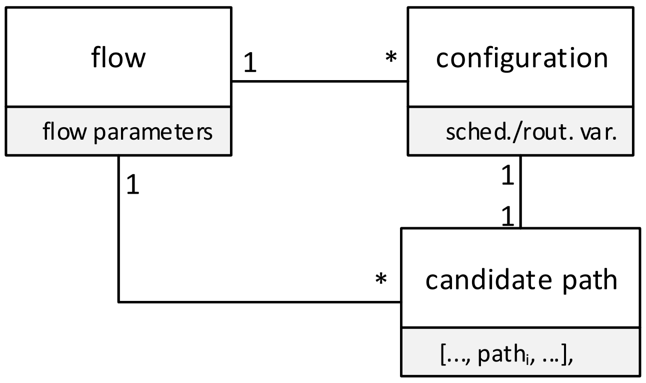


FIGURE 5.2
Relation between flows, configurations, and candidate paths.

integers from a finite set, which depends only on the properties of the network and the respective flow f itself.

Note that once we have computed the candidate paths, we can generate a new flow configuration simply by assigning valid values to f, ϕ, and π. We do not need to perform any costly routing or scheduling operation, nor do we need to know anything about other flows at this point. In general, for each flow, there exists a surjective mapping from configurations to flows, see Fig. 5.2.

Conflict

Conceptually, there exists a conflict between two configurations c_1 and c_2 if applying both configurations to the network and the source nodes of the flows of c_1 and c_2 violates the constraints for the routes and schedules of the traffic planning problem.

More intuitively, we have a conflict between two configurations if packets would have to be transmitted simultaneously on the same link to arrive at their respective destination without queuing. Since packet transmissions are serialized at each output port, we define configuration conflicts as a violation of the zero-queuing constraint.

DEFINITION 5.2 (Conflict)
Let c_1 and c_2 be two different configurations. The two configurations are in conflict if any packet sent with c_1 would be buffered due to a packet sent with c_2 or vice versa.

This is illustrated in Fig. 5.3. There is a conflict between configuration a_1 for flow f_a and configuration b_2 for flow f_b. The transmission at the source of f_b is scheduled too early, and packets sent with b_2 would have to be buffered until the transmission of packets from f_a is finished—which violates the zero-queuing principle. In contrast, configurations a_1 and b_1 are conflict-free due to the increased phase value for packets of f_b.

The zero-queuing constraint allows us to conflict-check pairs of configurations independently

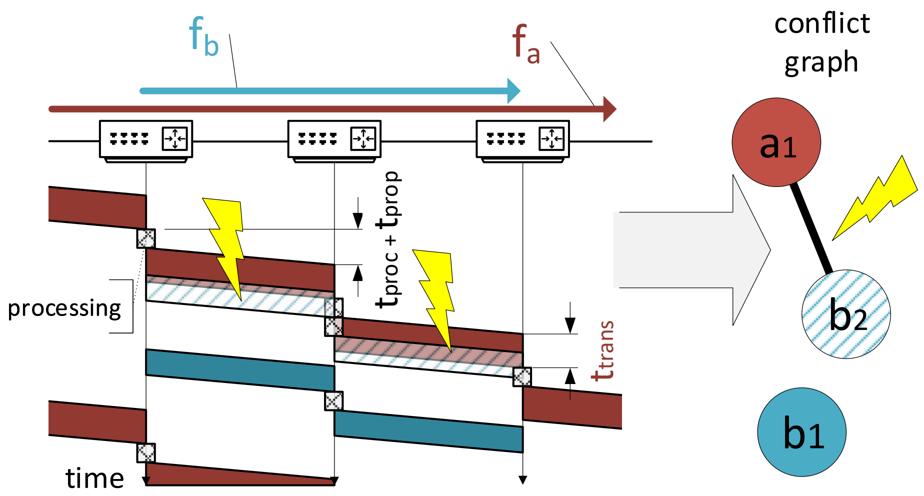


FIGURE 5.3 Configurations and conflict: Flows must traverse all links without buffering. There is a conflict between configuration a_1 for flow f_a and configuration b_2 for flow f_b, since the transmission at the source of f_b is scheduled too early, and packets sent with b_2 would have to be buffered until the transmission of packets from f_a is finished—which violates the zero-queuing principle. In contrast, configuration a_1 und b_1 are conflict-free due to the increased phase for packets of f_b.

in parallel. To compute whether two configurations conflict, we have to check if the scheduled packet transmissions are always temporally isolated on common edges, see Alg. 5.1.

Due to the cyclic property of the packet transmissions, we only check an interval of the length of the least-common multiple of the two flows, not the global hyper-cycle. However, packets may take longer than the hyper-cycle to reach the destination. Therefore, we have to use either modulo-arithmetics or array-rotations to account for transmissions that are crossing the hyper-cycle bounds.

One possibility to check for temporal isolation is illustrated in Fig. 5.4. This implementation closely resembles the idea of the reservation matrices from Sec. 4.3. Time is again represented

Algorithm 5.1: Checking for conflict between two configurations.

```
  input  : $c_1, c_2$
  output : true, if $c_1$ and $c_2$ conflict, else false
1 $\mathcal{L}_{\text{common}} \leftarrow$ common links on candidate paths of $c_1$ and $c_2$ ;
2 if $\mathcal{L}_{common} = \emptyset$ then return false;
3 else if $\forall \ell \in \mathcal{L}_{common}$ : reserved windows of $c_1$ and $c_2$ occupy mutually exclusive time intervals
    on $\ell$ then return false;
4 else return true;
```

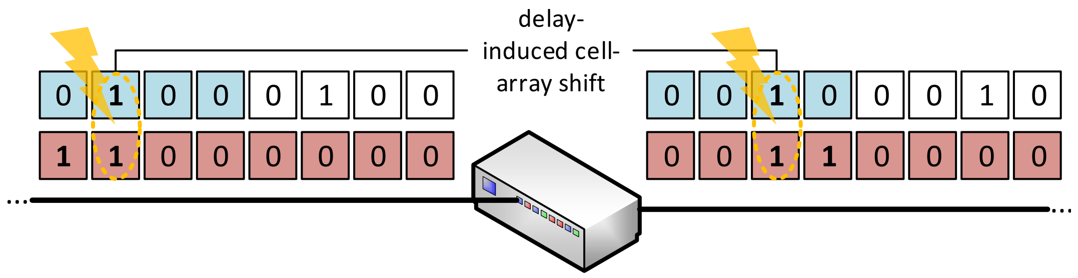


FIGURE 5.4 Discrete time intervals are modeled via arrays where each entry corresponds to a time interval in the transmission period. Delay is modeled via a circular shift of the array. If packets are scheduled such that the same time interval is used by more than one flow, the temporal isolation constraint is violated.

by arrays, and each array entry that corresponds to a time interval where the corresponding link is occupied by a transmission is marked (here: by value 1). If the ϕ-values of c_1 and c_2, respectively, are such that on any link that is part of both candidate paths the same interval is marked as reserved for both flows, c_1 and c_2 conflict.

Alternatively, we can represent the reserved windows by their start and end relative to the transmission cycle boundaries and use modulo arithmetic. Note that this requires proper handling of the wrap-around since on links other than the outgoing link of the source node the reserved interval may cross over the boundary of a transmission cycle. In this case, the reserved window is effectively represented by one part at the end of the transmission cycle and another part which “wrapped” back around to the start of the transmission cycle.

The effort to check whether two configurations conflict, scales—in principle—with the length of the candidate paths and the least common multiple of the cycle times of the two flows (not the global hyper-cycle). Since for each pair of flows f_a, f_b, we expect to encounter the same candidate-path pair π_a, π_b for many different phase-values ϕ_a, ϕ_b, we can use caching to store the common links of candidate-path pairs of two different flows. In practice, the effort depends on the shared sub-paths of the candidate paths and the relative primeness of the cycle times. Additionally, the computation of a conflict (see Alg. 5.1) allows for early exits. For example, if the candidate paths of two configurations are disjoint, we can be sure that there will be no conflict due to spatial isolation of the packets, and we need not even check the reserved windows.

Note that the existence (or non-existence) of a conflict is a commutative relation between two configurations. For its computation, we only require the network parameters and the configurations themselves. This is an important feature that allows the parallelization of a large portion of the traffic planning.

Conflict Graph

We encode the relations and constraints between the configurations in a conflict graph denoted by $\boldsymbol{G_c}$. A flow configuration is represented by a vertex in the conflict graph, and conflicts between configurations of *different* flows are represented by edges in the conflict graph. For example, in Fig. 5.3 there is an edge between configuration a_1 and configuration b_2.

The conflict graph includes for each flow a set of potential configurations, which we will call the flow's candidate configurations or candidates for short. We will use $\text{cand}(f)$ to denote the candidate set of flow f in $\boldsymbol{G_c}$, and we can interpret $\boldsymbol{G_c}$ as a colored graph. Given that—depending, for instance, on the granularity of the time-discretization or the number of candidate paths—for each flow, there might exist millions or even more of such configurations, we will later discuss strategies for a non-exhaustive conflict-graph construction such that the $\text{cand}(f)$ need not contain every single possible configuration of f.

DEFINITION 5.3 (Conflict Graph)
The conflict graph $\boldsymbol{G_c}$ is an undirected vertex-colored graph where all configuration vertices in $\text{cand}(f)$ are colored by f. Two configuration vertices in $\boldsymbol{G_c}$ are connected by an undirected edge if and only if they belong to different flows, and there is a conflict between the two corresponding configurations.

To insert a configuration c into $\boldsymbol{G_c}$, we add c to the vertex set of $\boldsymbol{G_c}$ and add an edge to every other configuration in of another flow in $\boldsymbol{G_c}$ which conflicts with c. To remove a single configuration c from $\boldsymbol{G_c}$, we delete all edges between c and the remaining configurations that exist in $\boldsymbol{G_c}$ and remove c from the vertex set of $\boldsymbol{G_c}$. Obviously, if c is the first configuration of a flow f to be added to $\boldsymbol{G_c}$, this introduces a new color f. Likewise, with the removal of the last configuration of a flow f, we remove the color f from $\boldsymbol{G_c}$. We detail a possible method to construct the conflict graph in Sec. 5.2.

Note that the conflict graph is not to be confused with the graph $\boldsymbol{G_n}$ representing the network topology. $\boldsymbol{G_c}$ is undirected, and an edge between two configurations indicates a mutual conflict between the two configurations. In other words, two configurations in $\boldsymbol{G_c}$ are connected with an edge if one or multiple of the constraints of the original traffic planning problem would be violated if we place the two flows with conflicting configurations into the network.

Relation between Traffic Planning and the Independent Colored Vertex Sets in the Conflict Graph

If we revisit the original traffic planning constraints from Sec. 3.5, we can roughly distinguish between intra-flow constraints and inter-flow constraints.

In this context, intra-flow constraints express, for example, that the route starts and ends at the appropriate nodes, or that the reserved window is scheduled appropriately regarding the reserved window on the preceding and/or the following link. These intra-flow constraints therefore only affect a single flow. Note that by design, a flow configuration already satisfies all flow-internal constraints.

The remaining inter-flow constraints, for example, temporal isolation and packet reorderings, can be violated due to interactions of packets of different flows. If we want to schedule more than one flow, we also have to make sure that we guarantee the remaining traffic planning constraints. But these inter-flow constraints are encoded as the priorly defined conflicts in the conflict graph!

DEFINITION 5.4 (Independent Vertex Set)
Denote by $\mathcal{V}$ the vertices and denote by $\mathcal{E}$ the edges in $\boldsymbol{G}_c$, respectively. $\mathcal{C} \subseteq \mathcal{V}$ is an independent subset of vertices in the conflict graph $\boldsymbol{G}_c$ iff $\forall u, v \in \mathcal{C} : (u, v) \notin \mathcal{E}$.

This has two implications. A), we can compute a traffic plan by searching for configurations that are not connected or, in graph-theoretic terms, configurations that form an independent vertex set in $\boldsymbol{G}_c$. B), a traffic plan is equivalent to a set of flow configurations, because a flow configuration defines when packets of that flow are traversing along which path through the network.

More formally, this is expressed in the following theorem.

THEOREM 5.5 (Feasible Traffic Plan)
Let $\boldsymbol{G}_c$ be a conflict graph for a flow set $\mathcal{F}$ in a network. Denote by $\mathcal{V}$ the candidate configurations for the flows in $\mathcal{F}$ in $\boldsymbol{G}_c$, and denote by $\mathcal{C} \subseteq \mathcal{V}$ a set of independent vertices in $\boldsymbol{G}_c$. Then $\mathcal{P} \subseteq \mathcal{C}$ is a traffic plan for the flows in $\mathcal{F}_{feasib}(\mathcal{P}) \subseteq \mathcal{F}$, iff $\forall f \in \mathcal{F}_{feasib} : \mathcal{P}$ contains exactly one configuration c which belongs to f.

PROOF (By construction.) Since $\mathcal{C}$ is an independent vertex set in $\boldsymbol{G}_c$, any subset of $\mathcal{C}$ is an independent vertex set in $\boldsymbol{G}_c$, too. In particular, any subset $\mathcal{P} \subseteq \mathcal{C}$ which contains at most one configuration for each flow from $\mathcal{F}$ is an independent vertex set in $\boldsymbol{G}_c$.

If $\mathcal{P} \neq \emptyset$, then $\mathcal{F}_{\text{feasib}}(\mathcal{P}) = \{f \in \mathcal{F} | \exists c \in \mathcal{P} : (c \text{ belongs to } f)\} \neq \emptyset$, that is, the set $\mathcal{F}_{\text{feasib}}(\mathcal{P})$ which contains all flows with at least one or more configurations in $\mathcal{P}$, is not empty, because each configuration belongs to exactly one flow. From the definition of the independent vertex set property and the definition of $\boldsymbol{G}_c$, it follows directly that all configuration $c \in \mathcal{P}$ are mutually conflict-free. Thus, by configuring every flow $f \in \mathcal{F}_{\text{feasib}}(\mathcal{P})$ according to the configuration $c \in \mathcal{P}$, it is guaranteed that none of the constraints on the routes, and schedules of $\mathcal{F}_{\text{feasib}}$ are violated.

If $\mathcal{P} = \emptyset$, all traffic planning constraints are trivially satisfied since $\mathcal{F}_{\text{feasib}}(\mathcal{P}) = \emptyset$. ■

Remark: We say a set of conflict-free configurations $\mathcal{C}$ *covers* a flow set $\mathcal{F}_{\text{feasib}}$ if $\forall f \in \mathcal{F}_{\text{feasib}} : \exists c \in \mathcal{C}$ with c belonging to f.

Thm. 5.5 is the basis for heuristics which provide partial solutions to the traffic planning problem. A partial solution to the traffic planning problem is a traffic plan for a subset of flows $\in \mathcal{F}$. According to Thm. 5.5, any (for practical purposes: non-empty) independent set $\mathcal{C}$ provides us with a valid traffic plan for at least those flows which it covers.

The special case where $\mathcal{C}$ covers all flows $\mathcal{F}$ is equivalent to the solution for the traffic planning problem from Sections 4.2 and 4.3 where we want to find a traffic plan for all flows.

COROLLARY 5.6 (Traffic Planning Solution)
Let $\boldsymbol{G}_c$ be a conflict graph for a flow set $\mathcal{F}$ in a network. Denote by $\mathcal{V}$ the candidate configurations for the flows in $\mathcal{F}$ in $\boldsymbol{G}_c$. Computing a set of independent vertices $\mathcal{C}$ in $\boldsymbol{G}_c$ such that for each flow in $\mathcal{F}$ there exists at least one configuration $c \in \mathcal{C}$ solves traffic planning problem for $\mathcal{F}_{feasib}$.

PROOF Follows directly from Thm. 5.5: For $\mathcal{F} = \emptyset$, $\mathcal{C} = \emptyset$. If $\mathcal{F} \neq \emptyset$, then $\forall f \in \mathcal{F} : \exists c \in \mathcal{C} :$ (c belongs to f), and all $c \in \mathcal{C}$ are mutually conflict-free because $\mathcal{C}$ is an independent vertex set in $\boldsymbol{G}_c$, that is, $\forall (c_1, c_2) \in \mathcal{C} \times \mathcal{C} : \nexists (c_1, c_2) \in \mathcal{E}$ with $\mathcal{E}$ the conflicts in $\boldsymbol{G}_c$.

Thus, we can trivially obtain a traffic plan $\mathcal{P}$ which guarantees that none of the traffic planning constraints are violated for $\mathcal{F}$ by selecting exactly one $c \in \mathcal{C}$ for every flow f ■

This means we can solve the traffic planning problem by searching for a set of independent vertices $\mathcal{C}$ which covers $\mathcal{F}$. If such a $\mathcal{C}$ contains multiple configurations for the same flow, we have to pick a single one that is included in the traffic plan $\mathcal{P}$, because we need *one* route and *one* phase (schedule) for each flow. If we have no particular traffic planning objective, we can make an arbitrary choice with respect to which configuration from $\mathcal{C}$ we select for a flow with multiple options in $\mathcal{C}$ because every choice will result in a feasible traffic plan for all flows.

The concept of conflict-graph-based traffic planning is generic in the sense that it can be applied to problems beyond our specific system model, provided that the following two conditions hold. Firstly, configurations of individual plannable entities need to be "additive" in the sense that it is possible to apply multiple configurations to individual nodes in the network without "destroying" the previously applied configurations, and secondly, it has to be possible to independently detect conflicts between any pair of configurations for the plannable entities. In our case, configurations are additive for both, routes—"addition" means adding a routing entry—and schedules. For the schedules, the additivity results from the temporal isolation constraint which allows us to merge the individual schedules on each bridge.

5.2 Conflict-Graph-Based Approach for Traffic Planning

In this section, we present an exemplary conflict-graph-based algorithm that exploits the advantages of the conflict-graph modeling to solve the original traffic planning problem. Figure 5.5 sketches the steps of this proof-of-concept conflict-graph-based traffic planning (CGTP) algorithm.

5.2.1 Overview

Our CGTP algorithm is iterative. In each iteration, we first grow the conflict graph, which means that we add additional configurations for each flow to the conflict graph. Our method to grow the conflict graph is discussed in Sec. 5.2.2.

After growing the conflict graph, we use a combination of different algorithms to try to find an independent vertex set $\mathcal{C}$ that covers all flows in $\mathcal{F}$ in the *current* conflict graph $\boldsymbol{G_c}$.

By default, we use a *fast* algorithm (here: a modified maximal independent vertex set algorithm) in every iteration. This default algorithm (see Sec. 5.2.3) is computationally cheap, but its speed comes at the cost of giving no guarantee regarding the number of flows covered.

Therefore, the execution of a second, slower algorithm (here: intermediate ILP) to try to compute $\mathcal{C}$ can be triggered. The second algorithm is computationally much more expensive, but it will find an independent vertex set in the current graph that covers the maximum amount of flows given enough time. Hence, we call it *max-cover* algorithm. However, intermediate executions of the max-cover algorithm have a runtime limit. We discuss this second algorithm and its trigger condition in Sec. 5.2.4.

Both, the fast algorithm and the max-cover algorithm can be followed up by the *completion heuristic*, see Sec. 5.2.5. When the completion heuristic is triggered, it tries to complete an almost complete independent vertex set $\mathcal{C}$ by generating new configurations only for those flows which have not been covered by the preceding algorithms.

In each iteration, we track the number of flows covered in the independent vertex set in a history $\mathbf{h}_{\text{found}}$. This history $\mathbf{h}_{\text{found}}$ is a list that is evaluated by the trigger rules to decide whether the max-cover algorithm or the completion iteration is to be executed in the current iteration, see Algorithms 5.4 and 5.5. While iterating, we also store the previously computed maximal independent vertex set which covers the most flows. This set is used to initialize the ILP and can be returned as partial solution if the CGTP algorithm is aborted.

We expect that, in practice, we will encounter scenarios where time-triggered traffic makes up only a fraction of the network load, for example, in converged networking scenarios. We assume that these scenarios will often be "easier to solve" compared to, for instance, scenarios where we saturate the network with the flows in $\mathcal{F}$, and $\mathcal{F}$ contains many flows with different cycle times and packet sizes. For an "easier to solve" traffic planning instance, we should be able to find a

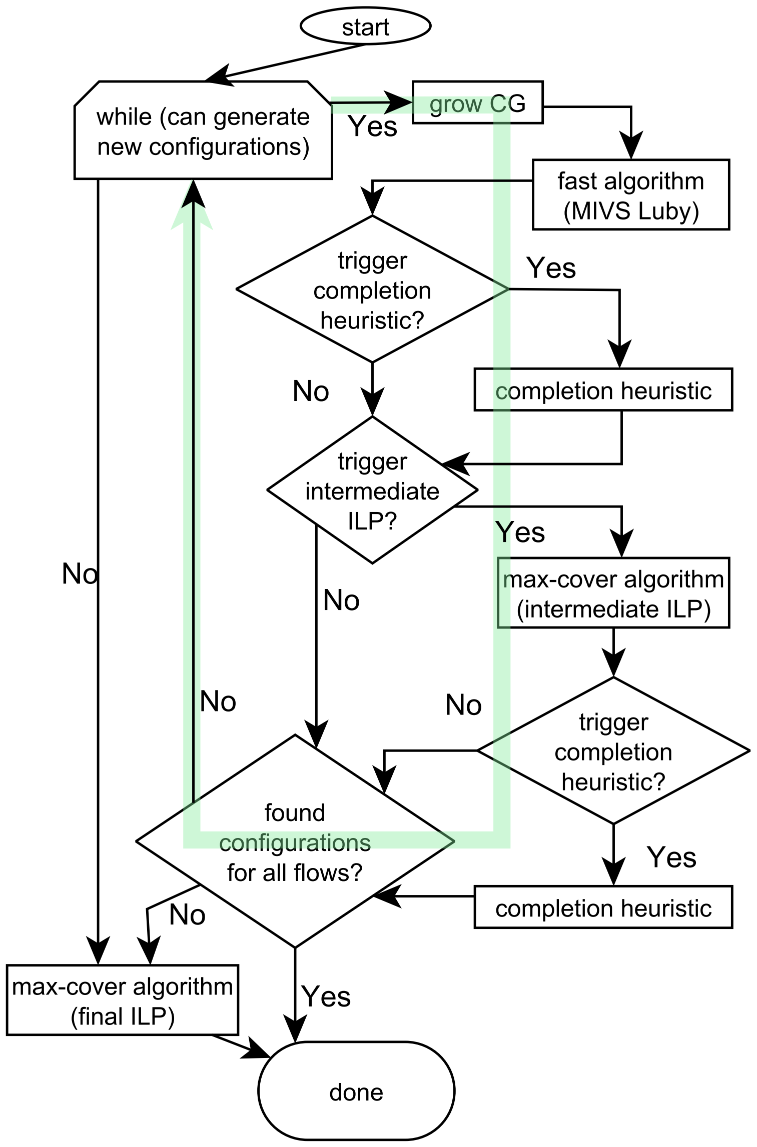

start
while (can generate new configurations)
Yes
grow CG
fast algorithm (MIVS Luby)
trigger completion heuristic?
Yes
No
completion heuristic
trigger intermediate ILP?
Yes
max-cover algorithm (intermediate ILP)
No
No
No
No
trigger completion heuristic?
found configurations for all flows?
Yes
completion heuristic
No
max-cover algorithm (final ILP)
Yes
done

FIGURE 5.5 Overview of the iterative, conflict-graph-based traffic planning (CGTP) algorithm. The green, opaque arrow indicates one iteration.

traffic plan after a few iterations and before we have added all configurations. In other words, "easier" traffic planning instances should result in earlier exits of the main loop (see Fig. 5.5) of the CGTP algorithm.

However, the underlying problem is known to be inherently NP-hard. Therefore, we can, in any case, end up with "difficult" instances that ultimately require something alike an exhaustive search. We consider a traffic planning instance for which we did not find a $\mathcal{C}$ that covers $\mathcal{F}$ during any of the iterations, even if the conflict graph finally contains all configurations, that is, encodes the whole configuration space, a "difficult" traffic planning instance. To handle these difficult instances, we execute the max-cover algorithm (here: intermediate ILP) one final time with a relaxed runtime limit compared to the intermediate executions of the max-cover algorithm.

Optionally, the final run of the max-cover algorithm can be executed without runtime limit. Without runtime limit, the max-cover algorithm is guaranteed to provide a solution covering the maximum amount of flows (except maybe for practical problems such as running out of memory). This means that we can modify the parameters of the algorithm to guarantee that the CGTP algorithm yields an exact solution, that is, a traffic plan for as many flows as possible for a particular problem instance.

After this overview, we present the details of the components of the CGTP algorithm. We start by breaking down how the conflict graph is built. Then, we explain the two algorithms to find independent vertex sets and the completion heuristic.

5.2.2 Building and Expanding the Conflict Graph

For the initial generation and the subsequent growth of the conflict graph in the CGTP algorithm, we define one stateful generator "function" per flow. Every time we invoke such a generator, it returns a new configuration for its flow—provided there still exists a new configuration for the respective flow (a configuration is "new" if it has not been returned previously by the generator). The generator state tracks which parts of the feasible configuration space of the associated flow it has already traversed, and it is updated whenever the generator function generates a new configuration.

Here, the state of our generator function implementation comprises most importantly the path index and the phase of the last generated configuration. On each invocation, a new configuration is returned with an incremented π until all candidate path indices for the current value of ϕ are covered. If all candidate paths for the current ϕ have been covered, ϕ is incremented and the candidate path index is reset to $\pi = 0$. This way, the generator functions sweep over the range of feasible values for their respective flow.

Remember that we have a discrete time representation, so every generator returns a finite amount of configurations. We call the discrete domain of these values the ϕ-π-space, see Fig. 5.6.

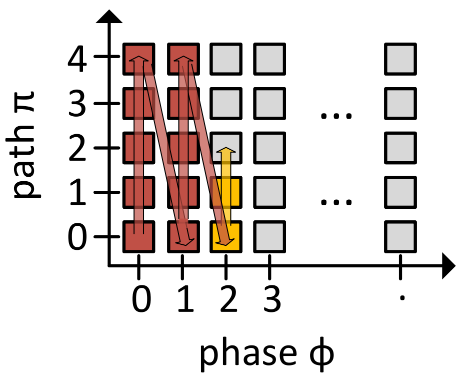


FIGURE 5.6
With each invocation, the configuration generator returns the next, new configuration by incrementing or resetting the values of ϕ and π, respectively.

Here, the configuration generators traverse the ϕ-π-space, "from left to right".

In the CGTP algorithm, we start with an initial conflict graph that contains a single configuration for each flow in the flow set. This means we invoke every per-flow generator function once and construct the initial conflict graph from those configurations by adding the corresponding conflicts between those configurations. In theory, there is a non-zero probability that there are no conflicts between these initial configurations, which means that is possible that we might construct an initial conflict graph that is already an independent vertex set.

Unless we have to schedule a few flows in a giant network such that the candidate paths do not even use common links, this is very unlikely, though. Therefore, we grow the conflict graph during the iterations of the CGTP algorithm. This exploits the property that it is possible to obtain a solution to the original traffic planning problem without a conflict graph including all configurations for all flows.

We grow the conflict graph at the beginning of each iteration by invoking the configuration generator functions for all flows in the flow set and inserting the so obtained configurations into the existing conflict graph. Note that much of the work to check whether a conflict exists between a configuration c_{insert}, which has to be added to the conflict graph, and a configuration c, which is already part of the conflict graph, can be executed in parallel, see Alg. 5.2.

5.2.3 Fast Algorithm: Adapted Maximal Independent Vertex Set Computation

According to Cor. 5.6, any independent vertex set $\mathcal{C}$ that covers all flows $\mathcal{F}$ solves the traffic planning problem. After some CGTP iterations, the conflict graph $\boldsymbol{G}_c$ contains many configurations per flow in $\mathcal{F}$, and thus the number of configurations $|\mathcal{V}|$ in the $\boldsymbol{G}_c$ is much larger than the number of flows in $\mathcal{F}$. With $|\mathcal{V}| \gg |\mathcal{F}|$, there is a chance for any independent vertex set $\mathcal{C}$ with $|\mathcal{C}| \geq |\mathcal{F}|$ to cover all flows in $\mathcal{F}$. In other words, any algorithm which finds a sufficiently large independent vertex set in the conflict graph may possibly solve the traffic planning problem.

Algorithm 5.2: Add a configuration c_{insert} to the conflict graph.

```
input  : $\boldsymbol{G_c}$, $c_{\text{insert}}$
output : $\boldsymbol{G_c}$
1 $\mathcal{V} \leftarrow$ configurations (vertices) in $\boldsymbol{G_c}$ ;
2 foreach $c \in \mathcal{V}$ do
3     $f_1 \leftarrow$ flow of $c_{\text{insert}}$ ;
4     $f_2 \leftarrow$ flow of $c$ ;
5     if $f_1 \neq f_2$ and conflict($c_{insert}, c$) then
6         add edge $(c_{\text{insert}}, c)$ to $\boldsymbol{G_c}$ ;
7     end
8 end
9 add configuration $c$ to vertex set of $\boldsymbol{G_c}$ ;
```

Therefore, we adapt Luby's algorithm for the computation of the maximal independent vertex set (MIVS) [Lub85]. Luby's algorithm is intuitive to understand and can be parallelized. Our adapted version is given in Alg. 5.3.

Luby's algorithm is an iterative algorithm, which terminates with high probability in $\mathcal{O}(\log |\mathcal{V}|)$ rounds with $\mathcal{V}$ the number of configurations in the conflict graph [Lub85]. Each iteration of the algorithm consists of two steps.

In the first step (see line 5ff., Alg. 5.3), the algorithm randomly selects some contending configurations. In our adapted algorithm (changes are highlighted in Alg. 5.3), the probability of a configuration c to become a contender depends on two factors, 1) the degree of the configuration, and 2) how many configurations that belong to the same flow as c are already part of the result set $\mathcal{C}$. For this, we track for each flow f the number of configurations from f which were previously included in $\mathcal{C}$ in the variable $\mathbf{fc}[f]$. The final probability accounts for both, the degree and the already included configurations, weighted by a factor a with $0 \leq a \leq 1$ and thus can raise the probability of configurations of flows not yet covered in the result set $\mathcal{C}$ to become contenders.

In step 2 (see line 11ff., Alg. 5.3), configurations from the set of contenders are selected to be included in the maximal independent vertex set. These steps repeat until a maximal independent vertex set is found. This means we stop when all configurations are either part of the result set $\mathcal{C}$ or neighbor configurations in $\mathcal{C}$.

Unfortunately, the adapted MIVS algorithm returns "only" a maximal independent vertex set. This means the fast algorithm returns any of the possibly multiple independent vertex sets in the conflict graph to which no additional configuration can be added. There is no guarantee that the maximal independent vertex set obtained by the fast algorithm is the maximum independent vertex set, that is, out of all maximal independent vertex set a set that contains the largest

Algorithm 5.3: Adapted Luby's algorithm for MIVS.

```
input  : $\boldsymbol{G}_c$, weighting factor $a$ (default $a$ =0.7)
output : independent vertex set $\mathcal{C}$
1  $\mathcal{C} \leftarrow \emptyset$ ;
2  $\boldsymbol{G}'_c(\mathcal{V}', \mathcal{E}') \leftarrow \text{copy}(\boldsymbol{G}_c(\mathcal{V}, \mathcal{E}))$ ;
3  while $\mathcal{V}' \neq \emptyset$ do
4      $\mathcal{X} \leftarrow \emptyset$;                                   // candidate set
5      foreach $c \in \mathcal{V}'$ do
6          $p_{\mathsf{deg}} \leftarrow \frac{1}{(2 \cdot \deg(c))}$ ;
7          $p_{\mathsf{sc}} \leftarrow 1 - \frac{\mathbf{fc}[f]}{\max(\mathbf{fc})}$ ;
8          add to $\mathcal{X}$ with probability $p = a \cdot p_{\mathsf{deg}} + (1-a) \cdot p_{\mathsf{sc}}$ ;
9      end
10     $\mathcal{C}' \leftarrow \mathcal{X}$ ;
11     foreach $(c_1, c_2) \in \mathcal{C}' \times \mathcal{C}' : (c_1, c_2) \in \mathcal{E}'$ do
12         if $\deg(c_1) \leq \deg(c_2)$ then
13             $\mathcal{C}' \leftarrow \mathcal{C}' - \{c_1\}$
14         else
15             $\mathcal{C}' \leftarrow \mathcal{C}' - \{c_2\}$
16         end
17         update($\mathbf{fc}$) ;
18     end
19     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$ ;
20     $\mathcal{Y} \leftarrow \mathcal{C}' \cup \text{neighborhood}(\mathcal{C}')$ ;
21     $\boldsymbol{G}'_c(\mathcal{V}', \mathcal{E}') \leftarrow$ subgraph on $\mathcal{V}' - \mathcal{Y}$ ;
22 end
```

amount of configurations. There is also no guarantee that the configurations in the maximal independent vertex set produced by the fast algorithm will ever cover all flows.

5.2.4 Max-Cover Algorithm: Finding Independent Vertex Sets Covering As Many Flows As Possible

Since the fast algorithm may not produce an independent set that covers all flows and thereby (fully) solves our problem, we employ another ILP as an alternative method. This ILP is guaranteed to find an independent vertex set that covers as many flows as possible, albeit it may take very long.

Therefore, this max-cover algorithm is only executed in two cases, a) if it appears that the fast algorithm does not make any progress over multiple iterations, that is, the number of covered flows is not increasing, or b) when the conflict graph contains all configurations for all flows and still no solution has been found.

Next, we explain the ILP formulation for finding an independent vertex set that covers as many flows as possible. Then we explain the trigger rule for the max-cover algorithm.

Integer-Linear Program for the Flow-aware Independent Vertex Set Problem The ILP to compute the independent vertex set that covers as many flows as possible is different from the ILP-based traffic planning in the related work and the previous chapter. It is much simpler and can be stated in the three lines:

$$\max \sum_{\underline{x_f} \in \underline{\mathcal{X}_\mathcal{F}}} \underline{x_f} \tag{5.1}$$

subject to

$$\forall(\underline{x_c^1}, \underline{x_c^2}) \in \text{edges}(\boldsymbol{G}_c) : \underline{x_c^1} + \underline{x_c^2} \leq 1 \tag{5.2}$$

$$\forall \underline{x_f} \in \underline{\mathcal{X}_\mathcal{F}} : \sum_{\underline{x_c} \in \underline{\mathcal{X}_\text{cand}[f]}} \underline{x_c} \geq \underline{x_f} \tag{5.3}$$

The ILP uses two sets of binary decision variables.

Every decision variable $\underline{x_c} \in \{0, 1\}$ in the decision variable set $\underline{\mathcal{X}_\mathcal{V}}$ represents one configuration in the conflict graph. The set edges($\boldsymbol{G}_c$) contains all edges in the conflict graph $\boldsymbol{G}_c$. If $\underline{x_c} = 1$, then the associated configuration c is part of the independent vertex set. Conversely, if $\underline{x_c} = 0$, then the associated configuration c is not part of the independent vertex set. Additionally, we define the helper sets $\underline{\mathcal{X}_\text{cand}[f]} \subseteq \underline{\mathcal{X}_\mathcal{V}}$ which includes all decision variables $\underline{x_c} \in \underline{\mathcal{X}_\mathcal{V}}$ which belong to the candidate configurations of a specific flow f.

Similarly, every decision variable $\underline{x_f} \in \{0, 1\}$ in the second decision variable set $\underline{\mathcal{X}_\mathcal{F}}$ represents a flow $f \in \mathcal{F}$. If $\underline{x_f} = 1$, the respective flow is covered by the independent vertex set. Consequently, if $\underline{x_f} = 0$ the associated flow f is not covered by the independent vertex set.

With the given objective function, the ILP solver searches for a variable assignment that returns an independent vertex set that covers as many flows as possible. If there exists such an independent vertex set that covers all flows, an ILP solver will find it—if it is provided with enough computing resources. Note that the NP-hard maximum independent vertex set problem [Gar+79] is reduced to this ILP if there is one vertex per flow.

Intermediate ILP Execution

Even though the ILP is "just three lines" long, solving this ILP may take an unknown, large amount of time which may be spent better by growing the graph and executing the fast algorithm. Therefore, we do not execute the max-cover algorithm in every iteration of the CGTP.

Instead, the max-cover algorithm is executed conditionally. For example, it may happen that the fast algorithm fails over the course of multiple iterations to return independent vertex sets which cover more flows even though the conflict graph has been growing. This can be caused, for example, if there is an "unfortunate" vertex degree distribution in the conflict graph such that the fast algorithm yields independent sets with many vertices of the same few flows. If the number of flows that are covered by the independent vertex set found with the fast algorithm stagnates or even decreases over time, then we want to execute the max-cover algorithm. This rationale is encoded in Alg. 5.4.

Algorithm 5.4: Trigger rule for max-cover algorithm.

```
input  : history $\mathbf{h}_{\mathsf{found}}$, window size $w_{\mathsf{ILP}}$,
output : trigger variable $r_{\mathsf{ILP}}$, window size $w_{\mathsf{ILP}}$
1  window ← slice of $w_{\mathsf{ILP}}$ last entries of $\mathbf{h}_{\mathsf{found}}$;
2  $d_{\text{past}} \leftarrow \sum \text{diff}(\mathbf{window})$ ;
3  if $d_{past} > 0$ then
4      $r_{\mathsf{ILP}} \leftarrow$ false ;
5      window size $w_{\mathsf{ILP}} \leftarrow w_{\mathsf{ILP}} + 1$ ;
6  else
7      $r_{\mathsf{ILP}} \leftarrow$ true ;
8      $w_{\mathsf{ILP}} \leftarrow$ minimal window size ;
9  end
10 return $r_{\mathsf{ILP}}$, $w_{\mathsf{ILP}}$;
```

Algorithm 5.4 evaluates how the amount of covered flows developed during a certain amount of past iterations. For this matter, we use a variable-sized window of the history which contains how many flows have been covered in the w_{ILP} last iterations. Alg. 5.4 returns a boolean r_{ILP} and an updated window size for the next iteration. The boolean r_{ILP} indicates whether to execute the max-cover algorithm in this iteration. The window grows when the fast algorithm "makes progress". The trigger rule evaluates if the number of covered flows has increased over the course of the iterations that are included in the window of the history. If the number of covered flows stagnates or decreases, we take this as an indication to execute the max-cover algorithm. To avoid that a single "bad" outlier, for example, caused by an unlucky random selection of contenders in the fast algorithm can cause the execution of the max-cover algorithm we increase the window size.

If the trigger condition is satisfied, we reset the window size to make it more sensitive to changes again and execute the max-cover algorithm. In our case, we invoke the max-cover algorithm—the ILP solver—with a runtime limit (default 5 min).

Additionally, there is a *limiter* to suspend the execution of the max-cover algorithm for a certain number of iterations if the max-cover algorithm has been triggered back-to-back for a certain number of past iterations. The limiter rule is motivated by the following two reasons that can cause the max-cover algorithm to fail to find a better independent vertex set—meaning covering more flows— in the current conflict graph:

In the first case, the conflict graph contains a solution, but it is very difficult to find, for example, if there is only one independent vertex set that covers all flows in the conflict graph. Then the max-cover algorithm may hit the time-out just because the problem is so difficult (but it could be solved given more time).

In the second case, the conflict graph does not even contain any independent vertex set which covers all flows at all (yet), for example, if we would have to schedule one or more flows with a phase for which we yet have not added configurations to the conflict graph (see Fig. 5.6: a required configuration may be "too far on the right side" of the ϕ-π-space). Then executing the max-cover algorithm is a waste of time, since we cannot find what does not exist—though the max-cover algorithm may still produce a partial solution, which may be of some value to the user.

Unfortunately, we do not know which of these two cases applies. Therefore, if we observe that even the max-cover algorithm struggles to make progress (else it would not have been triggered successively in the past iterations), spending the time which would otherwise be consumed by the execution of the max-cover algorithm instead on growing the graph (and spending only a moment to search for a solution with the fast algorithm) may resolve the problem in any case. The newly added configurations may make the problem easier to solve or even contain that particular configuration which we need to find an independent vertex set that covers all flows.

Therefore, the default setting (the limiter can be parameterized) is to suspend the max-cover algorithm for five iterations if it has been triggered twice in succession.

5.2.5 Completion Heuristic

The *completion heuristic* can be triggered after running either, the fast algorithm and the max-cover algorithm. It is triggered if the result produced by the preceding algorithm in this iteration leaves only a few flows uncovered. How close we have to be to cover all flows to execute the completion heuristic is governed by a threshold.

The idea behind the completion heuristic can be summarized as follows: Since we have come very close to the solution—an independent vertex set that covers all flows—using the configurations which are currently in the conflict graph, can't we just generate some more "new" extra configurations for the missing flows and see if they complete the solution?

Practically, the completion heuristic invokes the generator functions to get a limited set of new configurations for the missing flows. These extra configurations are added to the current conflict graph in the hope of finding among them some configurations which do not conflict with all configurations that are part of the current independent vertex set.

A fixed threshold regarding the number of missing flows may result in an excessive execution of the completion heuristic and is difficult to determine a priori. Therefore, we use a dynamic threshold. To adjust this threshold, we again evaluate how the number of covered flows developed in the past, see Alg. 5.5.

Algorithm 5.5: Update threshold for completion heuristic.

```
input  : history $\mathbf{h}_{\mathsf{found}}$, window size $w_{\mathsf{cplt.}}$, threshold $p_{\mathsf{thresh.}}$
output : window size $w_{\mathsf{cplt.}}$, threshold $p_{\mathsf{thresh.}}$
1  $\mathbf{window}^{\text{old}} \leftarrow$ slice of $w_{\mathsf{cplt.}}$ entries up to (including) the penultimate entry of $\mathbf{h}_{\mathsf{found}}$;
2  $\mathbf{window}^{\text{new}} \leftarrow$ slice of $w_{\mathsf{cplt.}}$ last entries of $\mathbf{h}_{\mathsf{found}}$;
3  $\bar{n}^{\mathsf{old}}_{\mathsf{found}} \leftarrow$ round(mean($\mathbf{window}^{\text{old}}$)) ;
4  $\bar{n}^{\mathsf{new}}_{\mathsf{found}} \leftarrow$ round(mean($\mathbf{window}^{\text{new}}$)) ;
5  if $\bar{n}^{\mathsf{new}}_{\mathsf{found}} < \bar{n}^{\mathsf{old}}_{\mathsf{found}}$ then
6      reduce $p_{\mathsf{thresh.}}$;
7      reduce $w_{\mathsf{cplt.}}$ linearly ;
8  else if $\bar{n}^{\mathsf{new}}_{\mathsf{found}} > \bar{n}^{\mathsf{old}}_{\mathsf{found}}$ then
9      increase $p_{\mathsf{thresh.}}$;
10     increase $w_{\mathsf{cplt.}}$ by $(w_{\text{max,cplt.}} - w_{\text{cplt.}})/2$ ;
11 else reduce $p_{\mathsf{thresh.}}$;
12 return $p_{\mathsf{thresh.}}$, $w_{\mathsf{cplt.}}$;
```

To this end, we use a sliding window approach that compares two windows of the history $\mathbf{h}_{\text{found}}$. The $\mathbf{window}^{\text{new}}$ contains the number of flows covered in the $w_{\text{cplt.}}$ most recent iterations. The $\mathbf{window}^{\text{old}}$ is shifted one iteration such that it contains the number of flows covered in the penultimate $w_{\text{cplt.}}$ iterations. For both windows, we compute the mean of covered flows and round it off to the next integer. If we observe a higher value for the newer window, the threshold $p_{\text{thresh.}}$ is raised which means that more flows have to be covered before the completion heuristic is triggered. In this case, the window-size $w_{\text{cplt.}}$ is increased, too, such that a single iteration where comparatively few flows have been covered has less impact. If the number of covered flows is decreasing, that is, the average of covered flows for the more recent iterations (after rounding) is less, we do the opposite. That is, the threshold is lowered to the effect that the completion heuristic is triggered with more missing flows, and the window size is reduced such that the threshold adjustment becomes more responsive. If the change is too small such that it is not registered after the rounding in line 3 and line 4 in Alg. 5.5, this indicates a lack of progress, and the threshold is lowered, too.

Considering the case that the completion heuristic is triggered but fails to cover the remaining flows, we can interpret the completion heuristic and its trigger rule as a secondary, optional growth phase of the conflict graph in an iteration. This second growth phase is restricted to flows which were left uncovered in the preceding conflict-checking step whereas the conflict-graph growth at the beginning of each iteration treats all flows indiscriminately. In this sense, the

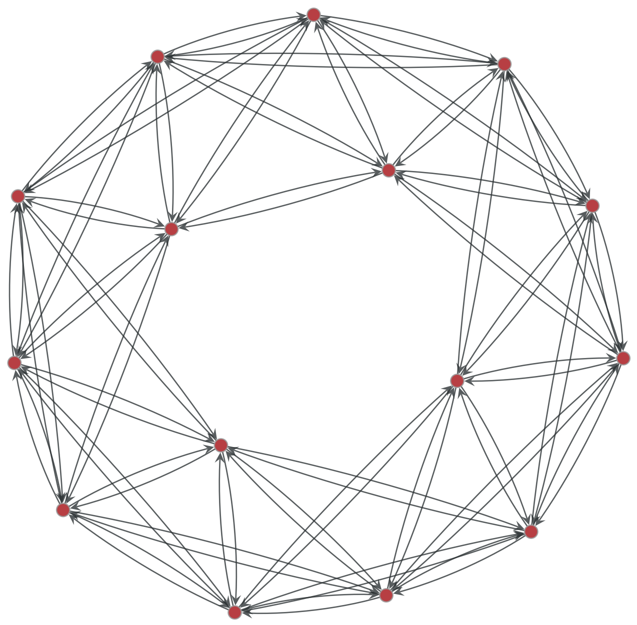


FIGURE 5.7
Example of the network topology: ring graph with $n = 15$ nodes and $m = 3$ neighboring nodes (in each direction) are connected.

completion heuristic also is a feedback mechanism inside the CGTP to “guide” the conflict-graph growth towards those flows which appear to be difficult due to being not covered by the independent vertex set produced by the fast algorithm and/or the max-cover algorithm.

5.2.6 Evaluation

Next, we quantitatively evaluate our conflict-graph-based traffic planning algorithm. Before we discuss the evaluation results, we explain the generation of evaluation scenarios and the evaluation method.

Evaluation Scenarios

We obtain problem scenarios by first generating a network topology and then creating a set of flows. The network topology is a ring graph where the nearest m-neighbors are connected (see Fig. 5.7). This topology is closely related to ring topologies, which are often found in industrial environments or in sensor arrays for direction-of-arrival tracking [Che+04]. This ring(n,m) topology combines several interesting features in comparison to the network topologies used in Sections 4.2.2 and 4.4.5. It is well characterized by its two properties, the number of nodes n, and the number of connected neighbors m in each direction, and we can be sure that there exist multiple different candidate paths for every pair of flows for $m > 1$. On the other hand, by varying the ratio of n to m, we can cover a spectrum between network topologies where many flows traverse are routed over a common set of links (large n, small m) and highly meshed

TABLE 5.1 Specification of compute nodes.

	PCsmall	PCbig
CPU	1 Intel Xeon E5-1650v3, 3.50 GHz	4 Intel Xeon E7-4850 v4, 2.1 GHz
RAM	16 GB	1 TB
host OS	CentOS Linux 7.7.1908, Kernel 3.10.0-1062,	Arch Linux, Kernel 5.2.5

	software
container	docker v19.03, Fedora 30-based container image
SW (CGTP)	Julia 1.2.0, LightGraphs, JuMP with Gurobi 8.1.0
SW (RILP)	Python 3.7, graph_tool, Pyomo with Gurobi 8.1.0

networks.

Unless specified differently, $m = 3$ neighboring nodes are connected. The processing delay is set to 2 µs (here: time is discretized in 1 µs-intervals). Propagation delay is neglected.

In each evaluation scenario, the flow parameters cycle time t_{cycle} and the size of the reserved window $t_{\text{resv}} = t_{\text{pkt}}$ are the same for all flows, and we have the same number of candidate paths per flow. By default, we consider 3 candidate paths per flow. Source node and destination node for each flow are randomly selected among the nodes of the network. This keeps the number of parameters low and reduces the probability of infeasible scenarios, for example, caused by relative prime t_{cycle}. We write the network properties and the flow parameters to plain-text files, which are ingested by the traffic planning programs.

Evaluation Setup

We implemented the conflict-graph-based traffic planning algorithm from Sec. 5.2 in Julia [Bez+17] and refer to this implementation of the conflict-graph-based traffic planning program as CGTP. CGTP uses the LightGraphs library [Bro+17] to compute for each flow the candidate paths via a k-shortest path routing algorithm and interfaces with the ILP solver Gurobi [Gur19] via JuMP [Dun+17].

Additionally, we also implemented an adapted version of the ILP with path selection from Sec. 4.3 to compare the conflict-graph-based traffic planning approach to a constraint-based approach. Here, we refer to this constrained-based approach as *reference ILP* (short, RILP, not to be confused with the ILPs used for the max-cover algorithm in CGTP). To run the RILP, we use a Python program and the modeling library Pyomo [Har+11] to build the RILP model for the traffic planning problem. Pyomo also interfaces with the ILP solver Gurobi, and we compute the k-shortest-paths for each flow using the graph-tool library [Pei14].

Tab. 5.1 summarizes the computing setup of our evaluations.

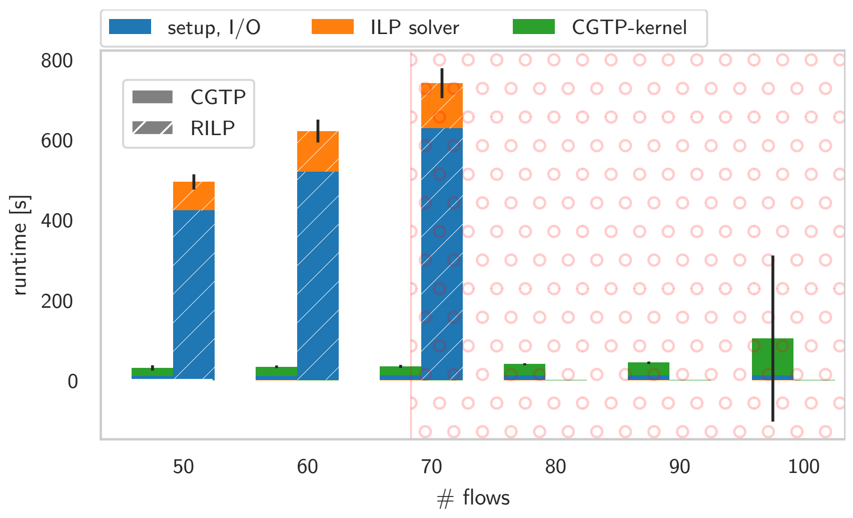


FIGURE 5.8 Comparison of runtime on compute nodes of type PC-small. The area marked by circles indicates a shortage of memory for the RILP.

Evaluation Results

Next, we present the evaluation results, starting with a performance comparison of CGTP and RILP before looking at different properties of CGTP. For all results in our evaluations, the CGTP-algorithm yielded optimal results in the sense that we obtained a traffic plan covering all flows.

Comparison CGTP vs. RILP For the comparison of CGTP and RILP, we use a network with 50 nodes. We increase the number of flows in steps of 10, starting at 50 flows, with $t_{\text{cycle}} = 300\,\mu\text{s}$ and $t_{\text{resv}} = t_{\text{pkt}} = 5\,\mu\text{s}$ for all flows. We solve 20 problems per step, and each problem is solved once with CGTP and once with RILP on compute nodes of type PCsmall, see Tab. 5.1.

For these scenarios, the *wall clock* total runtime, that is, the time from importing the problem from the files to writing the solution to disk, is divided into the time for setup and I/O, and the time for the actual solving process. In the case of CGTP, setup and I/O includes the time to read and write the files and to pre-compute the candidate paths for the flows. Time spent in the CGTP-kernel consists of the time for growing the conflict graph and searching for independent vertex sets. For RILP, setup and I/O includes, besides file operations and path pre-computations, the time required for the construction of the ILP model. The time spent on solving the ILP is measured around the function call, which interfaces with the solver.

The average runtimes are depicted in Fig. 5.8. The vertical line on top of each composite bar indicates the standard deviation of the total runtime.

While it is apparent that CGTP solves the problem in much shorter time, for example, the average total runtime of CGTP for 100 flows with 103.2 s is almost five times less compared to the average total runtime of RILP (493.4 s) for only 50 flows, we want to highlight another

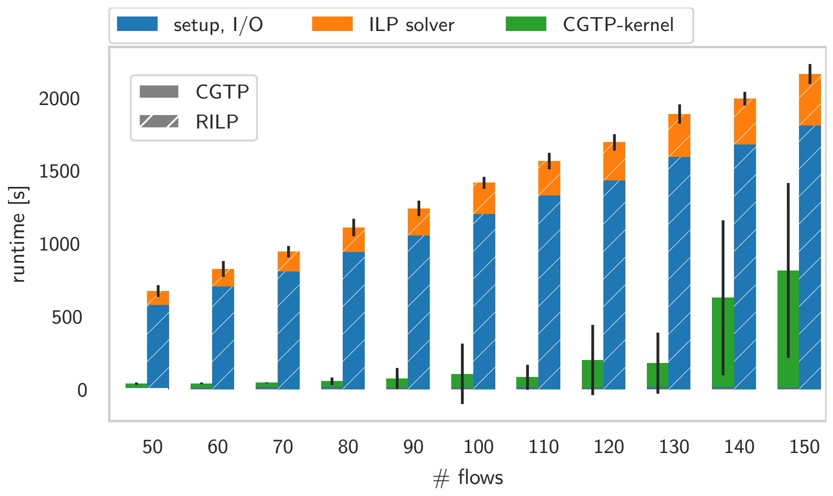


FIGURE 5.9
Comparison of runtime on compute node PCbig.

advantage of CGTP over constraint-based implementations. On the PCsmall compute nodes, the RILP software stack quickly hit the "memory bound". For already 70 flows, RILP could not solve 5 of 20 scenarios due to a shortage of memory, and for 80 or more flows no scenario was solved as indicated by the missing bars in Fig. 5.8. Constraint-based approaches which solve the traffic planning problem in the network domain inherently result in a large set of constraints (see Sections 4.2 and 4.3), and solvers often occupy a large chunk of memory for tracking the already covered solution space.

In contrast, CGTP runs to completion for the same evaluation scenarios on the PCsmall compute nodes. CGTP is more memory efficient during the solving process because the configurations in the conflict graph have a high information density compared to the individual variables in the RILP. To solve the flow-aware independent vertex problem, we only need the configurations and the edges in $\boldsymbol{G}_c$ which can be encoded using "a few" integers and adjacency lists. All the details in the network domain are only necessary when constructing or growing the graph or when finally assembling the solution. Thus, the conflict-graph-based approach has the very practical advantage of pushing out the borders of tractable problems in scenarios with limited memory. Admittedly, out-of-core implementations can extend the memory bound for both approaches.

We re-ran the evaluations on a compute node of type PCbig (see Tab. 5.1) with much more RAM. The results (averaged over 20 scenarios per step) are depicted in Fig. 5.9.

Up to 130 flows, the whole runtime of CGTP is on average faster than the time spent for just calling the ILP solver (excluding setup and I/O), even though the ILP solver is able to make use of all processor cores, whereas CGTP is implemented mostly single-threaded. While CGTP still outperforms RILP by a significant margin for all evaluation scenarios, we observe a steep increase in runtime and the variance of the total runtime for CGTP from 130 flows on. The

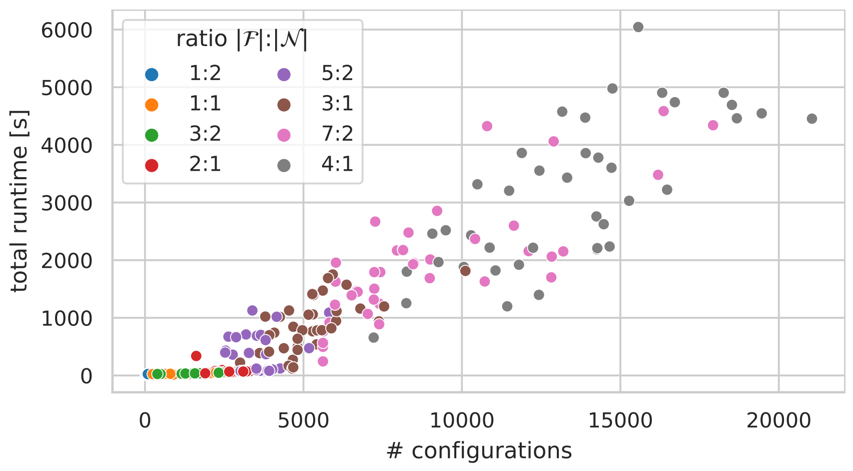


FIGURE 5.10
Behavior for varying ratio of flows to nodes in constant size network with 50 nodes.

average total runtime more than triples for CGTP from 130 flows (177.1 s) to 140 flows (625.0 s) and reaches 812.4 s for 150 flows.

In our evaluations, CGTP outperforms RILP, but we also observe that the region of scenarios that can be solved in practice with RILP is limited by the sheer size of the ILP model and the time spent on its construction. The iterative approach of CGTP does not exhibit this drawback.

Network Density Next, we investigate the performance of CGTP and the previously observed increase in runtime and runtime variance for larger flow sets. We use the same network topology with 50 nodes and with the same properties as in the previous section. We vary the number of flows from 25 to 200 with an increment of 25 flows per step (each with $t_{\text{cycle}} = 1000\,\mu\text{s}$, $t_{\text{resv}} = t_{\text{pkt}} = 5\,\mu\text{s}$), effectively changing the total ratio of flows to nodes in the data network from 1:2 to 4:1. We generate and solve 40 problem instances per step on the PCsmall compute nodes.

In Fig. 5.10, we plot the total runtime over the number of vertices of the conflict graph at the time when the solution was found. Each point in Fig. 5.10 represents an individual scenario. Points are colored according to the ratio of flows to nodes in the network. We observe comparably short mean runtimes for scenarios with 25 flows (12.0 s) to 100 flows (86.9 s) which form almost a line in the lower-left corner of the plot. However, from 125 flows (ratio 5:2) on, individual runtimes increase strongly and spread farther apart, indicating also an increasing variance of the runtimes. From 125 flows (ratio 5:2) on, we measured the following average runtimes: 297.5 s for 125 flows, 851.1 s for 150 flows, 1982.3 s for 175 flows, and 3134.7 s for 200 flows.

Two factors contribute to this behavior. Firstly, by increasing the number of flows while maintaining the network size, the network load and thus the difficulty of the traffic planning problem is increasing since conflicts become more likely. Secondly, we also see implementation-

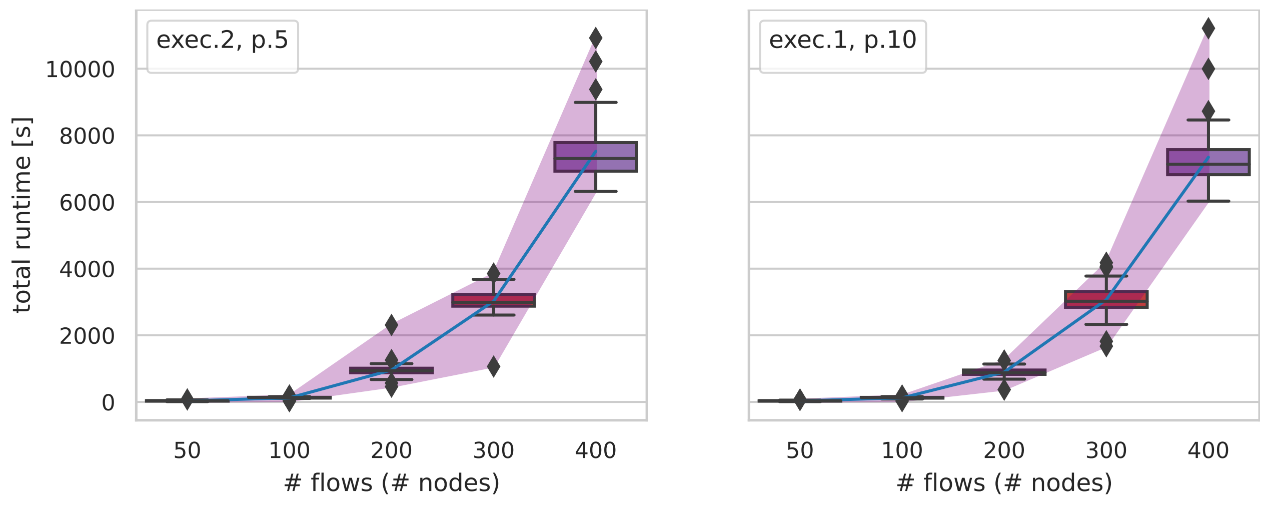


FIGURE 5.11 Total runtime for increasing problem size with different intermediate ILP settings. Left plot: after two successive executions of intermediate ILP for at most 5 min, suspend intermediate ILP for 5 iterations, right plot: at most 1 execution of intermediate ILP every 10 iterations for at most 10 min.

specific effects. As explained in Sec. 5.2, our proof-of-concept implementation of CGTP is an iterative approach which by default uses the adapted maximal independent vertex set algorithm in each iteration and, depending on the trigger condition, also executes an intermediate ILP. For scenarios with 100 or more flows, the intermediate ILP executions did not only get triggered multiple times, but we also observed several instances where the intermediate ILP solving had to be aborted due to hitting the runtime limit. The higher the number of flows, the more often this happened. Every intermediate ILP execution which was aborted due to the runtime limit adds 300 s to the total runtime. Therefore, for similar-sized conflict graphs, the number of unsuccessful intermediate ILP executions can cause strongly varying total runtimes.

Scaling Finally, we keep the ratio of flows to nodes in the network fixed to 1:1, that is, the more flows, the larger the network. We vary the number of flows (and nodes in the network) from 50 to 400, (each with $t_{\text{cycle}} = 1000\,\mu\text{s}$, $t_{\text{resv}} = t_{\text{pkt}} = 5\,\mu\text{s}$). We generate and solve in total 80 problem instances per step on PCsmall compute nodes.

This time, we use two different settings for the ILP trigger limiter. Half of the problem instances (left plot in Fig. 5.11) use the default setting where the intermediate ILP can be executed at most twice in succession, and the ILP solver runtime is limited to 5 min before ILP triggering pauses for five iterations (labeled *exec.2, p.5*). For the other half of the problem instances (right plot in Fig. 5.11), the intermediate ILP can be triggered at most once every ten iterations and is allowed to run at most 10 min (labeled *exec.1, p.10*).

Despite the different settings for the ILP trigger limiter, we observe similar values for the

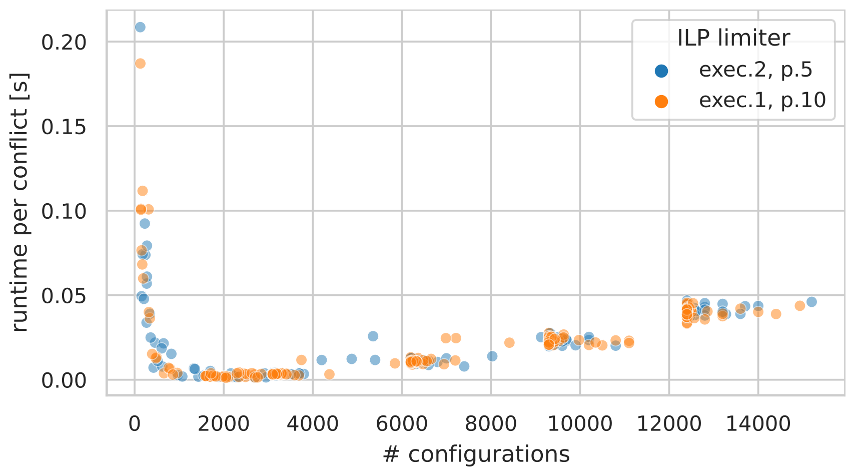


FIGURE 5.12 Runtime normalized to the number of conflicts (edges) in the conflict graph for increasing problem sizes.

average runtimes as well as the variance of the runtime. The increase in runtime is dominated by two factors, namely, the time it takes to grow the graph and the time it takes to search for the independent vertex set. The growing of the conflict graph, or more exact, the time it takes to insert a single configuration into the conflict graph grows with an increasing number of configurations in the conflict graph. The time required for searching the independent vertex set also grows with an increasing conflict graph but has the possibility to vary much stronger, depending on the actual structure of the conflict graph, the rounds in the fast algorithm, and the time spent in the max-cover algorithm.

In Fig. 5.12, we plot the total runtime normalized to the number of conflicts in $\boldsymbol{G}_c$ over the size of $\boldsymbol{G}_c$. Also, here the two settings for the intermediate ILP behave similarly. For small numbers of configurations ($\sim 10^2$ to 10^3 vertices), the overhead (I/O, path computation) dominates, hence we see a decline for increasing number of configurations for these problem instances. For larger conflict graphs, the time spent for growing the conflict graph and finding the solution dominates, and we observe a similar clustering (visible, for example, around the average number of configurations for 200 and more flow, see Tab. 5.2) as in Fig. 5.10. The average size of the conflict graph is similar for both settings (see Tab. 5.2), and there are few outliers in Fig. 5.12. This indicates that the combination of the fast algorithm, the max-cover algorithm, and the respective trigger rules succeed in consistently finding solutions without "over-growing the graph".

5.2.7 Evaluation Summary

We have seen in our evaluations that the conflict graph-based modeling of the traffic planning problem allows for an efficient implementation. Compared to the ILP-based approaches, our CGTP implementation could solve larger scenarios on less powerful machines in less time.

In particular, the combination of fast algorithm and max-cover algorithm and assorted

TABLE 5.2 Mean and standard deviation of conflict-graph dimensions for increasing problem sizes (cf. Fig. 5.12).

$\vert\mathcal{F}\vert$,$\vert\mathcal{N}\vert$	50	100	200	300	400
# configurations (mean)	$1.297{\cdot}10^3$	$3.043{\cdot}10^3$	$6.256{\cdot}10^3$	$9.438{\cdot}10^3$	$1.267{\cdot}10^4$
# configurations (std)	824	616	464	762	584
# conflicts (mean)	$1.445{\cdot}10^4$	$3.633{\cdot}10^4$	$8.368{\cdot}10^4$	$1.333{\cdot}10^5$	$1.838{\cdot}10^5$
# conflicts (std)	$1.338{\cdot}10^4$	$1.038{\cdot}10^4$	$1.187{\cdot}10^4$	$2.293{\cdot}10^4$	$2.000{\cdot}10^4$

triggering heuristics and completion heuristics can exploit the fact that we can find a traffic plan without constructing the conflict graph that contains every possible configuration for each flow.

This means the conflict graph and its reasonably intuitive representation of the traffic-planning problem provide an appropriate concept to explore the search space for traffic planning with the zero-queuing principle.

5.3 Dynamic QoS-Aware Traffic Planning with Conflict Graphs

So far, we have focused solely on offline traffic planning with a separation between the traffic planning phase and the runtime phase of the network. Once we have computed a valid traffic plan, traffic planning was done. Then, during runtime, all nodes simply act according to the computed traffic plan. This is what we refer to as static traffic planning in the sense that we compute a traffic plan for an a priori known, fixed set of flows once.

However, a general trend which is postulated, for example, for the domain of industrial automation, in both, industry white-papers [KUK16] and research articles [Raa+17; Pri+18], is a trend away from static environments towards dynamic, reconfigurable scenarios, for example, when attaching or physically moving a sensor or machine in a smart factory. The term *plug-and-produce* is frequently used in the context of Industry 4.0 to describe the need for flexible production facilities where devices can be added quickly and configured automatically, which also includes the adaption of network schedules and routes to integrate new devices into the network.

Similar challenges might arise in the future in the backbone of vehicular communication scenarios [Kar+11; Sad+20] where data flows have to be re-configured once a mobile node is handed over from one access point to another.

Some foundations for network management in such dynamic environments are already laid out in the form of concepts such as an SDN-like centralized network controller and associated management abstractions that provide a centralized view onto the network and can also be used to automate and deploy network configurations. Thus, the basic primitives for dynamically changing routes and schedules of nodes in a softwareized time-sensitive network exist. However, another required functionality is traffic planning that supports dynamic scenarios where flows can be added dynamically—which is more than just "static planning done fast".

In addition to the general challenges of network updates [Rei+12], we also have to consider the QoS requirements of the time-triggered flows during the transitions between traffic plans. For example, if we naively execute static traffic planning again from scratch each time new flows are to be added, this provides no control over the service degradation experienced by an individual flow. Furthermore, we usually cannot even guarantee that the new plan still includes all priorly active flows. Worse yet, we might even have to suspend the emission of new packets for some time to ensure that there is no interference induced by old packets—resulting in a "stop-and-go"-reconfiguration, or, if technically possible, we have to drop old packets. All of this boils down to the insight that dynamic traffic planning requires taking the current traffic flows into account when computing a new traffic plan.

We can identify two paradigms for how to accomplish this, namely, defensive planning and offensive planning.

Defensive planning does not change the configuration of active flows in order to add new flows. In other words, defensive planning takes the configuration of active flows as granted and uses the remaining network resources for routing and scheduling new flows. While defensive planning never affects the quality of service (QoS) of active flows, it might utilize network resources in a suboptimal way. For example, defensive planning might result in scenarios where new flows have to be rejected simply because the active flows have "fragmented" the remaining network resources, and we cannot "correct" past scheduling and routing decisions.

Obviously, this issue can be mitigated by what amounts to *offensive planning* where (some) active flows can be reconfigured when adding new flows. However, this introduces additional challenges: While offensive planning allows for better utilization of network resources, the transitory effects of reconfigurations possibly introduce short-term QoS degradation [Li+19], and the transition phase has to be temporally bounded to guarantee deterministic communication in the long run. This means offensive planning makes only sense if we can control, both the degree and duration of a QoS degradation, and it requires applications that can tolerate controlled QoS fluctuations. Such applications, for instance, correspond to the cyclic-synchronous traffic pattern where the emphasis is on latency and some jitter is acceptable.

Next, we propose a dynamic traffic planning approach that considers both, the temporal (scheduling) and spatial (routing) aspects of traffic planning. Our approach allows reconfiguring active flows (offensive planning) without packet drops or pausing of active flows while also providing bounded QoS degradation in terms of packet timings and packet-reorderings during the transition to the new traffic plan. These bounds can be given on a per-flow basis. This allows differentiating between a subset of active flows (belonging to jitter-resilient applications) that can be reconfigured, while the remaining active flows (belonging to jitter-sensitive applications) are exempt from reconfigurations.

As we will see, the conflict-graph representation from the previous sections turns out to be advantageous for dynamic traffic planning, too.

5.3.1 Dynamic Traffic Planning: What Changes?

We can interpret dynamic traffic planning as a generalization of the traffic planning task discussed in the previous section and chapter. In particular, we extend the conflict-graph concepts from Sec. 5.1 where the basic idea of representing flow configurations and their relations by vertices and edges in the conflict graph remains.

In the dynamic traffic planning task, there will be different sets of flows active in the network at different points in time. This requires us to be able to distinguish between different versions for active flows, and we have to deal with the transition from one traffic plan to the next traffic plan, which was not needed for static traffic planning. We define this additional terminology and concepts next.

Flow Dynamics

When we talk about dynamic traffic planning, we assume that there are already some flows that are active in the network according to some traffic plan. Whenever a set of new flows are to be added to the network, the planner computes a new traffic plan. Therefore, over time the planner computes a sequence of traffic plans, where a newly computed traffic plan becomes the current plan, which replaces the old one, see Fig. 5.13. In the following, we denote the current traffic plan by $\mathcal{P}$ (which becomes the old traffic plan after the update) and the new traffic plan by $\mathcal{P}'$.

We know that a traffic plan $\mathcal{P}$ describes when and along which routes the packets of a flow set are transmitted. In the dynamic case, the set of flows that transmit packets changes too with every new traffic plan. Therefore, we will use $\mathit{Active}\mathcal{F}(\mathcal{P})$ to denote the set of active flows where an active flow is covered or admitted by $\mathcal{P}$. For example, in Fig. 5.13, $\mathit{Active}\mathcal{F}(\mathcal{P}) = \{f_a, f_b, f_c\}$, $\mathit{Active}\mathcal{F}(\mathcal{P}') = \mathit{Active}\mathcal{F}(\mathcal{P}) \cup \{f_d\}$.

For the exposition of our approach, we concentrate on the case that the planner only attempts to add new flows to the network. Active flows can also be removed (see f_c in Fig. 5.13), which means that the source node stops sending packets and the planner is notified that it can reassign the resource previously reserved for that flow when computing the next traffic plan update. Removing active flows is trivial from the perspective of the planner, see Sec. 5.3.4.

Let $\mathit{Req}\mathcal{F}(\mathcal{P})$ be the set of flows that the applications request to be added to the network, that is, $\mathit{Req}\mathcal{F}(\mathcal{P})$ are the requests received by the planner while $\mathcal{P}$ is valid. By processing

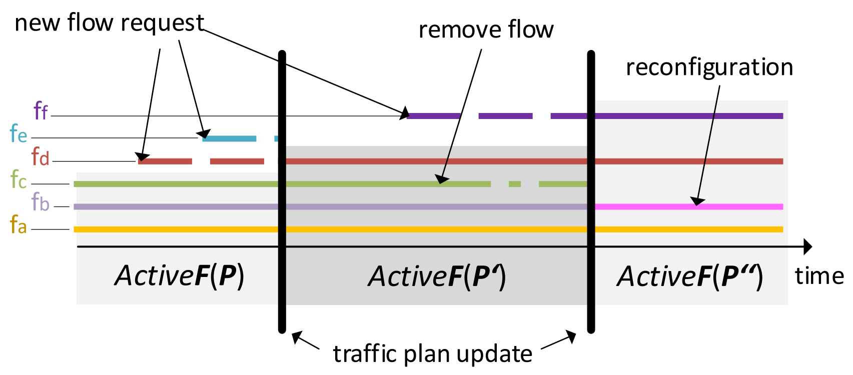


FIGURE 5.13 The planner computes a sequence of traffic plans. During the first traffic plan update, out of the two flow requests ($Req\mathcal{F}(\mathcal{P}') = \{f_d, f_e\}$), only f_d can be admitted (which requires a reconfiguration of f_b) and f_d is added to the set of active flows $Active\mathcal{F}(\mathcal{P}')$, while f_e could not be scheduled and is rejected. The second update removes f_c, which is not needed by the application any more.

these requests, the planner generates the new traffic plan $\mathcal{P}'$ with $Active\mathcal{F}(\mathcal{P}') \setminus Active\mathcal{F}(\mathcal{P}) \subseteq Req\mathcal{F}(\mathcal{P})$ where $\mathcal{P}$ is the plan preceding $\mathcal{P}'$.

Not only do the flows in the set of active flows change, but offensive planning allows reconfiguring active flows while a new plan is established (see f_b in Fig. 5.13). Consequently, a flow may have different configurations over time. The notion $\text{config}(f, \mathcal{P})$ identifies the configuration of flow f while $\mathcal{P}$ is valid. If the planner decides to reconfigure f in a succeeding plan $\mathcal{P}'$, then $\text{config}(f, \mathcal{P}) \neq \text{config}(f, \mathcal{P}')$. We say that each reconfiguration of a flow generates a new version of this flow, where each version exists as long as the corresponding plan is valid. To be able to distinguish the various flow versions, we will use the notion $\langle f, \mathcal{P}\rangle$, where flow f is associated with plan $\mathcal{P}$. This means $\text{config}(f, \mathcal{P})$ is the configuration of the flow version $\langle f, \mathcal{P}\rangle$.

In general, it can happen that a flow in $Req\mathcal{F}(\mathcal{P})$ may not be admitted due to network resource limitations (see f_e in Fig. 5.13). However, once a flow f has been admitted into the set of active flows, the end-to-end deadline constraints, and jitter-bounds are guaranteed until the end of its lifetime. This explicitly includes all future traffic plans and possible reconfigurations of f. For example, f_b in Fig. 5.13 is an active flow whose lifetime spans over the two depicted traffic plan updates: during the whole time, the source node of f_b can send packets that are guaranteed to arrive at the destination node in time. We expect a once established time-triggered flow to be "relatively" long-lived, that is, it persists for thousands or even millions of transmission cycles.

We focus specifically on the problem of *computing* the traffic plan. However, ultimately our traffic planning approach is only one building block of the controller, which likely will need to incorporate additional application-specific logic and interfaces. For instance, the event that triggers the computation of a new traffic plan might depend on the application. The traffic plan update could be initiated by applications issuing a flow request or a batch of flow requests. Alternatively, the planner could collect incoming flow requests and decide itself when to compute a new traffic plan, for example, periodically, or when the number of flow requests reaches a certain threshold. Consequently, we do not give any guarantees on the response time for flow requests. In other words, our goal is a *real-time data plane*, not a real-time control plane, although we strive for fast processing of requests through efficient planning. We will show later that we can parameterize our approach with regard to the trade-off between runtime and schedulability.

We assume that we do not have any prior knowledge as to how the flow set changes over time, else we could pre-compute a sequence of traffic plans. In the following, we assume that the planner processes a flow request only once, that is, either admits the new flow or rejects it. In principle, we can also implement other strategies. For example, rejected flow requests could be retained and each flow removal might then trigger re-admission attempts for those pending flow requests until admission is possible or the flow requests expires.

Additional Node Capabilities

Similar to the static case, the necessary flow information corresponding to config $(f, \mathcal{P})$ has to be made available to the source node and the bridges along the route of f in the network before a node sends or forwards packets of a particular flow f or flow version. Here, we assume that this flow information is propagated by a network controller. Different from the static case, the flow information in the nodes, in particular routes and schedules, now are expected to be updated at runtime by the controller.

However, flow information updates shall not affect in-flight packets to prevent network update issues such as black-holing. This means bridges temporarily may have multiple sets of flow information for each version of a reconfigured flow and deliver each packet according to the flow configuration that was used by the source node when sending the packet. To this end, the network provides a mechanism that allows distinguishing packets not only on the level of a flow but also on the level of a flow version. This can be achieved, for example, by tagging packets with additional metadata or using address schemes [Rei+12].

The controller lazily purges obsolete flow information sets from the nodes once all old packets have been received by the destination nodes. In this context, obsolete flow information sets refers to flows that either have been removed from $\mathit{Active}\mathcal{F}(\mathcal{P}')$ or have been reconfigured. In the latter case, while the flow itself remains active, the flow information sets for old flow versions $\langle f, \mathcal{P}\rangle$ are not used anymore. To perform this "garbage collection", the controller has to track the flow information sets associated with each traffic plan and then has to delete the corresponding routing entries, drop now unused scheduling entries, etc.

Conflict Graph Versions

As before, we encode the relations and constraints between the configurations in a conflict graph. Now, a new version of the conflict graph is generated by modifying the old one whenever a new traffic plan is to be computed.

A new version of the conflict graph is generated by adding/removing configurations to the previous version and locking or unlocking configurations as explained below. In the following, $\boldsymbol{G}_c(\mathcal{P})$ identifies the version of the conflict graph that is associated with plan $\mathcal{P}$, that is, $\boldsymbol{G}_c(\mathcal{P})$ has been used to compute $\mathcal{P}$. Analogously, we use $\text{cand}(f, \mathcal{P})$ to denote the candidate set of flow f in $\boldsymbol{G}_c(\mathcal{P})$.

We will see that the modifications that we perform on the conflict graph for dynamic traffic planning far exceed the expansion of the conflict graph in Sec. 5.2. Indeed, the expansion of the conflict graph in the iterations of the CGTP will become a suboperation in the construction of the conflict graph.

Relation: $\mathit{Active}\mathcal{F}(\mathcal{P})$ and Global Traffic Plan $\mathcal{P}$

In Thm. 5.5, we established the relation between the independent vertex in the conflict graph and the global traffic plan. In dynamic scenarios, the traffic plan and the active flows change in lockstep.

DEFINITION 5.7 (Traffic Plan)
A traffic plan $\mathcal{P}$ is a set of conflict-free configurations for $\mathit{Active}\mathcal{F}(\mathcal{P})$ in $\boldsymbol{G}_c(\mathcal{P})$ that contains exactly one configuration for each flow in $\mathit{Active}\mathcal{F}(\mathcal{P})$.

We say that traffic plan $\mathcal{P}$ admits flow f if $\mathcal{P}$ contains a configuration for f, that is,

$$\text{admits}(f, \mathcal{P}) = \begin{cases} 1 : \text{if } \exists \text{ configuration for } f \in \mathcal{P} \\ 0 : \text{if } \nexists \text{ configuration for } f \in \mathcal{P} \end{cases} . \tag{5.4}$$

As before, any $\boldsymbol{G}_c(\mathcal{P})$ possibly contains many different sets of conflict-free configurations. We specify the objective that guides the search for the configurations forming the new traffic plan $\mathcal{P}'$ in Sec. 5.3.2.

Transition Interference

Assume that we have a new traffic plan $\mathcal{P}'$ and want to replace the traffic plan $\mathcal{P}$ with $\mathcal{P}'$. We call the time a new traffic plan $\mathcal{P}'$ becomes valid its activation time t_{act}. To ensure that no source node is in the middle of a packet transmission at t_{act} and since the flows in $\mathit{Active}\mathcal{F}(\mathcal{P}')$

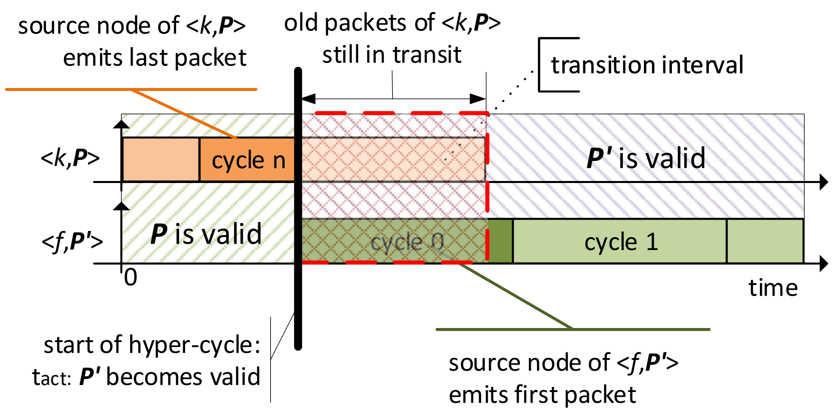

source node of <k,P> emits last packet
old packets of <k,P> still in transit
transition interval
<k,P>
cycle n
P' is valid
<f,P'>
P is valid
cycle 0
cycle 1
0
time
start of hyper-cycle: tact: P' becomes valid
source node of <f,P'> emits first packet

FIGURE 5.14 The possibility of transition interference between old packets of flow k from plan $\mathcal{P}$ and new packets of flow f from $\mathcal{P}'$ is limited to the *transition interval.*

may have different cycle times, we choose the start of a fresh hyper-cycle of all active flows in $\mathit{Active}\mathcal{F}(\mathcal{P})$ as activation time, that is,

$$t_{\text{act}} = t_0 + k \cdot \text{lcm}_{f \in \mathit{Active}\mathcal{F}(\mathcal{P})}\left(t_{\text{cycle}}(f)\right). \tag{5.5}$$

From the perspective of an active flow, it is perfectly fine for t_{act} to be many transmission cycles in the future because all flows in $\mathit{Active}\mathcal{F}(\mathcal{P})$ remain active with their established configurations from $\mathcal{P}$ until t_{act}.

Then at t_{act} the new traffic plan $\mathcal{P}'$ becomes valid. This means, from t_{act} on, we would like *all* the packets of each flow f in $\mathit{Active}\mathcal{F}(\mathcal{P}')$ to be transmitted according to $\text{config}(f, \mathcal{P}') = (f, \phi', \pi')$. If all packets sent according to the old plan $\mathcal{P}$ are delivered to the destination nodes before t_{act}, this would simplify our problem a lot. It implies that the network would be empty at t_{act} which would allow us to compute each new traffic plan $\mathcal{P}'$ independently of $\mathcal{P}$. In other words, under these circumstances computing a new traffic plan is equivalent to solving a new static traffic planning instance.

However, this simplified "solution" is inadequate, because it severely restricts the network utilization. Ultimately, for each packet to arrive within the transmission cycle it was sent by the source node, $\phi \leq t_{\text{cycle}} - t_{\text{e2e}} - t_{\text{resv}}$ has to hold. This means the transmission frequency and network utilization are inherently coupled to the network size.

Therefore, we provide a solution for the general case where there can be remaining yet-to-be-delivered packets from $\langle k, \mathcal{P} \rangle$, $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$ in the network when the new traffic plan $\mathcal{P}'$ becomes valid. In the general case, we have to account for the possibility of *transition interference*, that is, flow interference between the remaining old packets from $\langle k, \mathcal{P} \rangle$, $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$,

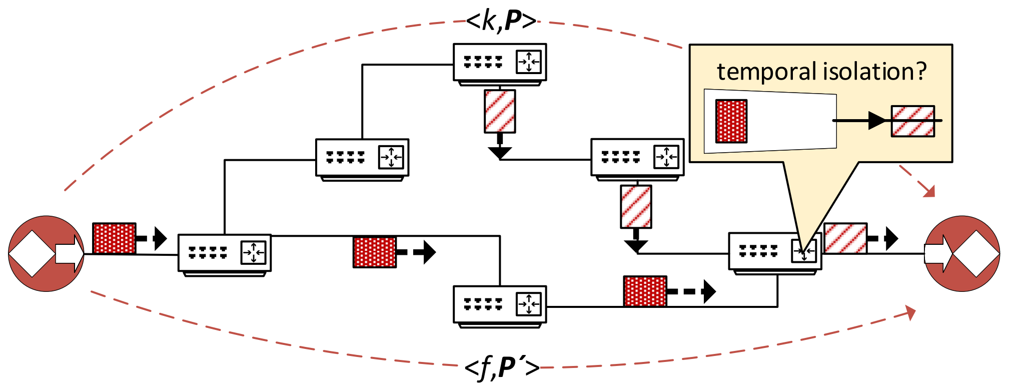


FIGURE 5.15 Scenario: packets from $\mathcal{P}'$ may interfere with yet-to-be-delivered packets from $\mathcal{P}$.

and the first packets from $\langle f, \mathcal{P}' \rangle$, $f \in \mathit{Active}\mathcal{F}(\mathcal{P}')$, when computing the new traffic plan $\mathcal{P}'$.

DEFINITION 5.8 (Transition Interference)
Let $\mathcal{P}$ denote the traffic plan valid before t_{act}, and let $\mathcal{P}'$ denote the new traffic plan with activation time t_{act}. Transition interference is defined to be between $\langle k, \mathcal{P} \rangle$, $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$ and $\langle f, \mathcal{P}' \rangle$, $f \in \mathit{Active}\mathcal{F}(\mathcal{P}')$ if any packet sent by the source node of $\langle k, \mathcal{P} \rangle$ prior to t_{act} is buffered due to a packet sent by the source node of $\langle f, \mathcal{P}' \rangle$ from t_{act} on, or vice versa.

We can algorithmically check whether transition interference occurs between $\langle k, \mathcal{P} \rangle$ and $\langle f, \mathcal{P}' \rangle$: If the route of $\langle k, \mathcal{P} \rangle$ and the route of $\langle f, \mathcal{P}' \rangle$ have common links, and any packet from $\langle k, \mathcal{P} \rangle$ sent before t_{act} is scheduled for transmission at any point in time when any packet sent by $\langle f, \mathcal{P}' \rangle$ from t_{act} on is scheduled on a common link, transition interference occurs, see Fig. 5.15.

Transition interference is restricted to a possible zero-length time interval, the *transition interval* t_{transit}, which starts at t_{act} and is upper-bounded by the end-to-end delay of $\langle k, \mathcal{P} \rangle$, see Fig. 5.14. Now we can also clarify the "purging" of obsolete flow information sets. It is easy to see that once the transition intervals of all flows in $\mathit{Active}\mathcal{F}(\mathcal{P})$ have expired, the controller can safely delete all unused flow information sets associated with $\mathcal{P}$ from the nodes in the network.

In the following, we only consider the case where transition intervals of different traffic plan updates are mutually exclusive, that is, we only consider one traffic plan $\mathcal{P}$ and its immediate successor $\mathcal{P}'$.

5.3.2 Problem Statement

From a high-level view, the dynamic traffic planning task can be described as follows: Let $\mathcal{P}$ be the traffic plan for the flows in $\mathit{Active}\mathcal{F}(\mathcal{P})$ computed from $\boldsymbol{G}_c(\mathcal{P})$, that is, the packets of flows in $\mathit{Active}\mathcal{F}(\mathcal{P})$ are transmitted according to $\mathcal{P}$. Now applications request that a set of flows $\mathit{Req}\mathcal{F}(\mathcal{P})$ be added to the network. Compute a new traffic plan $\mathcal{P}'$.

To compute $\mathcal{P}'$, we have to construct the new conflict graph $\boldsymbol{G}_c(\mathcal{P}')$ and search for a set of conflict-free configurations (that is, independent vertices in the $\boldsymbol{G}_c(\mathcal{P}')$). Previously, we were ultimately searching for any traffic plan that admits all flows which would mean a traffic plan that admits $\mathit{Active}\mathcal{F}(\mathcal{P}) \cup \mathit{Req}\mathcal{F}(\mathcal{P})$. Likewise, we would prefer to find a set of conflict-free configurations that admits all flows in $\mathit{Active}\mathcal{F}(\mathcal{P}) \cup \mathit{Req}\mathcal{F}(\mathcal{P})$ in the dynamic case, but this may not always be possible or not desirable if it may take too long. Different from the ILP formulations, we can exploit Thm. 5.5 in the conflict-graph-based approaches to try to efficiently compute a traffic plan for “as many as flows as possible” with a heuristic.

Here, we want to ensure that once the planner adds a flow to $\mathit{Active}\mathcal{F}(\mathcal{P})$, this flow remains active until the application itself indicates the flow can be removed. This means the planner shall not unsolicitedly evict any flow from $\mathit{Active}\mathcal{F}(\mathcal{P})$. Consequently, the new traffic plan shall include a configuration for *every* flow in $\mathit{Active}\mathcal{F}(\mathcal{P})$ and as many new flows as possible from $\mathit{Req}\mathcal{F}(\mathcal{P})$. The difference in the importance of active flows and new flows results in the following objective.

DEFINITION 5.9 (Dynamic Traffic Planning Objective)
Let $\boldsymbol{G}_c(\mathcal{P}')$ be a conflict graph that contains candidate configurations for all flows in $\mathit{Active}\mathcal{F}(\mathcal{P}) \cup \mathit{Req}\mathcal{F}(\mathcal{P})$ with $\mathit{Active}\mathcal{F}(\mathcal{P}) \cap \mathit{Req}\mathcal{F}(\mathcal{P}) = \emptyset$. We want to find a new traffic plan $\mathcal{P}' \subseteq \bigcup_f \mathrm{cand}(f, \mathcal{P}')$ that maximizes the objective

$$\max_{\mathcal{P}'} \sum_{f \in \mathit{Active}\mathcal{F}(\mathcal{P})} \mathrm{admits}(f, \mathcal{P}') + \sum_{f \in \mathit{Req}\mathcal{F}(\mathcal{P})} \frac{\mathrm{admits}(f, \mathcal{P}')}{|\mathit{Active}\mathcal{F}(\mathcal{P}) \cup \mathit{Req}\mathcal{F}(\mathcal{P})|}. \tag{5.6}$$

The factor $\frac{1}{|\mathit{Active}\mathcal{F}(\mathcal{P}) \cup \mathit{Req}\mathcal{F}(\mathcal{P})|}$ in Eq. (5.6) discounts the relative importance of flows in $\mathit{Req}\mathcal{F}(\mathcal{P})$. This means any flow in $\mathit{Active}\mathcal{F}(\mathcal{P})$ is more “valuable” than all flows $\mathit{Req}\mathcal{F}(\mathcal{P})$ taken together.

From a graph-theoretic perspective, this is a specific colorful independent vertex set problem where we want to find a set of independent vertices and each color has either unit weight or a weight inversely proportional to the total amount of colors.

5.3.3 Constructing a New Version of the Conflict Graph

Next, we explain how the planner constructs the new conflict graph $\boldsymbol{G}_c(\mathcal{P}')$ when processing $\mathit{Req}\mathcal{F}(\mathcal{P})$. With offensive planning, the planner has to consider both flow versions $\langle f, \mathcal{P}\rangle$ and $\langle f, \mathcal{P}'\rangle$ for each active flow f where $\mathcal{P}'$ indicates the new plan replacing $\mathcal{P}$.

Overview

Remember that after installing $\mathcal{P}'$, packets from both $\langle f, \mathcal{P}\rangle$ and $\langle f, \mathcal{P}'\rangle$ may be in the network for some time. Therefore, it is not sufficient to only avoid conflicts between any new flow versions $\langle f, \mathcal{P}'\rangle$, which would suffice in the static case. Instead, the planner additionally has to ensure that any $\langle k, \mathcal{P}\rangle$ does not interfere with any other flow $\langle f, \mathcal{P}'\rangle$. This leads to the following requirements:

R1 For all $k, f \in \mathit{Active}\mathcal{F}(\mathcal{P}')$, $\langle k, \mathcal{P}'\rangle$ does not interfere with $\langle f, \mathcal{P}'\rangle$ for $k \neq f$. That is, active flows of the new plan do not interfere.

R2 For all $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$ and $f \in \mathit{Active}\mathcal{F}(\mathcal{P}') \setminus \mathit{Active}\mathcal{F}(\mathcal{P})$, $\langle k, \mathcal{P}\rangle$ does not interfere with $\langle f, \mathcal{P}'\rangle$. That is, the active flows of the old plan do not interfere with the flows added to the new plan.

R3 For all $k, f \in \mathit{Active}\mathcal{F}(\mathcal{P})$, $\langle k, \mathcal{P}\rangle$ does not interfere with $\langle f, \mathcal{P}'\rangle$. That is, the active flows of the old plan do not interfere with their versions in the new plan. Note, if $f = k$, then we consider the old and new version of the same flow, $\langle f, \mathcal{P}\rangle$ and $\langle f, \mathcal{P}'\rangle$. Since packets from $\langle k, \mathcal{P}\rangle$ and $\langle f, \mathcal{P}'\rangle$ may populate the network at the same time, we have to ensure that no transition interference occurs.

R1 is a basic requirement, which is sufficient for defensive planning. Allowing for reconfigurations of active flows, however, we must add R2 and R3 to ensure the zero-queuing constraint. In order to fulfill R1, the planner has to make sure that the configurations associated with flows in $\mathit{Active}\mathcal{F}(\mathcal{P}')$ are not in conflict. To this end, the planner first of all has to *add candidate configurations* for each flow in $\mathit{Req}\mathcal{F}(\mathcal{P})$ to the conflict graph. The expanded conflict graph is the basis for solving the planning problem as stated above.

Now let us see, how additionally R2 and R3 can be guaranteed. Note that the interference of two flows/flow versions implies that their configurations are conflicting, while the opposite is not true. For example, if we ensure that the packets of one flow or flow version only enter the network after all packets of another flow have already left the network, these two flows do not interfere even if their configurations are conflicting. Therefore, we can fulfill R2 by *delaying the activation* of the added flows until all packets that belong to flow versions of $\mathcal{P}$ have died out. Assume the transmission of packets of each flow $\langle k, \mathcal{P}\rangle$ with $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$ is stopped at time t. All packets sent by these flows have left the network by $t + \tau$, where $\tau \geq \max_k(t_{\text{e2e}}(k) - t_{\text{cycle}}(k))$ is greater or equal to the largest transition interval t_{transit} of any flow $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$. If the transmission of each $\langle f, \mathcal{P}'\rangle$ with $f \in \mathit{Active}\mathcal{F}(\mathcal{P}') \setminus \mathit{Active}\mathcal{F}(\mathcal{P})$ does not start before $t + \tau$, then no flow $\langle k, \mathcal{P}\rangle$ interferes with any flow added to $\mathcal{P}'$ even if their configurations conflict. Hence, we prevent transition interference between flows from $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$ and $f \in \mathit{Active}\mathcal{F}(\mathcal{P}') \setminus \mathit{Active}\mathcal{F}(\mathcal{P}) \subseteq \mathit{Req}\mathcal{F}(\mathcal{P})$ from ever happening by adjusting the activation time of new flows, that is, outside of the actual computation of $\mathcal{P}'$. Roughly speaking: the controller tells the source nodes of the new flows that their actual activation time is not t_{act} but $t_{\text{act}} + \lceil \frac{t_{\text{transit}}}{t_{\text{cycle}}} \rceil \cdot t_{\text{cycle}}$ with t_{transit} the duration of the longest transition interval of any flow in $\mathit{Active}\mathcal{F}\,(\mathcal{P})$. In the example in Fig. 5.14, the first "useable" transmission cycle for $\langle f, \mathcal{P}' \rangle$ is cycle 1. For the source node of a new flow, this has the same effect as if it took a few transmission cycles more to compute $\mathcal{P}'$. Therefore, fulfilling R2 requires no special treatment when constructing the conflict graph.

Note that this only applies to transition interference involving flows that are newly added to $\mathit{Active}\mathcal{F}\,(\mathcal{P}')$. Fulfilling R3 the same way as R2 is definitely not desirable since delaying data packets of an active flow might degrade QoS substantially. In simplified terms, our solution for R2—"turning on the machines a little bit later"—does not work for R3, because "the machines are already running". Therefore, we have to make sure that the planner cannot select configurations for active flows which conflict with the old configurations of these flows. We achieve this by *locking those configurations* that the planner must not select for the new plan. Note that the configurations to be locked depend on the old configuration of the active flows. Since the configuration of an active flow may change from plan to plan, configurations are unlocked each time after the new plan has been computed.

Adding Candidate Configurations

Since $\boldsymbol{G}_c\,(\mathcal{P})$ only contains candidates for flows in $\mathit{Active}\mathcal{F}\,(\mathcal{P})$, the planner has to generate and insert new candidate configurations for all flows in $\mathit{Req}\mathcal{F}\,(\mathcal{P})$ to $\boldsymbol{G}_c\,(\mathcal{P}')$. In principle, the planner could add all candidates for each flow to $\boldsymbol{G}_c\,(\mathcal{P}')$. However, we know from the previous section that we can find a traffic plan for all flows even if the conflict graph contains just a subset of all possible candidates of each flow. We exploit this and add only a limited number (upper-bounded by a parameter n_{ub}) of candidate configurations for each flow in $\mathit{Req}\mathcal{F}\,(\mathcal{P})$ to the conflict graph. From a graph-theoretic perspective, we are now also adding new colors to $\boldsymbol{G}_c\,(\mathcal{P})$ with each new flow.

Due to the nature of the traffic planning problem, it is often inherently difficult to identify promising candidate configurations a priori. We again use a heuristic where for each flow the planner has a stateful configuration generator that returns a new configuration each time it is invoked. Here, the generator walks through the ϕ-π-space in a more complex pattern: Starting at $\phi = 0$, $\pi = 0$, π is incremented until all configurations for a particular value of ϕ have been covered, before increasing ϕ by $\Delta\phi$. If $\phi + \Delta\phi$ exceeds the allowed range for ϕ, ϕ is reset to the next, lowest uncovered phase-value, see Fig. 5.16. In our case, the planner sets $\Delta\phi$ to the 75-th percentile of the transmission duration of all flows with candidate configurations in $\boldsymbol{G}_c\,(\mathcal{P}')$. Hence, we expect that one to two phase increments suffice to resolve a possible interference on a single link.

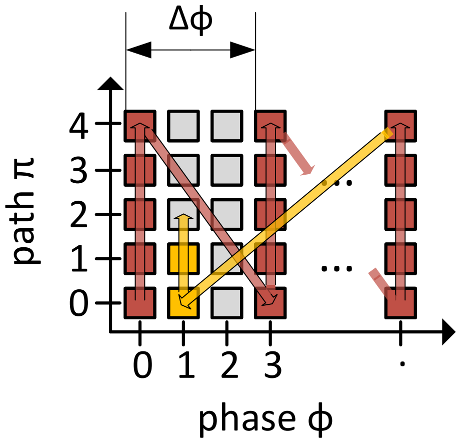


FIGURE 5.16
New candidates are generated by traversing the ϕ-π-space with $\Delta\phi$ the stride width in the ϕ-dimension.

While it is obvious that we have to add candidates for flows in $\mathit{Req}\mathcal{F}\,(\mathcal{P})$, we can also increase the set of candidates for flows in $\mathit{Active}\mathcal{F}\,(\mathcal{P})$ in $\boldsymbol{G}_c\,(\mathcal{P}')$, which is equivalent to the conflict-graph growth step in the CGTP from Sec. 5.2. The larger cand $(f, \mathcal{P}')$, the more of the solution space is covered, and the more alternatives for an active flow "to make way" for flow in $\mathit{Req}\mathcal{F}\,(\mathcal{P})$ exist. Here, we add more configurations for each flow in $\mathit{Active}\mathcal{F}\,(\mathcal{P})$ when processing $\mathit{Req}\mathcal{F}\,(\mathcal{P})$ until the configuration generator has traversed the ϕ-range of a flow for the first time. After that point, new configurations for active flows are only added if the planner could not admit all flows from the previous request, that is, a simple feedback loop controls the growth of the conflict graph.

In principle, we are free to choose to traverse the ϕ-π-space when adding configurations to the conflict graph. This means that the number of available configurations in the ϕ-π-space is decoupled from the size of the conflict graph. For example, if we want a time resolution of 1 ns instead of 1 µs, we can scale $\Delta\phi$ accordingly. Then, instead of advancing three steps in the ϕ-direction, we advance 3000 steps and end up with a similarly sized conflict graph.

This decoupling is an important feature because the effort to add a single configuration c to $\boldsymbol{G}_c\,(\mathcal{P}')$ scales with the number of configurations $\mathcal{V}$ already in the conflict graph ($\mathcal{O}(\mathcal{V})$) since we have to check for each existing configuration already in $\boldsymbol{G}_c\,(\mathcal{P}')$ whether it conflicts with c. However, we usually do *not* have to pay the *total* cost of constructing a conflict graph with $\mathcal{V}$ configurations, which scales with $\mathcal{O}(\mathcal{V}^2)$ unless all active flows are replaced by new flows in one update. When constructing $\boldsymbol{G}_c\,(\mathcal{P}')$, we start with $\boldsymbol{G}_c\,(\mathcal{P})$. Hence, the cost of adding a configuration effectively gets distributed over the lifetime of a flow. The planner could also add the additional configurations for flows in $\mathit{Active}\mathcal{F}\,(\mathcal{P})$ when idle, that is, while no $\mathit{Req}\mathcal{F}\,(\mathcal{P})$ has to be processed, to further accelerate the flow-request response time. Since the lifetime of active flows may differ, the planner should balance the number of candidates per flow in the long term, for example, by adding more configurations for younger flows.

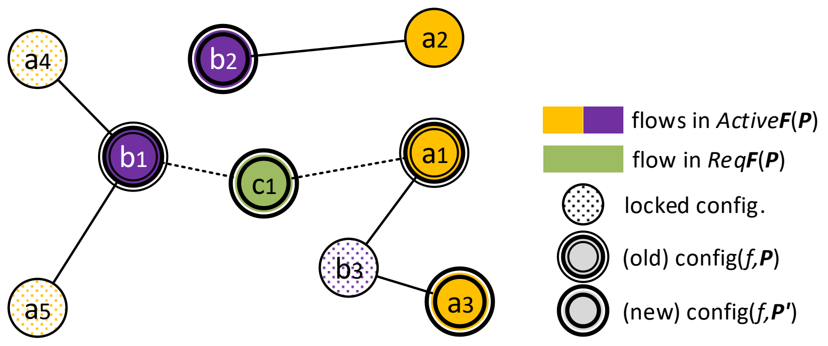


FIGURE 5.17 Example: 1-hop neighbors of active configurations a_1, b_1 are locked in $\boldsymbol{G}_c(\mathcal{P}')$, a_2, a_3, b_2 remain eligible for the active flows.

Locking Configurations

$\boldsymbol{G}_c(\mathcal{P}')$, generated as described in the previous section, may include configurations that cause interference between the old and new versions of active flows, violating R3. We yet have to make sure that the planner cannot select configurations for active flows that violate R3. In detail, the following two conditions have to hold:

1. For all $f \in \mathit{Active}\mathcal{F}(\mathcal{P})$, there exists no configuration $c \in \mathrm{cand}(f, \mathcal{P}')$ that results in a new version $\langle f, \mathcal{P}' \rangle$ that interferes with $\langle f, \mathcal{P} \rangle$. That is, the planner cannot assign a new configuration to f, such that the old and new version of f interfere.

2. For all $f, k \in \mathit{Active}\mathcal{F}(\mathcal{P})$, $k \neq f$ there exists no configuration $c \in \mathrm{cand}(f, \mathcal{P}')$ that results in a new version $\langle f, \mathcal{P}' \rangle$ which interferes with $\langle k, \mathcal{P} \rangle$. That is, the planner cannot assign a new configuration to f which results in interference with an old version of any other active flow k.

We achieve both conditions by locking all those configurations in $\boldsymbol{G}_c(\mathcal{P}')$ which would violate one of the conditions if selected by the planner. A locked configuration must not be selected by the planner.

To fulfill the first condition, for each $f \in \mathit{Active}\mathcal{F}(\mathcal{P})$, we have to visit each $c \in \mathrm{cand}(f, \mathcal{P}')$ in $\boldsymbol{G}_c(\mathcal{P}')$ to check whether a new version of f that uses the visited candidate c results in transition interference with the old version that uses $\mathrm{config}(f, \mathcal{P})$. If transition interference would occur, we have to lock the candidate. Obviously, this ensures that the planner cannot select a candidate c for $\mathrm{config}(f, \mathcal{P}')$ that could cause interference with $\langle f, \mathcal{P} \rangle$. Note that the configuration graph includes only edges for conflicting configurations of different flows. Therefore, we maintain an additional data structure that allows efficient access to the candidates of a particular flow.

To fulfill the second condition, for each $k \in \mathit{Active}\mathcal{F}(\mathcal{P})$, we have to visit all 1-hop neighbors of $\mathrm{config}(k, \mathcal{P})$ in the conflict graph, that is, all candidate configurations of any flow other than k that conflict with $\mathrm{config}(k, \mathcal{P})$. If a visited neighbor c is a configuration of another flow f in $\mathit{Active}\mathcal{F}(\mathcal{P})$, the planner checks for transition interference between a new version of f

with configuration c and $\langle k, \mathcal{P}\rangle$. If there would be interference, this configuration c is locked. Obviously, this prevents the planner from selecting for any other $f \in \mathit{Active}\mathcal{F}(\mathcal{P})$ a candidate configuration c for $\operatorname{config}(f, \mathcal{P}')$ that would cause interference. Note, here we just have to follow the edges of $\operatorname{config}(k, \mathcal{P})$ to access the candidates potentially to be locked, see Fig. 5.17.

(Permanently) Pinning Flows

To prevent the planner from ever reconfiguring an active flow f, we can *permanently* remove all configurations of f other than $\operatorname{config}(f, \mathcal{P})$ from $\boldsymbol{G}_c(\mathcal{P})$—*pinning* f to its configuration for its whole lifetime. Obviously, this ensures $\operatorname{config}(f, \mathcal{P}) = \operatorname{config}(f, \mathcal{P}')$ for future updates. In contrast, locking only temporarily excludes some candidates from $\mathcal{P}'$.

5.3.4 Computing the New Traffic Plan

Once the new conflict graph has been constructed by adding additional configurations and locking currently "forbidden" candidate configurations, we have to compute the traffic plan. From a graph-theoretic perspective, we can reduce the maximum (colorful) independent vertex set problem to the problem of finding a traffic plan which maximizes Eq. (5.6). This means computing a new traffic plan remains an NP-hard problem [Gar+79]. Since both, the adapted Luby's algorithm and the ILP formulation from Sec. 5.2 do not natively consider the objective from Eq. (5.6), we use a newly developed heuristic, the *Greedy Flow Heap Heuristic* (GFH). The name GFH draws from the fact that the objective Eq. (5.6) improves with every additional flow included in the new traffic plan. Next, we give a brief overview of the GFH and explain our strategy to compute the new traffic plan.

Greedy Flow Heap Heuristic

From a birds-eye view, GFH is an iterative greedy approach that in each iteration computes ratings for the configurations in $\boldsymbol{G}_c(\mathcal{P}')$ and adds the configurations with the best ratings to an intermediary set of conflict-free configurations $\mathcal{C}$. We say that $\mathcal{C}$ admits a flow f if $\mathcal{C}$ contains a configuration for f. When selecting the configuration to add to $\mathcal{C}$, GFH distinguishes between flows in $\mathit{Active}\mathcal{F}(\mathcal{P})$ and $\mathit{Req}\mathcal{F}(\mathcal{P})$ and prioritizes the former accordingly. During the GFH execution, $\mathcal{C}$ is always an independent vertex set in $\boldsymbol{G}_c(\mathcal{P}')$.

The GFH has a re-run mechanism. The re-run mechanism can be used to improve the number of flows admitted by $\mathcal{C}$ by re-running the configuration selection steps with different starting conditions, for example, if not all flows are in $\mathcal{C}$ after the first run. The number of re-runs (default $n_{\text{re-runs}} = 3$) can be parameterized.

For the same flow f, there may be multiple configurations in the final $\mathcal{C}$ returned by the GFH, but we can only include one of these (we need only one route and phase) in a traffic plan. In this case, we arbitrarily select one configuration for every flow with multiple configurations in $\mathcal{C}$ since all configurations in $\mathcal{C}$ are conflict-free. In other words, $\mathcal{P}' \subseteq \mathcal{C}$.

The GFH was presented first in [Fal+21] and has been developed by Heiko Geppert. For a more detailed explanation of the GFH, we refer to Appendix A.2.

Rejecting and Removing Flows

If $Req\mathcal{F}(\mathcal{P})$ contains flows that are not admitted by $\mathcal{C}$, the planner rejects the corresponding flows. All candidate configurations for a rejected flow are consequently purged from $\boldsymbol{G}_c(\mathcal{P}')$. Similarly, applications could indicate to the planner that active flows shall be removed from the network. In this case, the planner also purges the corresponding candidate configurations from the conflict graph.

Optimization: Progressive Strategy for Offensive Planning

The planner employs locking and GFH in a two-phase meta-strategy for offensive planning:

In the first, defensive phase, for each flow in $Active\mathcal{F}(\mathcal{P})$ *every* configuration in $\text{cand}(f, \mathcal{P}') \setminus \text{config}(f, \mathcal{P})$ is locked in $\boldsymbol{G}_c(\mathcal{P}')$. That is, we conserve the configuration of all active flows. Then $\boldsymbol{G}_c(\mathcal{P}')$ exposes only one configuration per active flow and all candidates for new flows to the GFH. If we already find a traffic plan $\mathcal{P}'$ for $Active\mathcal{F}(\mathcal{P}) \cup Req\mathcal{F}(\mathcal{P})$, we are done and usually will have saved computation time, since the GFH considers fewer configurations.

Only if we cannot admit all new flows, we release the conservative locks (configurations that cause transition interference remain locked) and expose the "full" conflict graph to the GFH. This second phase widens the search space for the GFH at the expense of longer runtime, and active flows may now be reconfigured. If the GFH rejects any active flows in the second phase, we revert to the result from the first phase which is guaranteed to include all active flows in the new traffic plan $\mathcal{P}'$.

5.3.5 Installing the Plan

After having computed the new plan, the controller propagates the sub-plans to the nodes in the network. The sub-plan sent to a particular bridge defines how the nodes are supposed to route incoming packets, when to reserve transmissions windows for these packets and when the new plan becomes active, that is, the traffic plan's activation time t_{act}. In other words, after having computed the new plan, it is installed in two steps: Firstly, we send the sub-plans, flow-information sets, and the activation time for the new plan to all the nodes in the network while the old traffic plan is still active. Secondly, at the respective t_{act}, the nodes switch over to the new flow information sets. Such a time-based two-step update pattern is, for example, specified for TSN bridges in [IEE18b].

A transition to the new traffic plan shall only happen if all nodes have agreed that they have received the new sub-plans *and* will put their sub-plan into action at t_{act}. This means each traffic plan update requires to solve the well-known consensus problem with a deadline that equals t_{act} (minus the time nodes need to process update-protocol packets in the worst-case). However, it is well-known that is impossible to solve the consensus problem in asynchronous systems [Fis+85]. In particular, termination within bounded time cannot be guaranteed in such systems.

From a practical perspective, though, our specific network update problem has some properties which make this less of an issue:

Firstly, as soon as nodes or links fail, our network may not fulfill its primary function anymore because such errors can interrupt the service provided to already active flows. This means, before addressing the secondary problem of how to update a traffic plan in a network with failures, we have to decide and specify how to react to these failures with regard to the active networked applications themselves. For example, we could shut everything down in face of a failure and restart the network only after resetting it to a correct state. In this case, we do not update a failed network. For more involved strategies with more fault tolerance, we need not only address the consensus problem, but we also need additional concepts or countermeasures that have to be incorporated into the modeling and computation of the traffic planning, too. This could include built-in redundancy or fail-over on the networking level, as well as support for operation in a partially degraded network on a subset of unaffected candidate paths, which is out of the scope of this thesis.

Secondly, we can implement *static* real-time control channels between the controller and each network node. These channels could be established when the network is initialized. For example, we can even use the traffic planning approach described in this section to set up these channels—before processing any flow requests issued by "regular" applications—as pinned time-triggered flows such that their associated QoS is not subject to degradation. With real-time channels and bounded update-protocol processing times in the nodes, we effectively execute the network update in a synchronous system. Therefore, agreeing on the update within a deadline is possible. Instead of using time-triggered flows, we could also turn to real-time control channel implementations that occupy fewer network resources at the cost of increased deadlines, for example, non-time-triggered flows with bandwidth guarantees. We can safely handle longer deadlines of the real-time control channels since the controller can set the activation time to a hyper-cycle boundary in the future where the consensus protocol has terminated with the given real-time control channels and the worst-case update-packet processing duration in the nodes has elapsed. Obviously, the control-channel implementation choice involves a compromise between network resources spent on control channels and the time it takes to install an update in the network.

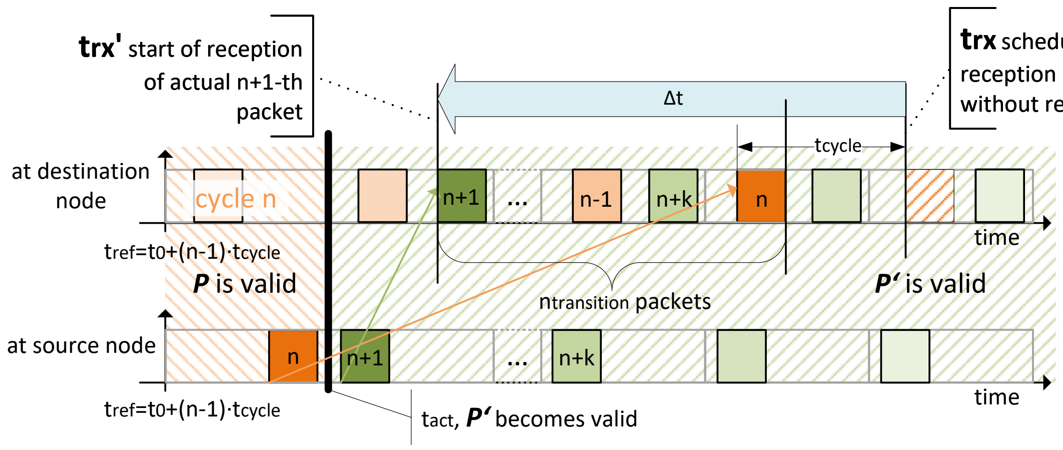


FIGURE 5.18 Reconfiguration of an active flow can temporarily cause jitter and packet reordering. Here, the packets with numbers $n+1$ to $n+k$ arrive "too early", that is, before packet n.

As discussed in Sec. 5.3.3, the controller has to postpone the activation time for *sources* of new flows after the transition interval whereas this is not necessary (and it would also be impractical) for bridges that forward packets of new flows. This means there is one activation time for the source nodes of flows from $Active\mathcal{F}(\mathcal{P})$ and the bridges. The activation time for the source nodes of new flows however is delayed by $\lceil \frac{t_{\text{transit}}}{t_{\text{cycle}}} \rceil \cdot t_{\text{cycle}}$ with t_{transit} the duration of the longest transition interval. While the flows from $Active\mathcal{F}(\mathcal{P})$ already use the new traffic plan, which has also been deployed to all bridges by that time, the source nodes of new flows have to wait a little longer before they are allowed to start sending packets.

5.3.6 QoS Considerations

The reconfiguration of active flows can degrade the QoS by introducing jitter. Since we aim for deterministic real-time communication, we must also quantify and possibly contain the QoS degradation caused by reconfigurations to a level acceptable for the applications. Next, we study how the reconfiguration of an active flow can degrade QoS. The level of degradation depends on the "distance", that is, the phase shift and the difference in the lengths of the routes, of the flow's old and new configuration. After showing how these properties affect QoS degradation, we present a way how applications can control the degree of degradation.

Computing QoS Degradation

Without reconfiguration, the destination node of an active flow receives the next packet every t_{cycle} seconds after the reception of the previous packet, and packets are received in the order they were sent from the source node. Now assume, a new traffic plan $\mathcal{P}'$ supersedes $\mathcal{P}$, and an active flow $f \in \mathit{Active}\mathcal{F}(\mathcal{P})$ is reconfigured, that is, $\text{config}(f, \mathcal{P}) \neq \text{config}(f, \mathcal{P}')$. Without loss of generality, let the activation time of the new traffic plan $\mathcal{P}'$ be the start of the $n+1$-th cycle of f.

Consequently, the point in time t'_{rx} when the destination node receives the $n+1$-th packet, which is sent with $\text{config}(f, \mathcal{P}')$, may differ from the point in time t_{rx} when the destination would have received the $n+1$-th packet without reconfiguration, that is, if the $n+1$-th packet would have been sent with the old $\text{config}(f, \mathcal{P})$, see Fig. 5.18. We call the magnitude of this deviation $\Delta t = t'_{\text{rx}} - t_{\text{rx}}$ the (reconfiguration) *jitter*.

THEOREM 5.10 (Packet Jitter Δt)
After a reconfiguration of flow f from $(f, \phi, \pi) = \text{config}(f, \mathcal{P})$ to $(f, \phi', \pi') = \text{config}(f, \mathcal{P}')$, the arrival of the new packets at the destination node deviates by

$$\Delta t = (\phi' - \phi) + (\text{len}(\pi') - \text{len}(\pi)) \cdot d_{hop} \tag{5.7}$$

from their respective scheduled arrival according to previous $\text{config}(f, \mathcal{P})$ *with* $\text{len}(*)$ *denoting the number of bridges on the candidate path with index* $*$.

This means Δt is composed of a term $(\phi' - \phi)$ expressing the phase-difference, and a term $(\text{len}(\pi') - \text{len}(\pi)) \cdot d_{\text{hop}}$ that expresses the difference between the traversal times of a packet on the old route and the new route, respectively.

PROOF W.l.o.g., we use the start n-th cycle as reference time t_{ref}. In the n-th cycle, the source node starts transmitting the n-th packet at $t_{\text{ref}} + \phi$. The reception of this packet by the destination node starts at $t_{\text{ref}} + \phi + t_{\text{src}} + t_{\text{trans}} + t_{\text{prop}} + \text{len}(\pi) \cdot d_{\text{hop}} + t_{\text{dst}}$, see Sec. 3.4. If $\mathcal{P}$ remained valid, the reception of the $n+1$-th packet then would start t_{cycle} later at $t_{\text{rx}} = t_{\text{ref}} + \phi + t_{\text{src}} + t_{\text{trans}} + t_{\text{prop}} + \text{len}(\pi) \cdot t_{\text{perhop}} + t_{\text{dst}} + t_{\text{cycle}}$.

However, the new plan $\mathcal{P}'$ and thus $\text{config}(f, \mathcal{P}')$ becomes valid at $t_{\text{ref}} + t_{\text{cycle}}$, which is the start of the $n+1$-th cycle. The reception of the $n+1$-th packet sent by the source node with $\text{config}(f, \mathcal{P}')$ starts at $t'_{\text{rx}} = t_{\text{ref}} + t_{\text{cycle}} + \phi' + t_{\text{src}} + t_{\text{trans}} + t_{\text{prop}} + \text{len}(\pi') \cdot d_{\text{hop}} + t_{\text{dst}}$ at the destination node. By inserting these values, we get

$$\Delta t = t'_{\text{rx}} - t_{\text{rx}} = (\phi' - \phi) + (\text{len}(\pi') - \text{len}(\pi)) \cdot d_{\text{hop}} \tag{5.8}$$

■

If $\Delta t > 0$, there is an additional delay between the last packet of $\langle f, \mathcal{P} \rangle$ and the first packet of $\langle f, \mathcal{P}' \rangle$. Intuitively, a reconfiguration that results in either a longer path or a phase increment (that is, $\phi' > \phi$), or both, results in $\Delta t > 0$.

If $\Delta t < 0$, the new packets from $\langle f, \mathcal{P}' \rangle$ arrive too early. This happens, if either the reconfiguration results in phase decrement ($\phi' < \phi$), or a shorter route $\ell(\pi') < \ell(\pi)$, or both of these. Potentially, this can lead to a situation where packets from $\langle f, \mathcal{P}' \rangle$ overtake the last packets from $\langle f, \mathcal{P} \rangle$ in the network if the relative end-to-end deadline is allowed to be greater than the cycle time.

We can quantify the number of the affected packets, which possibly arrive in a different order than the order in which they were sent and do not have an inter-arrival time t_{cycle}, with the following theorem.

THEOREM 5.11 (Number of Packets Affected by Reconfiguration)
The number of packets affected by a specific reconfiguration of flow f is

$$n_{transition} = \begin{cases} 1 & \text{if } \Delta t > 0 \text{ (new packets arrive “late”),} \\ \left\lfloor \frac{|\Delta t|}{t_{cycle}} \right\rfloor + \left\lceil \frac{|\Delta t|}{t_{cycle}} \right\rceil & \text{if } \Delta t \leq 0 \text{ (new packets arrive “early”)} \end{cases}. \tag{5.9}$$

PROOF W.l.o.g., we use the start of the reception of a (virtual) $n+1$-th packet sent with config $(f, \mathcal{P})$ as reference time point t_{ref}, see Fig. 5.18. The reception of the actual $n+1$-th packet, which is the first packet sent with the new config $(f, \mathcal{P}')$, starts at $t_{\text{ref}} + \Delta t$ at the destination node.

If the reconfiguration results in $\Delta t = 0$, for example, if the new configuration has the same ϕ and a route of the same length, there is no jitter visible at the destination node, hence $n_{\text{transition}} = 0$. Due to $t_{\text{cycle}} \geq t_{\text{trans}}$, a reconfiguration that results in $\Delta t > 0$ affects only a single packet.

If the $n+1$-th packet arrives too early (that is, $\Delta t < 0$), there are $\left\lfloor \frac{|\Delta t|}{t_{\text{cycle}}} \right\rfloor$ packets from $\langle f, \mathcal{P} \rangle$ in the time interval from $t_{\text{ref}} + \Delta t$ which are received immediately after the start of the reception of a new packet from $\langle f, \mathcal{P}' \rangle$ at the destination node.

Conversely, from time $t_{\text{ref}} + \Delta t$ on, there are $\left\lceil \frac{|\Delta t|}{t_{\text{cycle}}} \right\rceil$ packets of $\langle f, \mathcal{P}' \rangle$ which are received immediately after the start of the reception of an old packet from $\langle f, \mathcal{P} \rangle$. This means each of these new packets follows right after an old packet (with inter-arrival time $\neq t_{\text{cycle}}$) and therefore is counted into $n_{\text{transition}}$. ∎

It is easy to see that we can upper bound the number of packets that possibly arrive in a different order than the order in which they were sent, and do not have an inter-arrival time t_{cycle} by $n_{\text{transition}} \leq 2 \cdot \left\lceil \frac{|\Delta t|}{t_{\text{cycle}}} \right\rceil$. The term $2 \cdot \left\lceil \frac{|\Delta t|}{t_{\text{cycle}}} \right\rceil$ approximates the worst-case where old and new packets arrive interleavingly at the destination node until all old packets are delivered.

It is important to note that any individual packet that is sent by a source node will always arrive within the end-to-end deadline bounds. Reconfigurations only affect the relative inter-arrival times in a packet train.

Restricting QoS Degradation

Applications can provide the planner with QoS degradation constraints in the form of thresholds on $|\Delta t|$ and the number of affected packets. If the destination node has no facilities to handle packet reorderings, for example, has no buffer, then such constraints can be used to ensure that all packets are received in the sending order. The QoS degradation constraint results in the following condition for $\boldsymbol{G}_c\,(\mathcal{P}')$:

QoS condition For each $f \in \mathit{Active}\mathcal{F}\,(\mathcal{P})$, there exists no configuration $c \in \operatorname{cand}(f, \mathcal{P}')$, where the reconfiguration from $\operatorname{config}(f, \mathcal{P})$ to c exceeds the threshold for $|\Delta t|$ or $n_{\text{transition}}$.

Analog to Condition 1 for locking configurations (see Sec. 5.3.3), we can prevent reconfigurations that violate the QoS condition by *locking*. This can be achieved with minimal overhead since the planner anyway traverses $c \in \operatorname{cand}(f, \mathcal{P}')$ and only needs to check the QoS condition for each candidate c of f that is not locked already due to transition interference.

5.3.7 Evaluation

In this section, we evaluate dynamic traffic planning using a prototypical C++-implementation of the planner. The dynamics in the set of active flows over time add an dimension to the evaluation scenarios. Whereas previously the traffic planner computed a single traffic plan for each evalation scenario, here each evalation scenario consists of a *sequence* of planning rounds. In each round, the planner processes a set of flow requests which can include requests for new flows as well as requests for the removal of active flows. Besides the conflict graph adjustments and the computation of the new traffic plan $\mathcal{P}'$, the planner also validates the absence of configuration conflicts in the new traffic plan in each round.

Evaluation Scenarios and Parameters

Table 5.3 gives an overview of the evaluation scenario parameters which have been derived from typical industrial use cases [Ind19; IEE21a]. For each flow, we draw the values for t_{resv} and t_{cycle} uniformly at random from the respective set. The number of new flows to add $|\mathit{Req}\mathcal{F}\,(\mathcal{P})|$ and the number of flows to remove is either drawn from a truncated Poisson distribution (poiss.) with averages as stated in Tab. 5.3 or is deterministic (det.) for each planning round. We place/remove the flows in clusters in the network to simulate control units connected to multiple sensors and actuators, where all flows in a cluster start at or target one common “cluster” node. The cluster sizes correspond to commonly found configurations of I/O bays of industrial control units. For example, to add 25 new flows the request may contain three clusters of one, eight, and 16 flows, respectively, or any other combination that adds up to 25 flows. By default, 20% of the new flows are pinned permanently, and we limit Δt for each remaining active flow to

TABLE 5.3 Overview of evaluation scenario parameters.

Figure	$\lvert Active\mathcal{F}(\mathcal{P})\rvert$	$\lvert Req\mathcal{F}(\mathcal{P})\rvert$	flows to remove	n_{ub}	t_{cycle} [µs]	n_{path}	network
5.19	500	25 (poiss.)	25 (poiss.)	$\leq$ {50,100}	{250,500,1000,2000}	3	ring(64,3)
5.20A	800	50 (poiss.)	50 (poiss.)	$\leq$ {50,100}	{250,500,1000,2000}	3	ring(64,3)
5.21,5.22	250	25 (det.)	25 (det.)	$\leq$ 100	{200,250,500}	3	ring(64,3)
5.23,5.24A	500	50 (det.)	50 (det.)	$\leq$ 100	{250,500}	5	var., 49 nodes
5.23,5.24B	500	50 (det.)	50 (det.)	$\leq$ 100	{250,500}	5	var., 81 nodes

$t_{\mathrm{resv}} = t_{\mathrm{pkt}}$ (packet size)	$\in \{1\,\mu s, 3\,\mu s, 5\,\mu s, 12\,\mu s\}$, i.e., packets sizes: 125 B – 1500 B on links with 1 $\frac{\mathrm{Gbit}}{\mathrm{s}}$
flow clusters	$\in \{1, 2, 4, 8, 16, 32\}$ (constrained by $\lvert Req\mathcal{F}(\mathcal{P})\rvert$ and amount of flows to remove)
network parameters	processing delay $t_{\mathrm{proc}} = 2\,\mu s$, propagation delay $t_{\mathrm{prop}} = 1\,\mu s$

$t_{\mathrm{cycle}} - t_{\mathrm{resv}}$ such that packets arrive in order at the destination node. We omit to specify an explicit value for t_{e2e}. Instead, we use a k-shortest path algorithm which provides us with the candidate paths with the lowest end-to-end delays that are possible within the given topology.

For the evaluations of runtime and schedulability, we consider circular networks with n nodes where each node is connected to the next m nodes in both directions (denoted by ring(n,m)). With this network topology, we can expect similar behavior for the flow placements since each node in the network has the same degree, and there is an equal number of alternative paths between any pair of nodes. Then, we specifically investigate the effects of different network topologies.

In Tab. 5.3, column n_{ub} denotes the upper bound on how many new candidates per flow the planner may add at most to the existing candidates of that flow in $\boldsymbol{G}_c(\mathcal{P}')$. Likewise, n_{path} is the upper bound on the number of candidate paths per flow. Both, n_{ub} and n_{path}, can be considered parameters of our planner which result in different trade-offs with regard to runtime and schedulability. For example, for meshed networks, it is intuitive to expect better results with more candidate paths at the price of longer runtimes (see Sec. 4.4.5), but we expect a diminishing return with regard to the number of candidate paths due to shared sub-paths on the k-shortest paths. Therefore, we use 3-5 candidate paths per flow, which already yielded a very high schedulability in our evaluations (see Sec. 5.2.6).

Evaluation Results

Next, we present the evaluation results with a focus on "how fast can we obtain a new traffic plan" (runtime), "how good are the solutions" in terms of schedulability (comparison), and the effects of different network topologies. If not indicated otherwise, we used a desktop-grade computer with Intel i7-10700K (8 cores) and 16 GB RAM for the evaluations.

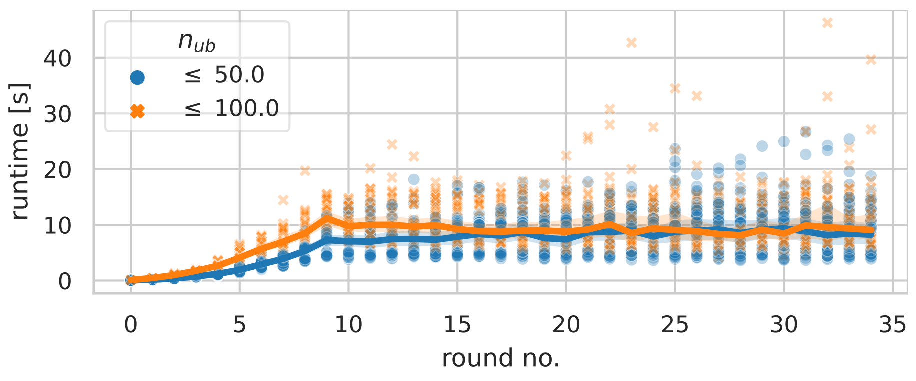


FIGURE 5.19 Total runtime per planning round (≈500 active flows).

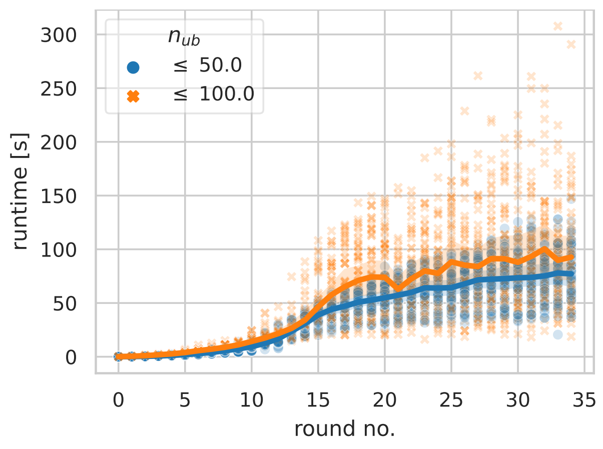
Runtime For each combination of average active flows (500, 800) and per-flow candidate set increment limit (n_{ub}: 50, 100), we evaluated 60 scenarios with regard to the runtime. Figures 5.19 and 5.20 show the results for 35 planning rounds.

We plot the *total* runtime for each planning round in Fig. 5.19 and Fig. 5.20A. The total runtime includes the time to compute the candidate paths for new flows, all the operations on the conflict graph (adding, locking, pinning, and removing candidate configurations), and the GFH runtime. For these scenarios, we measured up to 9.072 GB RAM usage by the planner.

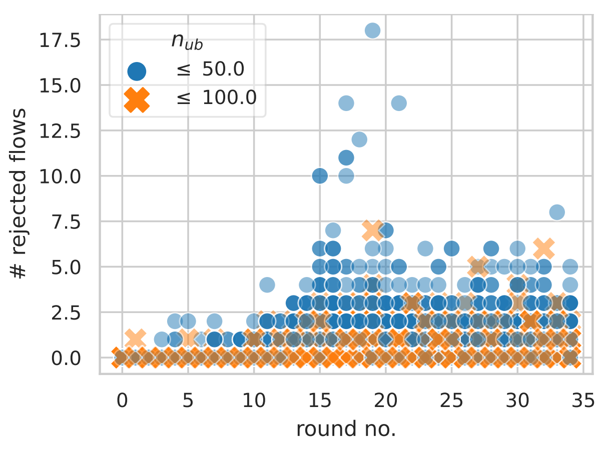
In the first 10 (16) rounds, we always issued requests to add 50 flows (no flow-removals) to initialize the conflict graph from scratch until $\mathit{Active}\mathcal{F}(\mathcal{P})$ contains ≈ 500 (800) flows. During these initialization rounds, we observe an increasing total runtime. After the initialization rounds, the planner receives requests to remove and add flows.

In the smaller scenarios with ≈500 flows (see Fig. 5.19), the runtime per round plateaus (avg: 8.24 s±3.58 s for n_{ub}=50; 9.2 s±4.03 s for n_{ub}=100) after the initialization rounds. In other words, here, it takes on average less than 10 s to exchange 25 active flows (and their configurations) against 25 new flows (and the respective new configurations) even though the planner can reconfigure ≈400 active flows. Here, we also have a quite “stable” conflict graph size in the post-initialization rounds which results in the comparatively constant runtimes per round.

In Fig. 5.20A, the runtimes for the bigger scenarios with ≈ 800 active flows are depicted. Here, the runtime continues to grow after the 16 initialization rounds (max: 146.83 s for n_{ub}=50; 307.72 s for n_{ub}=100). This behavior can be explained with Fig. 5.20B where we plot for each scenario how many new flows the planner rejected in each round. As discussed in Sec. 5.3.3, after the first pass over the ϕ-π-space the planner adds more candidates for *active flows* if flows from the previous request had to be rejected. This means we actually observe the desired expansion of the solution-space since we have to fit ≈300 more flows in the same network as for the smaller scenarios, which causes the planner to grow the conflict graph more aggressively—resulting in longer runtimes per round.



A Total runtime per round.



B Rejected new flows per round.

FIGURE 5.20 Runtimes and number of rejected flows per planning round for scenarios with ≈800 active flows.

The effects of varying n_{ub} can also be observed. If the planner can add up to 100 (additional) candidates per flow in a round, it can admit more new flows but processing times increase. In the scenarios with ≈800, flows the planner rejected on average 0.84 flows per request set after the initialization rounds for $n_{\mathrm{ub}} = 50$, whereas on average only 0.23 flows per round were rejected for $n_{\mathrm{ub}} = 100$.

Comparison Next, we compare defensive and offensive traffic planning with regard to schedulability and show that new algorithms such as the GFH are required to make offensive traffic planning feasible.

The evaluation scenarios vary with respect to how many active flows are on average permanently pinned to their configuration. We increased the ratio of pinned flows from 0% to 100% in 20%-increments and evaluated 40 scenarios for each round and pinning-setting. Here, there is no QoS restriction on Δt for each active flow that can be reconfigured.

In Fig. 5.21, we plot the cumulative number of flows rejected by the planner over 14 rounds. By pinning every active flow (labeled *w/o reconfiguration*, permanently pinning 100% of flows in Fig. 5.21), our approach performs defensive traffic planning, which results in a total of 62.5 rejected new flows on average. In comparison, if the planner performs offensive traffic planning, less than half as many flows—30.4 flows on average—are rejected (labeled *w/ reconfiguration*, permanently pinning 0% of flows in Fig. 5.21).

Since the GFH is a heuristic, the question arises of how "optimal" the GFH results are. To answer this, we compare the GFH against a drop-in integer linear programming implementation, which is given the same input as the GFH, namely, the conflict graph $\boldsymbol{G}_c(\mathcal{P}')$ and the flow sets

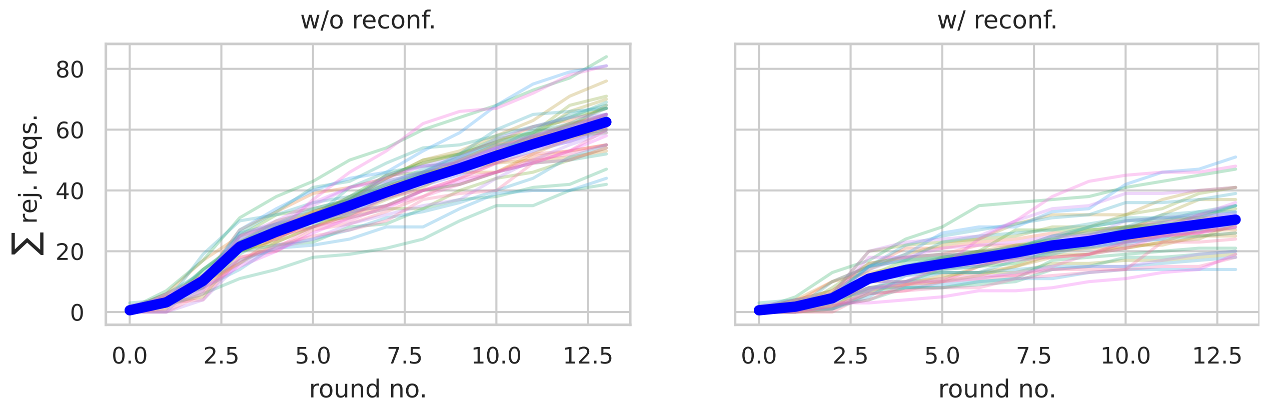


FIGURE 5.21 Defensive versus offensive planning: Cumulative rejected flows (thin lines: per scenario, bold line: average).

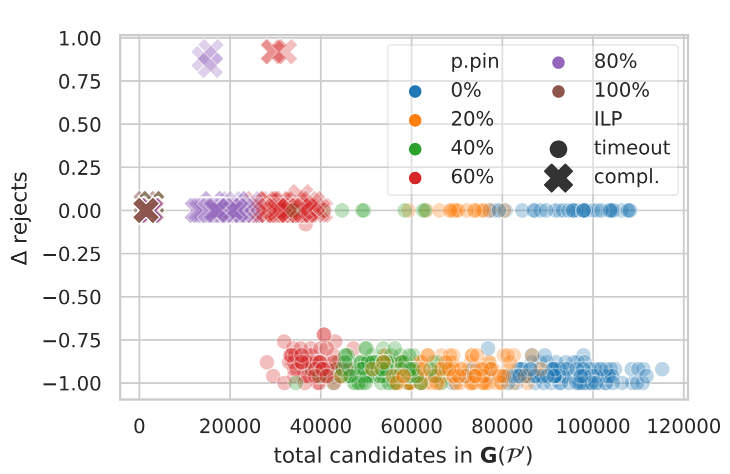
$\mathit{Active}\mathcal{F}(\mathcal{P})$ and $\mathit{Req}\mathcal{F}(\mathcal{P})$, and has to compute $\mathcal{P}'$ which optimizes Eq. (5.6). To make it clear, this ILP, which is based on the max-cover algorithm from Sec. 5.2.4, does not encode the whole traffic planning problem itself but requires the planner to encode the traffic planning constraints in $\boldsymbol{G}_c(\mathcal{P}')$.

For this comparison, we saved the conflict graph instances from the last four planning rounds from Fig. 5.21—presumably the largest conflict graphs in each scenario—to disk, and solved the corresponding ILP instances with a tool-chain using Julia [Bez+17], JuMP [Dun+17], and Gurobi v9.1.1 [Gur21]. The ILP solver had a runtime limit of 5 min (for comparison, the maximal GFH runtime was 8.3 s). Due to the higher memory requirements (up to 56.694 GB) of the ILP toolchain, we used a server-grade computer with two AMD EPYC 7401 processors (each 24 cores) and 256 GB RAM, but both, GFH and ILP solver, were limited to using max. 16 threads as in the other evaluations.

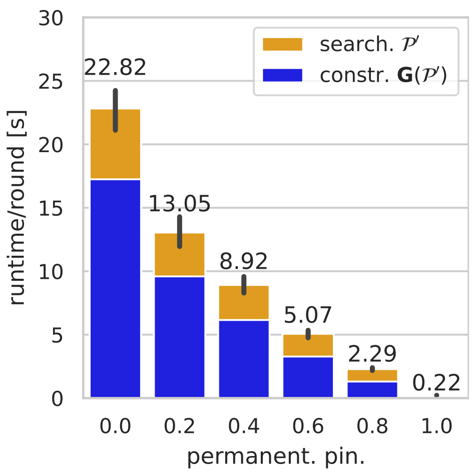
Figure 5.22A plots for each planning round the relative difference in the number of rejected new flows

$$\Delta\text{rejects} = \frac{\text{rejects GFH} - \text{rejects ILP}}{\text{new flows (total)}} \tag{5.10}$$

over the number of candidate configurations in the conflict graph. If the ILP solver rejected fewer flows, that is, computed a better result, we have $\Delta\text{rejects} > 0$ and vice-versa. In Fig. 5.22A, the ILP solver could provide better solutions for small conflict graphs, and, except for a few outliers where GFH reverted to the result from the first phase, the advantage of the ILP over GFH is small. Yet, once conflict graphs contain 30'000 or more candidate configurations, the ILP solver frequently hits the time limit (for 0% pinned flows: GFH avg=4.9 s±1.8 s; ILP avg=290.7 s±71.4 s) and would reject most new flows. This means even if we factor in the performance benefits of the conflict graph modeling itself (see Sec. 5.2), the GFH algorithm



A GFH vs. ILP: Relative difference wrt. rejected new flows for varying ratio of permanently-pinned active flows and 5 min runtime-limit.



B Runtimes per planning round for scenarios from Fig. 5.22A with GFH.

FIGURE 5.22 Comparison of schedulability and runtime for varying ratios of non-reconfigurable (permanently-pinned) active flows.

pushes the boundaries for offensive traffic planning further out compared to state-of-the-art exact approaches (in our case, integer linear programming) which are limited to either small scenarios or a small fraction of reconfigurable flows (pinning 80% of all flows).

For reference, we also depict the average runtimes of the planner using the GFH for each round of the scenarios from Fig. 5.22A for the different ratios of permanently pinned flows in Fig. 5.22B (error-bars indicating the variance of the total runtime per round). Figure 5.22B highlights the flexibility of our approach: if we "re-interpret" flow pinning as a tuning parameter, for example, used for probabilistic pinning of new flows by the planner, we can cover the full range between offensive planning with high schedulability and extremely fast defensive planning (see Fig. 5.22B: max. 250 ms for processing rounds with ≈250 active flows on the server-grade machine).

Network Topology To investigate the effects of network topology and network size, we generated scenarios for networks with 49 nodes and 81 nodes for different graph models, namely, Waxman, Price, ring(n, m), and Erdős-Rényi, using graph generators from [Hag+08; Pei14]. We evaluated 60 scenarios for each size-topology combination with 500 active flows on average. After the initialization, the planner tries adding 50 new flows and removing 50 active flows in each planning round.

The average runtimes after initialization and warm-up rounds are plotted in Fig. 5.23. In Fig. 5.24, we plot for all traffic planning rounds (including initialization and warm-up rounds)

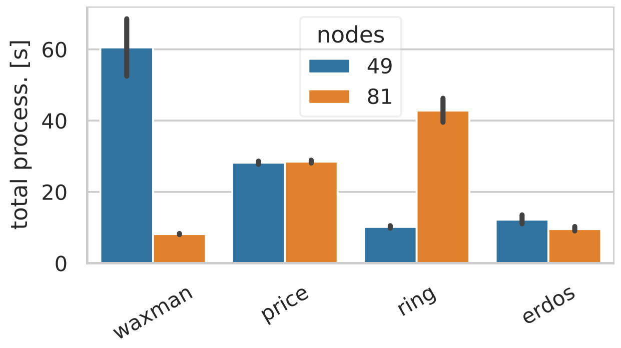
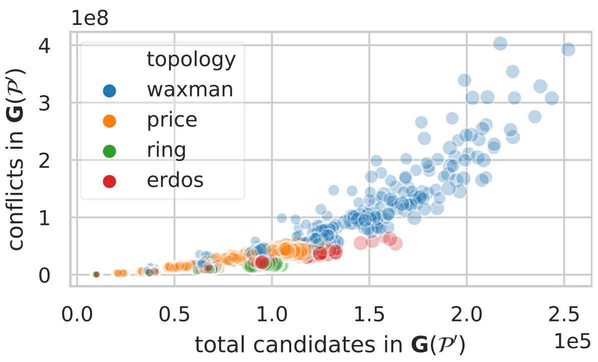
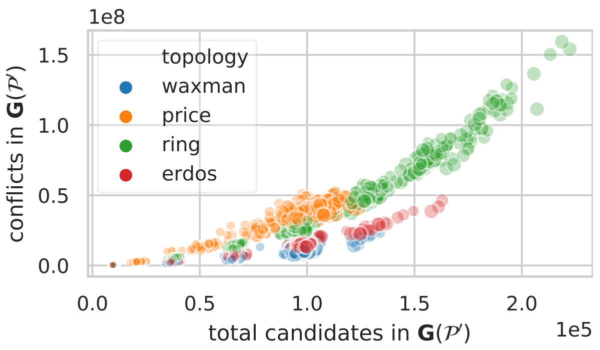


FIGURE 5.23
Average runtime per planning round for different network topologies.



A Conflict graph (49 nodes).



B Conflict graph (81 nodes).

FIGURE 5.24 Impact of different network topologies on conflict-graph size. (Note the different ranges of the axes.)

the number of conflicts (edges in $\boldsymbol{G}_c(\mathcal{P}'))$ over the number of configurations (vertices in $\boldsymbol{G}_c(\mathcal{P}'))$ of the conflict graphs constructed by the planner. We observe that a network with more nodes does not per se result in higher runtimes or bigger conflict graphs suggesting that our approach scales with the actual difficulty rather than just the network size.

Waxman networks with 49 nodes result in the largest conflict graph sizes overall and highest average (60.5 s) and maximal (283.72 s) total runtime per round. In Price networks with only one path between every two nodes, the planner cannot resolve conflicts via routing. Compared to the other topologies, on average more than ten times as many new flows were rejected per round (49 nodes: avg=10.4±5.12 rej./round; 81 nodes: avg=11.6±5.19 rej./round) in scenarios with Price network topologies. In scenarios with Price networks, GFH often required all re-runs, and we measured the third-highest total runtimes.

Evaluation Summary

Our evaluations show that the conflict graph approach can efficiently compute updated traffic plans for scenarios with hundreds of active flows using the GFH algorithm. Our modeling of the problem in combination with the GFH heuristic is efficient because we can reuse most of the state which encodes the solution space from the previous step. It also makes offensive planning computationally feasible compared to ILP-based approaches and thus allows to improve the quality of new traffic plans in terms of the number of rejected flows.

Therefore, our approach is a step towards true online-reconfiguration capabilities in terms of traffic planning. As discussed, true online-reconfiguration capabilities are not just a matter of traffic planning, but we also need to deploy the new traffic plans to the running network using a network control and configuration protocol stack with low processing latencies, too.

5.4 Related Work

In this section, we continue the discussion of our traffic planning contributions in the context of the related work, which we started in Sec. 4.5.

We organize this discussion of related work using new categories that also address the differences between our ILP-based approaches from the previous chapter and our conflict-graph-based approaches from this chapter. Namely, we discuss our approaches and the related work in terms of the traffic planning objectives, the dynamic aspects of traffic planning, and the traffic planning method.

Objective In this category, we are concerned with the actual traffic planning objective: Is traffic planning considered as a matter of pure constraint-satisfaction, that is, is it sufficient to compute any feasible plan for all flows? Or is traffic planning an optimization problem where we have to find the "best" possible traffic plan among the feasible traffic plans? And if yes, what is the objective? For example, is it merely a relaxation of the constraint-satisfaction problem (if you cannot find a feasible traffic plan for all flows, at least find a traffic plan for as many flows as possible) [Nay+16]?

Or is the objective added "on top" of the constraint-satisfaction problem? We can find many variants of the latter. For example, [Sch+17; Sch+20] formulate an objective to minimize the maximal end-to-end delay. In [Cra+16b], an objective to minimize the number of queues reserved for time-triggered flows is proposed, [Dür+16; Pah+19a] aim for short schedules, and [Oli+18] describes an objective for weighted per-flow jitter minimization.

Our ILP-based approaches from Sections 4.2 and 4.3 primarily fall into the category of constraint-satisfaction, but we have shown how to "add the optimization" of complemental flows (see Sec. 4.4) on top to obtain an optimized traffic plan for *all* flows.

On the other hand, the conflict-graph-based approaches more or less implicitly incorporate the objective to find traffic plans with as many flows as possible (for example, in the iterative CGTP-algorithm, Sec. 5.2). For dynamic traffic planning with conflict graphs (see Sec. 5.3) the objective Eq. (5.6) itself is a weighted maximization of the number of admitted flows—though practically, we consider a solution only acceptable if it admits all active flows.

Dynamics We can differentiate between traffic planning for static or dynamic scenarios. We addressed static planning in Sec. 4.2 – Sec. 5.2 and dynamic planning in Sec. 5.3. Out-of-the-box, most static planning approaches cannot be used for dynamic traffic planning, since—by nature—they do not account for old packets from a previous traffic plan which results in timing violations and/or packet reorderings during reconfigurations.

In principle, incremental approaches [Raa+17; Nay+18a] can be used for static scenarios and defensive planning in dynamic scenarios: "Freezing" the configurations of active flows ensures that "nothing bad" happens when new flows are added because the configurations of active flows from each previous step remain immutable for the future. However, defensive traffic planning prohibits adjustments of any past decision that may turn out to have been sub-optimal with respect to schedulability when facing new flows. This can result in low network utilization if a flow, which has been added early on, obstructs new flows.

This could be overcome by offensive planning that allows reconfiguring active flows. Reconfigurations of time-triggered flows are addressed in [Li+19; Pan+21]. An enhancement for two-phase network updates [Rei+12] with schedule-aware per-flow update times was presented in [Li+19]. The proposed update mechanism aims to reduce packet loss—caused by missed timings due to the reconfiguration—when transitioning from the current traffic plan to a given, new traffic plan. While [Li+19] considers traffic planning as an external step, [Pan+21] presents an ILP-based approach for the computation/modification of schedules which eliminates the possibility of missed timings altogether. Two algorithms are described in [Pan+21] that both allow packet-drop-free reconfigurations of active flows (offensive planning), by either computing the new schedule directly, or modifying an otherwise obtained, given schedule.

In contrast, our approach from Sec. 5.3 considers routing and scheduling (albeit for fixed topologies) when computing the new traffic plan. The existing approaches provide service guarantees during a reconfiguration either by prohibiting reconfigurations (defensive planning) or only in terms of packet drops. In contrast, our approach allows incorporating application-specific *per-flow* bounds on the (temporary) QoS degradation in terms of jitter and packet reorderings. That is, our approach enables offensive traffic planning for time-triggered flows with inhomogeneous QoS requirements, for instance, with regard to jitter-tolerances.

Method The method used to compute a traffic plan is often inseparable from the actual definition of the traffic planning problem. For example, once problem formulation or constraints cannot be expressed with linear expressions alone, integer-linear programming becomes a kludge at best. In any case, we can roughly identify two large "camps" with regard to the methods that are used to compute traffic plans.

On one hand, we have constraint-programming approaches, most prominently ILP and SMT/OMT (Optimization Modulo Theories). These constraint-programming approaches operate at the core by mapping the traffic planning constraints to constraints in the respective constraint-programming framework and then passing the constraint set to a general-purpose solver [Ste10; Nay+16; Poz+16; Cra+16a; Sch+17; Smi+17; Ste+18].

Later approaches enhance this scheme with additional pre-processing steps or heuristics to reduce the scope of the problem presented to the constraint-solver [Poz+16; Nay+18a; Nay+18b; Ata+19; Jin+19]. Examples of this are the aforementioned topology pruning or candidate path (pre-)computation. In some cases, this reduction will come at the cost of losing some solutions or optimality, and in many cases, this loss is not quantified. This applies, for example to iterative approaches which decompose the problem into sub-problems and then "freeze" the previously computed results.

The other "camp" does not rely on general-purpose constraint-programming frameworks or uses them only as fallback [Dür+16; Raa+17; Pah+19b; Sye+19]. Instead, the traffic plan is computed by a custom algorithm. Greedy algorithms are often used and in many cases provide great speed-ups, though this may be paid for by the (theoretic) sub-optimality of the discovered solutions. Similar to heuristics that build on constraint-programming approaches, a prematurely chosen, sub-optimal configuration for any individual flow can later on obstruct the placement of other flows.

Usually, these heuristics and most constraint-programming approaches operate on a level of abstraction which directly corresponds to the entities in the data plane such as individual links or packet timings. In contrast, we can interpret our conflict graph as some kind of transformation of the traffic-planning solution space which—via the notion of flow configurations—facilitates searching for feasible (partial) traffic plans.

Our conflict-graph-based approaches bridge the gap between constraint-programming and custom algorithms. With a complete conflict graph and a max-cover algorithm, it shares many similarities with constraint-programming approaches in that it will find a traffic plan if it exists. With an incomplete conflict graph that contains only a subset of candidate configurations for each flow paired with the adapted Luby's algorithm or the GFH, it is effectively a heuristic. With regard to the method, our approaches cover the whole range—from exact, constraint-based approaches to fast heuristics.

5.5 Discussion

In this chapter, we presented conflict-graph-based traffic planning approaches.

Viewed through the lens of the ILP-based approaches from Chap. 4), our conflict-graph modeling continues the idea of aggregating the traffic planning choices, which emerged in Sec. 4.3. In Sec. 4.3, the candidate paths served as a proxy for the fine-grained route computation constraints from Sec. 4.2 in the sense that a specific candidate path selection "toggled" the remaining link variables. A flow configuration can be interpreted as a replacement of all the individual scheduling and routing variables of a single flow. Aggregating all choices with regard to one possible "placement" of a flow reduces the interactions between traffic planning decisions for different flows to pairwise configuration conflicts under zero-queuing. This has the advantage that we need no knowledge about the global traffic plan when identifying these conflicts.

This does not only have the potential to speed up the solving process, for example, by parallelization, but allows to quickly obtain feasible solutions for a subset of the flows if the solving process is aborted prematurely. We also need not build a conflict graph that covers or considers the whole solution space, that is, all possible flow placements, right from the beginning. Instead, we can start with a small conflict graph, built from a small set of initial configurations, and continuously grow the graph until a solution is found. We also exploit this for dynamic scenarios, for instance, with dynamically changing flow sets, see Sec. 5.3, where we continuously adapt the conflict graph in response to the flow requests.

Another advantage of the conflict-graph-based approaches is the relative simplicity of the conflict graph itself. From an algorithmic view, it is intuitively easier to develop a custom heuristic to search independent colorful vertex sets in the conflict graph than it is to write an efficient algorithm that tries to find a valid variable assignment for several thousand variables in a constraint-based approach—which is supported by the performance of the GFH, see Sec. 5.3.7.

The ease of comprehension of the conflict-graph modeling facilitates enhancing the conflict-graph-based traffic planning to incorporate additional consideration into the traffic planning phase. In Sec. 5.3, for example, we added prioritization (active flows are more important). With locking and pinning, we introduced two mechanisms that can be combined with different rules for which configurations to lock/pin to implement other traffic-planning policies, too.

In summary, in conflict-graph-based approaches, we can manipulate the conflict graph and its construction to control which flow placements are up for selection in the first place, and we can encode our own strategies into algorithms to search for a traffic plan in the conflict graph, too.

6 Modeling Time-Triggered Service Intermittence with Network Calculus

In the previous part of the thesis, we have presented different methods for the synthesis of traffic plans for time-triggered flows where the traffic plans satisfy the desired end-to-end delay and jitter bounds by design. In particular, these traffic plans rely on the fact that packet emissions by traffic sources of time-triggered flows are synchronized to the reserved windows. In other words, the schedule for when a source node emits a packet and when gates of the time-aware shapers in the bridges open and close were computed jointly. In this case, it is not too difficult to reason about the service offered to the (time-triggered) traffic, and zero-queuing simplifies it even further.

But, for example, in converged networks, we may additionally have asynchronous traffic, for instance, shaped real-time traffic as known from IEEE 802.1 AVB or rate-constrained traffic in TTEthernet [Boy+16], which is also subject to real-time requirements. The bandwidth available to these flows on a particular link varies over time according to the schedule of the reserved windows for the time-triggered flows. In these scenarios, it is not so easy anymore to obtain guarantees for the non-time-triggered traffic, since all data flows, including the non-time-triggered, asynchronous flows, can be affected by time-aware shaping.

Generally speaking, bridges with time-aware shapers (see Chap. 3) are one example of network elements that intermit service according to a repeating schedule. Here, we refer to the forwarding and transmission of data as the service offered by the network elements (bridges). Service intermittence refers to the effect that there is some service for some intervals, for example, when the gate of the time-aware shaper is open, and no service for other intervals, for example, when the gate of the time-aware shaper is closed. Since service intermittence is controlled by a cyclic and deterministic schedule, we also speak of time-triggered service intermittence.

Time-triggered service intermittence in network elements is not always necessarily primarily motivated by real-time constraints. Schedules for service intermittence can also be incorporated into cyber-physical systems to increase their (power-)efficiency. Usually, we do not want to offer service when it is not required, not demanded, or when it is useless. For instance, if we can anticipate that there is negligible vehicular traffic outside rush hours, we could turn the communication link of a traffic sensor off for these time periods. More generally, this motivation can be found in applications where transmission schedules in the network elements are applied with the goal of power-conservation in energy-constraint devices [Pal+13; He+18; Bec+19], for instance, in wireless applications where data is buffered when the link is turned off. While for these wireless applications with time-triggered transmission intervals microsecond timings are not the primary objective, there are many scenarios that require bounds on the time of data delivery [Fat+15] and the buffer usage, for example, to avoid packet loss.

Taken to its conclusion, we get the very generic class of network elements that are controlled by a repeating schedule such that the schedule entries trigger intermittence or resumption of the service process at specific times. Compared to the first part of this thesis, we adopt a broader and more high-level view that abstracts different systems with time-triggered service intermittence.

Instead of pursuing a "constructive" task where we have to synthesize a traffic plan, we rather do the opposite in this chapter. We are given a system with time-triggered service intermission (and its configuration) and a traffic description, and we want to analyze the service and service guarantees offered in this scenario.

The service offered by these network elements does not only depend on the offered traffic load and the properties of the network (for example, link bandwidth, cyclic schedule) but additionally on the *current time*. Thus, these network elements, and by extension the whole network, exhibit a time-varying behavior.

Since it is non-trivial to give deterministic, provable real-time guarantees for the asynchronous traffic in these scenarios, formal frameworks are highly useful, because they provide a systematic way to prove guarantees on the end-to-end delay and other metrics such as the maximum queue length in network elements. Envelope-based approaches such as Network Calculus [Bou+01], which describe data flows with cumulative functions and network elements with service curves, are commonly used to analyze real-time systems. Thus, it is no surprise that Network Calculus has already been applied not only to analyze networks implementing non-time-triggered QoS mechanisms such as Weighted Fair Queuing of Integrated Services (IntServ) or TSN credit-based shaping as used, for instance, in multimedia communication [Bou98; Die+12; De +14] but also for time-triggered network elements with intermittent service (see Sec. 6.6). However, the research, often targeted at very specific technologies, is predominantly considering scenarios where the underlying service process ultimately offers constant-rate service, which means that the data is forwarded with constant rates during service intervals.

But what happens if you abandon the assumption of service processes with ultimately constant-rate service (for example, in case of wireless MIMO links where the number of active antennas [IEE+18; IEE21b] is changed)? And how bad is it, to use time-invariant functions to model time-variant systems?

In the following, we focus on the fundamental implications of modeling time-triggered network elements with service intermittence and provide multiple service curve formulations to facilitate the analysis in the Network Calculus framework.

6.1 Network Calculus Introduction

Deterministic Network Calculus [Bou96; Cru91] is a modeling framework to derive worst-case bounds on performance metrics such as backlog and delay. One of the core concepts of Network Calculus is the description of properties of the arrival processes by arrival curves and the description of queuing network elements by service curves. Arrival curves and service curves are cumulative, non-decreasing functions in

$$\mathcal{F}_{t.i.} = \{f : f(t) \geq f(\tau) \geq 0, \forall t \geq \tau \geq 0; \text{else } f(t) = 0\} \tag{6.1}$$

for time-invariant (t.i.) functions [Fid10], and respectively we have

$$\begin{aligned}\mathcal{F}_{t.v.} = \{ \; & f : f(s,t) \geq 0, \forall t \geq s; \text{else } f(s,t) = 0 \\ & \wedge \forall u \leq v \leq w : f(u,v) + f(v,w) = f(u,w) \; \}\end{aligned} \tag{6.2}$$

for time-variant (t.v.) functions [Bou14; Dan+13]. We consider a continuous time-domain with (s and) $t \in \mathbb{R}$.

Arrival curves indicate the accumulated amount of arriving traffic for a time interval. Conversely, service curves describe the accumulated amount of offered service for a time interval. In the following, arrival curves and service curves are left-continuous in each time variable.

Network Calculus defines operations such as convolution and deconvolution (using min-plus / max-plus algebra) on arrival and service curves and allows concatenating systems in a way similar to common system theory [Bou+01].

We derive both, time-variant service curves and time-invariant service curves for time-triggered network elements. The time-variant service curve can be expected to provide higher fidelity regarding the modeled system. However, this comes at a price: a) evaluating properties of a system with a time-variant description is more difficult (compare Equations (6.7) to (6.9) in Sec. 6.1.4), and b) important algebraic properties are lost with time-variant analysis compared to time-invariant analysis. Most prominently, the convolution operation is not commutative anymore, which would simplify the analysis of multi-hop scenarios.

Moreover, regarding the literature, time-invariant Network Calculus has attracted far more attention. This has the practical consequence that none of the major toolboxes and libraries for Network Calculus (DiscoDNC [Bon+14], CyNC [Sch+07], RTC [Wan+06b], RTaW Pegase [Rea], etc.) advertise support for time-variant formulations. The prevalence of time-invariant service curves in the literature and in the tooling is one more reason to investigate time-invariant service curves, too, and to compare them against time-variant service curves.

TABLE 6.1 Basic Network Calculus operations on $\mathcal{F}_{t.i.}$ / $\mathcal{F}_{t.v.}$.

operation	time-invariant $f, g \in \mathcal{F}_{t.i.}$
pointwise minimum $\oplus$	$f \oplus g(t) = \inf\,[f(t), g(t)]$
convolution $\otimes$	$f \otimes g(t) = \inf_{0\le\tau\le t}\,[f(t-\tau) + g(\tau)]$
left-deconvolution $\oslash$	$f \oslash g(t) = \sup_{\tau\ge 0}\,[f(t+\tau) - g(\tau)]^+$
operation	**time-variant $f, g \in \mathcal{F}_{t.v.}$**
pointwise minimum $\oplus$	$f \oplus g(s,t) = \inf\,[f(s,t), g(s,t)]$
convolution $\otimes$	$f \otimes g(s,t) = \inf_{s\le\tau\le t}\,[f(s,\tau) + g(\tau,t)]$
left-deconvolution $\oslash$	$f \oslash g(s,t) = \sup_{\tau\le s\le t}\,[f(\tau,t) - g(\tau,s)]^+$

Next, we provide formal definitions for arrival curves and service curves, and the Network Calculus operations.

6.1.1 Min-Plus Operations

Network Calculus uses the algebraic structure $\{\mathcal{F}, \oplus, \otimes\}$ [Cha+99; Bac+01; Bou+01; Dan+13]. The basic operations on functions in $\mathcal{F}$ used in this section (see Tab. 6.1, with $[x]^+ = \max(0, x)$) comprise the pointwise-minimum $\oplus$, the convolution $\otimes$, and the left-deconvolution $\oslash$ [Cha+99; Dan+13].

6.1.2 Arrival Curves

Assume that the cumulative arrivals at time t are given by $A(t)$. Then the arrivals are constrained by arrival curve α iff

$$\forall s \le t : A(t) - A(s) \le \begin{cases} \alpha(t-s) & \alpha \text{ is time-invariant} \\ \alpha(s,t) & \alpha \text{ is time-variant} \end{cases}, \tag{6.3}$$

or, alternatively, using the convolution

$$\begin{cases} A(t) \le A \otimes \alpha(t) & \text{time-invariant} \\ A_{t.v.}(s,t) = [A(t) - A(s)]^+ \le A \otimes \alpha(s,t) & \text{time-variant} \end{cases}. \tag{6.4}$$

6.1.3 Service Curves

Assume that the cumulative departures of a particular system are given by $R(t)$. Then that system offers a (simple) service curve β to the flow with arrivals $A(t)$ iff

$$\forall t \geq 0\ \exists s \leq t : R(t) - A(s) \geq \begin{cases} \beta(t-s) & \beta \text{ is time-invariant} \\ \beta(s,t) & \beta \text{ is time-variant} \end{cases}. \tag{6.5}$$

An important, more restrictive type of service curve is the so-called strict service curve [Bou+09; Bou+10]. A strict service curve has to satisfy

$$\forall \text{ backlogged intervals } (s,t] : R(t) \geq R(s) + \begin{cases} \beta(t-s) & \beta \text{ is time-invariant} \\ \beta(s,t) & \beta \text{ is time-variant} \end{cases}. \tag{6.6}$$

The strict service curve definition is related to the work-conserving property since it provides service guarantees over *every* continuously backlogged time interval, that is, intervals where continuously $R(t) < A(t)$. In contrast, a system with the more general, simple service curve might idle during such time intervals.

6.1.4 Deterministic Worst-Case Bounds

The goal of Network Calculus analysis is to calculate system properties such as worst-case bounds on delay and the amount of backlog. Arrival curve and service curve are required to this end (except for some special cases, such as the availability of minimum and maximum services curves [Bou+01]). Given arrival curve α and service curve β, the maximum backlog (the biggest amount of data being buffered in the system) [Fid10] can be computed by

$$\text{backlog}(\alpha,\beta) \leq \begin{cases} \alpha \oslash \beta(0) & \alpha,\beta \text{ time-invariant} \\ \sup_{t\geq 0}\left[\alpha \oslash \beta(t,t)\right] & \alpha,\beta \text{ time-variant} \end{cases}. \tag{6.7}$$

The maximum backlog can be thought of as the maximal vertical distance between the arrival curve constraining the arrivals in the system and the service curve of the system.

Equivalently, in the time-invariant case, the maximum virtual delay [Fid10] is given by

$$\begin{aligned} \text{v.delay}(\alpha,\beta) &\leq \inf\left[w \geq 0 : \alpha \oslash \beta(-w) \leq 0\right] \\ &= \inf\left[w \geq 0 : \sup_{\tau\geq 0}\left\{\alpha(\tau) - \beta(\tau+w)\right\} \leq 0\right], \end{aligned} \tag{6.8}$$

and in the time-variant case by

$$\begin{aligned} \text{v.delay}(\alpha,\beta) &\leq \sup_{t\geq 0}\left[\inf\left\{w\geq 0 : \alpha \oslash \beta(t+w,t) \leq 0\right\}\right] \\ &= \sup_{t\geq 0}\left[\inf\left\{w\geq 0 : \sup_{\tau\leq t+w}\left[\alpha\left(\tau,t\right)-\beta\left(\tau,t+w\right)\right]\leq 0\right\}\right]. \end{aligned} \tag{6.9}$$

The maximum virtual delay can be visually interpreted as the maximal horizontal distance between arrival curve and service curve, which is the maximal time needed for the value of the departures to reach the same value as the arrivals [De +14]. In case of FIFO, that is, data is serviced in the order of its arrival, the maximal virtual delay becomes the maximum waiting time for data in the system.

6.2 Network Elements: System Models

Next, we will introduce the system models for the two different time-triggered network elements with intermittent service (see Fig. 6.1). We will use them to highlight the fundamental principles and capabilities of the different service curve approaches in Sec. 6.3. In the remainder of this chapter, we use the fluid model where data can arrive and receive service in arbitrarily sized quantities. Therefore, different from the first part of this thesis (Chap. 2 – Chap. 5), packetization is ignored and data can arrive and receive service in arbitrarily sized quantities. We assume initially empty systems with no prior arrivals before $t = 0$ and no queue overflow.

We focus on a single network element and defer the discussion regarding multi-hop scenarios to Sec. 6.5. Nevertheless, the special case of a single network element can already be used to model an idealized communication bus or a network where every host is directly attached to one central bridge.

Data arriving at the network element is modeled by arrival process A with arbitrary, known arrival curve α. We consider arrival process A externally given, that is, we assume that we have no influence on arrival process A in the following. Data from arrival process A can be queued at the network element. We assume that enqueuing and dequeuing at the network element occur instantaneously. During times controlled by the time-triggered controller, service process S (defined by β_{S}) offers service to the arrivals of arrival process A. Cumulative departures at the network element are denoted by R.

The service curve β_{S} of the service process S (in isolation) is not to be confused with the service curve β of the complete network element. The service curve β describes the service offered to arrivals from arrival process A and depends on β_{S}, the type of time-triggered mechanism, and the schedule.

The time-triggered mechanism operates according to a cyclic schedule which contains entries

TABLE 6.2 Repeating schedule with cycle length t_{cycle}.

Interval i:	t_{enabling}	$t_{\text{disabling}}$
0	$t_{er,0}$	$t_{dr,0}$
1	$t_{er,1}$	$t_{dr,1}$
...		...

for the time instances when the controller enables the service process S to offer service to the arrivals of arrival process A (enabling time) and the corresponding time instances when the controller disables service process S from offering service to arrivals of arrival process A (disabling time). To this end, the time-triggered mechanism queries the current time t_{curr} from the clock of the network element. Before explaining the meaning of enabling and disabling for the two types of network elements, we first specify the parameters of the repeating schedule (see Tab. 6.2), with

$t_{er,i} :=$	ith enabling time of S *relative* to the start of the cycle,
$t_{dr,i} :=$	ith disabling time of S *relative* to the start of the cycle,
$t_{\text{cycle}} :=$	period of the schedule (cycle length).

The number of enabled intervals per cycle is given by $(i_{\max} + 1)$. The enabled intervals must not overlap, and we require $0 \leq t_{er,0} < t_{dr,0} < t_{er,1}$ and $t_{dr,i_{\max}} \leq t_{\text{cycle}}$. Service process S is enabled at t_{curr} if $\exists i \in [0, i_{\max}]$ such that

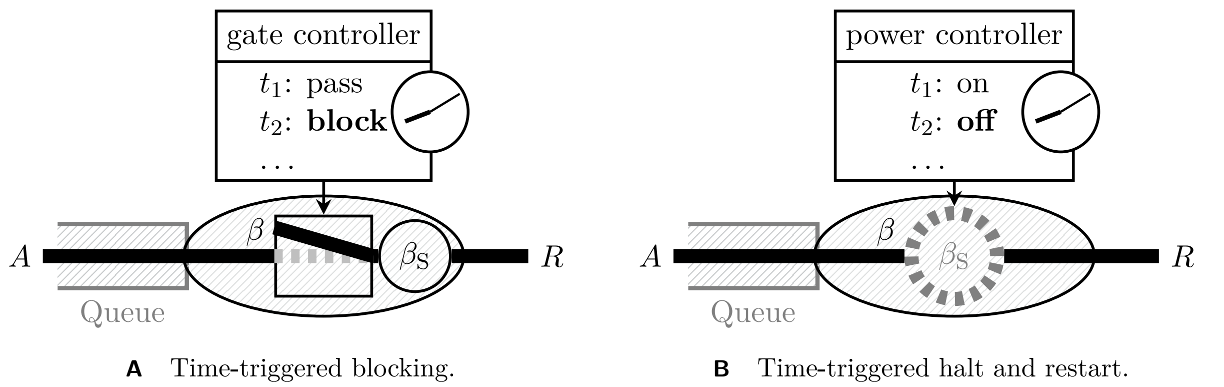
$$(t_{\text{curr}} \bmod t_{\text{cycle}} > t_{er,i}) \bigwedge (t_{\text{curr}} \bmod t_{\text{cycle}} \leq t_{dr,i}) \tag{6.10}$$

and disabled otherwise.

Regarding the time-triggered mechanisms, we consider two cases (see Fig. 6.1) where enabling and disabling of service process S are implemented differently.

6.2.1 Blocking

In the first case (Fig. 6.1A), there is an additional element—a gate—in between queue and service process S. The service process S has a strict, time-invariant service curve β_{S} and starts operating, that is, is powered on at time $t = 0$ and runs continuously. Starting at each enabling time, the gate controller opens the gate, which means that data in the queue is transparently passed through to the service process S until the next disabling time. At each disabling time, the gate controller closes the gate which means that the service process S is blocked from accessing the arrivals of arrival process A until the next enabling time. We will refer to this type of network element as a time-triggered network element with blocking. Note that the service process S is only disabled from the perspective of the arrival process A. While the



A Time-triggered blocking. **B** Time-triggered halt and restart.

FIGURE 6.1 Service curve β describes a composite network element comprised of a service process S with service curve β_S manipulated by a time-triggered mechanism.

service process is blocked for arrivals from A, the service process S could service data from other arrival processes, that is, $\beta_S(\tau)$ is the total amount of service the service process S has offered up to time $\tau = t$ of which only the fraction inside enabled intervals is available to arrivals from arrival process A, and t is the time in the domain of the network element. This behavior can, for example, occur in the bridges with time-aware shaping from Chap. 3, where gates restrict access of the different queues of an output port to the link.

6.2.2 Halt and Restart

In the second case (Fig. 6.1B), the service process S is manipulated directly by a so-called power controller. At each enabling time, the power controller (re-)starts the service process S. If the service process S is active, it offers the strict, time-invariant service curve β_S to the arrivals of arrival process A until the next disabling time when the power controller halts the service process S. If the service process S is disabled, it offers no service at all. We will refer to this type of network element as a time-triggered network element with halt-restart behavior. Here, β_S has only temporary meaning, because $\beta_S(\tau)$ is the amount of service the service process S can offer $\tau = t - t_{\text{enabling}}$ time units into an enabled interval. Thus, in case of halt-restart behavior, time instances where the service process S is restarted can be considered as renewal-points [Bec+15] of the service process S.

6.3 Deriving Service Curves

After presenting the basic Network Calculus background and introducing the system models in the previous sections, we derive time-variant service curves for time-triggered network elements with intermittent service in Sec. 6.3.1. The time-invariant service curves are subsequently introduced in Sec. 6.3.2.

In the remaining sections, we will use the following notation: the corresponding index in the cyclic schedule that belongs to the overall n-th enabled interval is given by $\varphi(n) = n \bmod (i_{\max}+1)$. For each enabled interval n, we introduce the symbols

$$\text{and} \qquad \left.\begin{aligned} t_{e,n} &= t_{er,\varphi(n)} \\ t_{d,n} &= t_{dr,\varphi(n)} \end{aligned}\right\} + \left\lfloor \frac{n}{i_{\max}+1} \right\rfloor \cdot t_{\text{cycle}}$$

where $t_{e,n}$ is the start, and $t_{d,n}$ is the end of the overall n-th enabled interval. In other words, $t_{er,i}, t_{dr,i}$ are relative to the beginning of a schedule cycle, whereas $t_{e,n}, t_{d,n}$ are absolute times.

6.3.1 Time-Variant Service Curve

The service offered to incoming data in a certain time interval depends on the position of that time interval on the time axis. This observation suggests using a time-variant formulation [Cha+99; Cha+02].

Depending on the type of time-triggered network element (see Sec. 6.2), the time-variant direct service curve is either given by the formula in Thm. 6.1 or Thm. 6.2.

THEOREM 6.1 (Time-Variant: Time-Triggered Blocking)
Let $\beta_S(t)$ be the time-invariant, strict service curve of the continuously running service process S. In a time-triggered queuing system where access to this service process S is blocked outside of enabled intervals according to a given repeating schedule, the time-variant service curve is given by

$$\beta(s,t) = \sum_{n=0}^{\infty} \left[\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))\right]^+ . \tag{6.11}$$

PROOF It holds that $\beta(s,t) \in \mathcal{F}_{t.v.}$, since $\beta(s,t)$ is the sum of $f(t), f \in \mathcal{F}_{t.i.}$. Let $A(t)$ and $R(t)$ be the arrival function and departure function, respectively. Considering an arbitrary backlogged interval $(s,t]$, we show that $R(t) - A(s) \geq \beta(s,t)$. We can distinguish the following cases, starting with those cases where the interval intersects with at most one gate open interval, before we cover the case where $(s,t]$ intersects with multiple gate open intervals of the flow of interest.

If no enabled interval is included in $(s,t]$ at all, then for all enabled intervals before $(s,t]$, we have $[\beta_{\text{S}}(t_{d,n}) - \beta_{\text{S}}(s)]^+$ and, conversely, for all enabled intervals after $(s,t]$, we have

$$[\beta_{\text{S}}(t) - \beta_{\text{S}}(t_{e,n})]^+ = [\beta_{\text{S}}(\min(t, t_{d,n})) - \beta_{\text{S}}(\max(s, t_{e,n}))]^+ = 0.$$

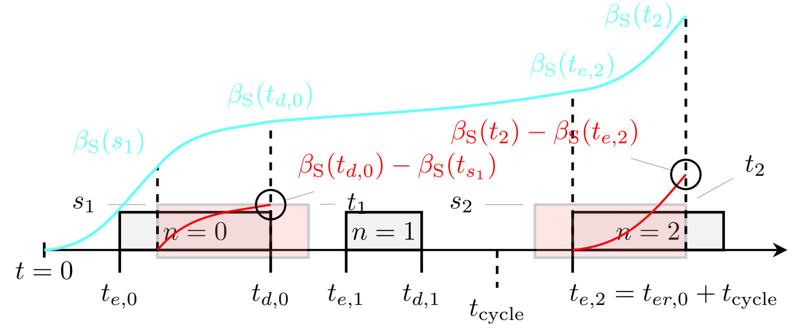


FIGURE 6.2
Start or end of enabled interval included in $(s, t]$.

Since for $(s,t]$ which do not intersect with any enabled interval $\min(t, t_{d,n}) \leq \max(s, t_{e,n})$ and $\beta_S \in \mathcal{F}_{t.i.}$, $R(t) - A(s) \geq 0$. Thus, enabled intervals not included in $(s,t]$ contribute nothing to $\beta(s,t)$.

We need $[\cdot]^+$ in Eq. (6.11), since it is possible that $\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n})) < 0$.

If the interval $(s,t]$ starts in the enabled interval, but the enabled interval ends before t (see enabled interval $n = 0$, $(s_1, t_1]$ in Fig. 6.2), we have

$$R(t) - A(s) \geq \beta_S(t_{d,n}) - \beta_S(s).$$

In case, the interval $(s,t]$ starts before the enabled interval but ends before the enabled interval ends again (see enabled interval $n = 2$, $(s_2, t_2]$ in Fig. 6.2), we have

$$R(t) - A(s) \geq \beta_S(t) - \beta_S(t_{e,n}).$$

If the enabled interval includes the interval $(s,t]$ (see enabled interval $n = 0$, $(s_1, t_1]$ in Fig. 6.3), we have

$$R(t) - A(s) \geq \beta_S(t) - \beta_S(s).$$

If the interval $(s,t]$ includes the enabled interval (see enabled interval $n = 2$, $(s_2, t_2]$ in Fig. 6.3), we have

$$R(t) - A(s) \geq \beta_S(t_{d,n}) - \beta_S(t_{e,n}).$$

If the interval $(s,t]$ intersects with multiple enabled intervals (see Fig. 6.4, where $(s,t]$ intersects with three enabled intervals), we define $n_s = \max\{n : t_{d,n} \leq s\} + 1$ the first enabled interval intersecting with $(s,t]$, likewise, the last intersecting interval is $n_t = \min\{n : t_{e,n} > t\} - 1$. According to the behavior of the time-triggered element with blocking, the value of R increases only when the service process S is enabled to offer service to the arrivals of A. For every

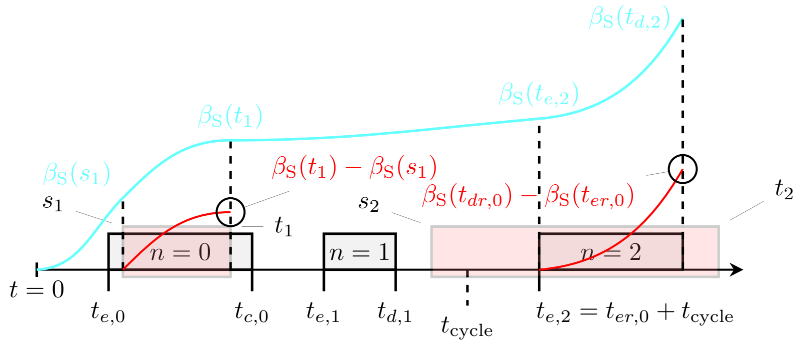


FIGURE 6.3 Enabled interval contains $(s, t]$ or vice versa.

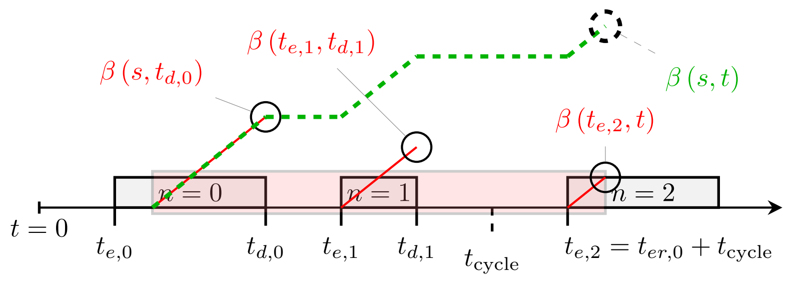


FIGURE 6.4 Offered service over multiple enabled intervals.

$n \in [n_s, n_t]$ the lower bound of the amount of departures (that is, arrivals which have received service from S) is given by $[\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))]^+$, and the lower bound of departures over the whole interval $(s, t]$ is then given by accumulating the amount of service

$$R(t) - A(s) \geq \sum_{n=n_s}^{n_t} [\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))]^+$$

in the individual enabled intervals in $[n_s, n_t]$. Considering that

$$\sum_{n=0}^{n_s-1} [\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))]^+ = 0,$$

$$\text{and} \sum_{n=n_t+1}^{\infty} [\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))]^+ = 0,$$

we can write

$$R(t) - A(s) \geq \sum_{n=0}^{\infty} [\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))]^+,$$

thus $R(t) - A(s) \geq \beta(s, t)$. ■

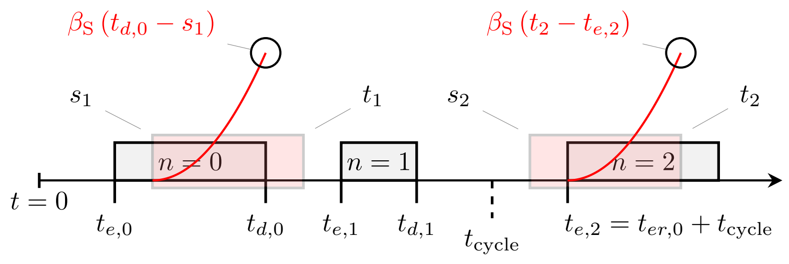


FIGURE 6.5
Start or end of enabled interval included in $(s,t]$.

THEOREM 6.2 (Time-Variant: Time-Triggered Halting and Restarting)
Let $\beta_S(t)$ be the time-invariant, strict service curve of a service process S. In a time-triggered queuing system where service process S is restarted at the beginning of each enabled interval and halted at the end of each enabled interval according to a given repeating schedule, the time-variant service curve is given by

$$\beta(s,t)=\sum_{n=0}^{\infty}\beta_S\left(\min\left(t,t_{d,n}\right)-\max\left(s,t_{e,n}\right)\right). \tag{6.12}$$

PROOF We prove Thm. 6.2 analogously to Thm. 6.1. Again, it holds that $\beta(s,t)\in\mathcal{F}_{t.v.}$. Considering an arbitrary backlogged interval $(s,t]$, we show that $R(t)-A(s)\geq\beta(s,t)$ with $A(t)$ and $R(t)$ the arrival function and departure function, respectively. We can distinguish the same cases as before:

If no enabled interval is included in $(s,t]$ at all, then for all enabled intervals before $(s,t]$, we have $t_{d,n}-s\leq 0$. For all enabled intervals after $(s,t]$, we have $t-t_{e,n}\leq 0$. Therefore,

$$R(t)-A(s)\geq\beta_S\left(\min\left(t,t_{d,n}\right)-\max\left(s,t_{e,n}\right)\right)=0,$$

since $\beta_S\in\mathcal{F}_{t.i.}$.

If the interval $(s,t]$ starts in the enabled interval, but the enabled interval ends before t (see enabled interval $n=0$, $(s_1,t_1]$ in Fig. 6.5), we have

$$R(t)-A(s)\geq\beta_S\left(t_{d,n}-s\right).$$

If the interval $(s,t]$ starts before the enabled interval but ends before the enabled interval ends again (see enabled interval $n=2$, $(s_2,t_2]$ in Fig. 6.5), we have

$$R(t)-A(s)\geq\beta_S\left(t-t_{e,n}\right).$$

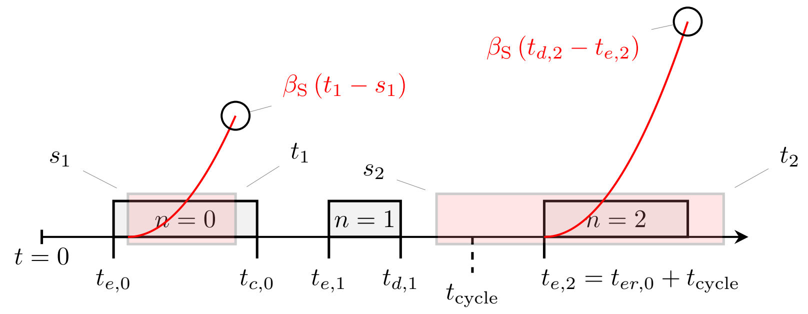


FIGURE 6.6
Enabled interval contains $(s,t]$ or vice versa.

If the enabled interval includes the interval $(s,t]$ (see enabled interval $n = 0$, $(s_1, t_1]$ in Fig. 6.6), we have

$$R(t) - A(s) \geq \beta_S(t - s) .$$

If the interval $(s,t]$ includes the enabled interval (see enabled interval $n = 2$, $(s_2, t_2]$ in Fig. 6.6), we have

$$R(t) - A(s) \geq \beta_S(t_{d,n} - t_{e,n}) = \beta_S\left(t_{dr,\varphi(n)} - t_{er,\varphi(n)}\right) .$$

If the interval $(s,t]$ intersects multiple enabled intervals (see Fig. 6.4, where $(s,t]$ (partially) covers three enabled intervals), the argumentation is analogous to the proof of Thm. 6.1. The main difference is that during each enabled interval n in $(s,t]$ the service process S of the network element with halt-restart can generate at least $\beta_S(\min(t, t_{d,n}) - \max(s, t_{e,n}))$ units of departing traffic. The lower bound of the total departures in any interval $(s,t]$ is the accumulated amount of the departures in the enabled intervals, hence

$$R(t) - A(s) \geq \sum_{n=0}^{\infty} \beta_S(\min(t, t_{d,n}) - \max(s, t_{e,n})) = \beta(s,t).$$

■

In Eq. (6.11) and Eq. (6.12), respectively, the summands

$$\begin{cases} [\beta_S(\min(t, t_{d,n})) - \beta_S(\max(s, t_{e,n}))]^+ , \\ \beta_S(\min(t, t_{d,n}) - \max(s, t_{e,n})) \end{cases}$$

describe the service that can be offered by server S with service curve β_S during the n-th enabled interval in the time interval $(s,t]$. For $\beta_S(t) = [C \cdot t]^+$ the two theorems yield the same service curve β, since $\beta_S(t-s) = \beta_S(t) - \beta_S(s)$. Therefore, we will use this choice of β_S in Figures 6.4 and 6.7 to illustrate the construction of $\beta(s,t)$ for $(s,t]$ overlapping more than one enabled interval and the time-variant property for both cases.

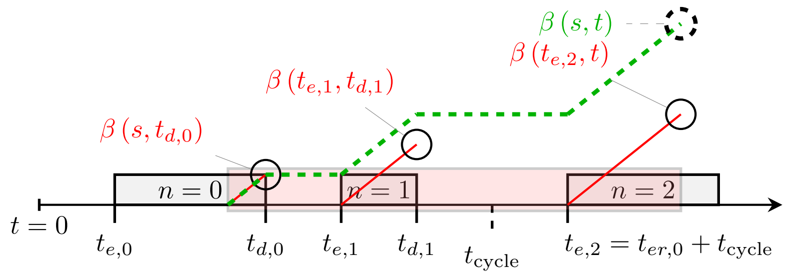


FIGURE 6.7 Offered service, if we shift the interval $(s,t]$ from Fig. 6.4 on the time-axis.

We get the dashed curve depicted in Fig. 6.4 by evaluating $\beta(s,t)$ with a fixed s and an increasing value of t in $s \leq t \leq s + \Delta t_{\text{range}}$. Shifting the whole interval $(s,t]$ from Fig. 6.4 on the time axis yields a differently shaped (not just shifted) service curve as shown in Fig. 6.7.

We can see that the static time-triggered scheduling results in a time-variant service curve, even if β_{S} is time-invariant since $\exists\delta$ for non-trivial schedules such that $\beta(s,t) \neq \beta(s+\delta, t+\delta)$. This means the service offered over intervals of equal length may be different.

We want to point out that in general, the service offered by time-triggered network elements with blocking differs massively from time-triggered network elements with halt-restart even if the service process S has the same service curve β_{S}. For example, consider a service process S with some kind of initialization "cost" such as some latency or lower service rate at $t=0$. For network elements with blocking, this initialization has to be accounted for *only once*, whereas for network elements with halt-restart, this initialization has to be accounted *for each enabled interval*. This necessitates the distinction between blocking behavior and halt-restart behavior in general.

6.3.2 Time-Invariant Service Curves

After having developed formulations for the time-variant service curve, we are going to provide two formulations for time-invariant service curves. Firstly, we use a leftover approach, and secondly, we use a direct approach.

Leftover Service Curve with Instant Arrivals

The key idea of the leftover service curve involves the subtraction of the amount of service that is offered to *another* arrival process A^v (which might be an aggregate of multiple arrival processes) from the total service offered to the arrival process A by the service process S. Thus, arrival process A can receive only that amount of service from S that is "left-over" by A^v (and hence the name).

The leftover service approach is well-known in Network Calculus and has been used in [Zha+17; Zha+18b] to model time-triggered blocking behavior. The particular leftover service curve

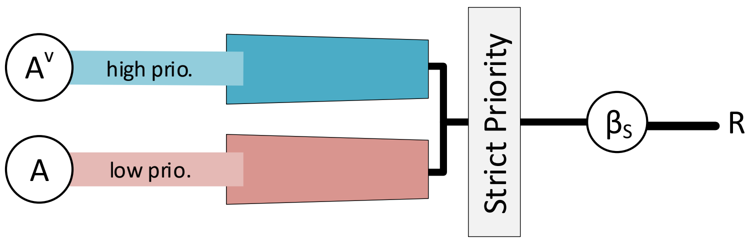


FIGURE 6.8 Artificial (virtual) arrival process A^v occupies service process S during disabled intervals of the actual arrival process A.

approach we present here follows the principles of the approach presented in [Zha+17; Zha+18b]. However, we will show that, in general, the leftover service curve approach is not applicable to time-triggered network elements with halt-restart (see Thm. 6.2).

For the leftover service curve approach, an additional (virtual) arrival process A^v is constructed, which is prioritized by the service process S over the arrival process A (see Fig. 6.8). The arrivals of the (virtual) arrival process A^v are scheduled according to the *inverse schedule* of the network element. This means at each disabling time in the schedule of the network element there are arrivals for A^v. The amount of arrivals of the virtual arrival process A^v is chosen to occupy the service process S (at least) until the corresponding disabling time for arrival process A^v, which is equivalent to the next enabling time in the schedule of the network element for the actual arrival process A. Thus, A^v effectively disables the service process S from servicing any data of arrival process A during the disabled intervals of the original schedule for arrivals from arrival process A—simply by keeping the service process busy servicing the virtual arrivals.

To construct A^v in accordance with our system model (see Sec. 6.2), we construct the arrival curve for the arrival process A^v from a set of intermediate arrival curves $a^v_{\text{imd},n}$. The intermediate arrival curve

$$a^v_{\text{imd},n}(t) = \sum_{k=n}^{\infty} \beta\left(t^v_{e,k}, t^v_{d,k}\right) \cdot \mathbb{1}_{\left\{t>t^v_{e,k}\right\}} \tag{6.13}$$

with $\mathbb{1}_{\{t>T\}} = \{1 : t > T; 0 : \text{otherwise}\}$ and

$$\beta\left(t^v_{e,k}, t^v_{d,k}\right) \text{ with } \left.\begin{matrix} t^v_{e,k} = t^v_{er,\varphi(k)} \\ t^v_{d,k} = t^v_{dr,\varphi(k)} \end{matrix}\right\} + \left\lfloor \frac{k}{i^v_{\max}+1} \right\rfloor \cdot t^v_{\text{cycle}} \tag{6.14}$$

models the instantaneous arrival of the maximum amount of data from arrival process A^v that can receive service during the k-th enabled interval of the arrival process A^v *in the inverse schedule*, immediately at the beginning of the k-th enabled interval of the arrival process A^v. The first enabled interval to be included in each intermediate arrival curve $a^v_{\text{imd},n}$ is the overall n-th interval in the *inverse schedule*.

TABLE 6.3 Asymptotic behaviors for time-invariant service functions with bounded $n^v_{\max}$.

const. rate [Bou14]	$\exists \sigma, \rho \in \mathbb{R}, \forall t : f(t) = \rho t + \sigma$ or $\forall t : f(t) = \infty$
ultimately const. rate [Bou14]	$\exists T, \exists \sigma, \rho \in \mathbb{R}, \forall t > T : f(t) = \rho t + \sigma$ or $\forall t > T : f(t) = \infty$
pseudo-periodic [Bou14]	$\exists c, \exists d \in \mathbb{R} \setminus 0, \forall t : f(t+d) = f(t) + c$
ult. pseudo-periodic [Bou14]	$\exists T, \exists c, \exists d \in \mathbb{R} \setminus 0, \forall t > T : f(t+d) = f(t) + c$
accelerating	$\forall a, d \in \mathbb{R}^+, \forall t : f(t+d) - f(t) \leq f(t+d+a) - f(t+a)$
ult. accelerating	$\exists T, \forall a, d \in \mathbb{R}^+, \forall t > T : f(t+d) - f(t) \leq f(t+d+a) - f(t+a)$

We construct the artificial arrival curve for arrival process A^v

$$\alpha^v(t) = \sup_{n \in [0, n^v_{\max}]} \left(a^v_{\mathrm{imd},n} \left(t + t^v_{e,n} \right) \right), \tag{6.15}$$

by taking the supremum of the shifted intermediate arrival curves. To account for the worst-case over all times when the service process S is disabled for the actual arrival process A, each intermediate arrival curve used in the construction of the arrival curve for the (virtual) arrival process A^v is shifted such that the first arrival of data of the respective intermediate arrival curve occurs at the time-origin.

This artificial arrival curve a^v thus emulates the effect of the service process S being not available during disabled intervals. The arrival curve a^v for A^v given in Eq. (6.15) is artificial in the sense that it is introduced only for modeling purposes and does not necessarily reflect the actual arrival curve of any other arrival process. In practice, there could be other traffic flows at time-triggered network elements with blocking which use other queues and actually receive service.

Note that if we do not know anything about the behavior of β_{S}, then $n^v_{\max} = \infty$. For $n^v_{\max} = \infty$, it is infeasible to evaluate Eq. (6.15) computationally. However, if β_{S} conforms to one of the asymptotic behaviors[1] given in Tab. 6.3, $n^v_{\max}$ can be bounded by

$$\left\lfloor \frac{n^v_{\max}}{i^v_{\max} + 1} \right\rfloor \cdot t^v_{\mathrm{cycle}} \geq T + t_{\mathrm{cycle}}. \tag{6.16}$$

Subtracting the arrival curve $\alpha^v(t)$ from the service curve of the service process β_{S} yields the leftover service curve

$$\beta(t) = \left[\sup_{s \leq t} \left(\beta_s(s) - \alpha^v(s) \right) \right]^+. \tag{6.17}$$

[1][Bou14] uses the term affine instead of constant-rate. In the related work, usually an affine or ultimately affine service function is chosen for β_{S} for which Eq. (6.17) can be used in practice since $n^v_{\max}$ is finite.

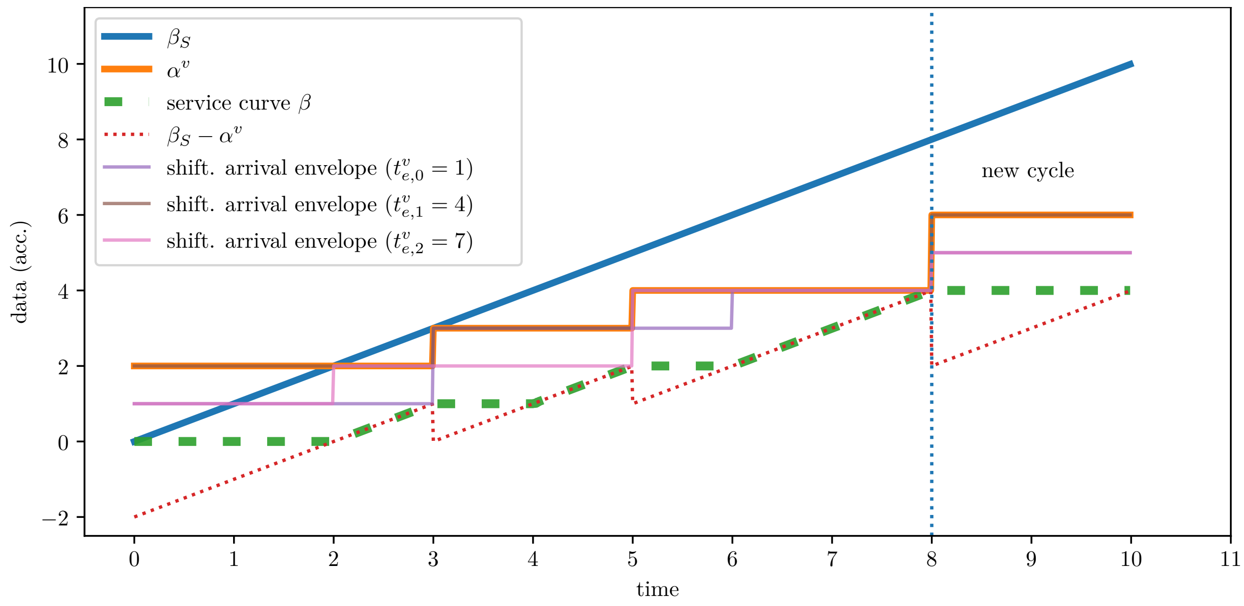


FIGURE 6.9 Leftover service for cyclic schedule in Tab. 6.4.

TABLE 6.4 Example schedule and corresponding inverse schedule for A^v with $i_{\max} = i^v_{\max} = 2$ and $t_{\text{cycle}} = t^v_{\text{cycle}} = 8$.

	Original Sched.		Inverse Sched. for A^v	
Interval i:	t_{enabling}	$t_{\text{disabling}}$	t^v_{enabling}	$t^v_{\text{disabling}}$
0	0	1	1	2
1	2	4	4	6
2	6	7	7	8

The $[\ldots]^+$ expression is necessary because due to the model with instantaneous arrivals, it can happen that $\beta_s(t) - \alpha^v(t) \notin \mathcal{F}_{t.i}$.

In Fig. 6.9, we plotted the shifted intermediate arrival curves, the artificial arrival curve, and the leftover service curve using the exemplary cyclic schedules from Tab. 6.4. In Fig. 6.9, the difference $\beta_s(t) - \alpha^v(t)$ (cf. the red dotted line) starts below zero and is not non-decreasing (cf. decreases at $t = 3$, $t = 5$, and $t = 8$) and therefore $\beta_s(t) - \alpha^v(t)$ is not a function in $\mathcal{F}_{t.i}$.

A proof for the leftover service curve has already been provided in [Zha+18b; Zha+17] and can be similarly applied to our modified arrival curve for the (virtual) arrival process A^v.

However, in the context of time-triggered network elements with intermittent service, the leftover service curve approach from Eq. (6.17) is in general not applicable to time-triggered network elements with halt-restart (see Thm. 6.2). Consider a scenario with $\beta_{\text{S}}(t) = \{t^2 :$

$t \geq 0, 0$ otherwise$\}$ and a schedule where disabled intervals and enabled intervals of length 1 alternate, starting with a disabled interval for arrival process A. Then, according to Eq. (6.14) and Thm. 6.2, we get $\beta\left(t^{v}_{e,k}, t^{v}_{d,k}\right) = 1$ per disabled interval of arrival process A. However, for instance, for $t = 3$ the arrival process A will have been offered at least $\beta(3) = \beta_{\mathrm{S}}(3) - 2 = 7$ units of service according to Eq. (6.17).

This is a contradiction. Even in the best case, arrival process A will have observed only two complete, enabled intervals in each of which it would have been offered 1 unit of service. While this choice of $\beta_{\mathrm{S}}(t)$ is quite contrived, with current wireless technology it is possible to vary the bandwidth per time interval in a controlled way, for instance, by adjusting the number of streams of MIMO links [IEE+18; IEE21b] or 5G resource blocks assigned to a connection.

Direct Service Curve

Here, we provide an alternative to the leftover service curve approach from the previous section that yields a valid time-invariant service curve for time-triggered network elements with halt-restart. For this approach, we again reuse the time-variant service curve, more precisely Eq. (6.11) and Eq. (6.12), respectively, to "directly" construct a time-invariant service curve.

The main distinction between a time-variant service curve and a time-invariant service curve is that the former describes the lower bound of offered service in any *specific* interval, whereas the latter describes the lower bound of offered service *irrespective of the position* of the interval. Therefore, the basic idea to derive the direct time-invariant service curve from the time-variant service curve is to get the overall lower bound of offered service by considering *all* intervals of the time-variant service curve.

THEOREM 6.3 (Time-Invariant: Time-Triggered Service Intermittence)
The time-invariant, direct service curve for a time-triggered network element is given by the infimum

$$\beta(t) = \inf_{n \in [0, n_{max}]} \left(\begin{cases} \beta\left(0, t\right) & n = 0 \\ \beta\left(\left(t_{\mathit{offset},n}\right), \left(t + t_{\mathit{offset},n}\right)\right) & n > 0 \end{cases} \right), \tag{6.18}$$

$$\textit{with} \qquad t_{\mathit{offset},n} = t_{dr,\varphi(n)-1} + \left\lfloor \frac{n-1}{i_{max}+1} \right\rfloor \cdot t_{cycle} \tag{6.19}$$

where $\beta(s,t)$ is the time-variant, direct service curve for the time-triggered network element with given repeating schedule and service process S with service curve β_S. For time-triggered network elements with halt-restart (see Eq. (6.12)), n_{max} can be set to $i_{max} + 1$, since in all cycles the same amount of service is being offered. For time-triggered network elements with blocking (see Eq. (6.11)), we have $n_{max} = \infty$ in general.

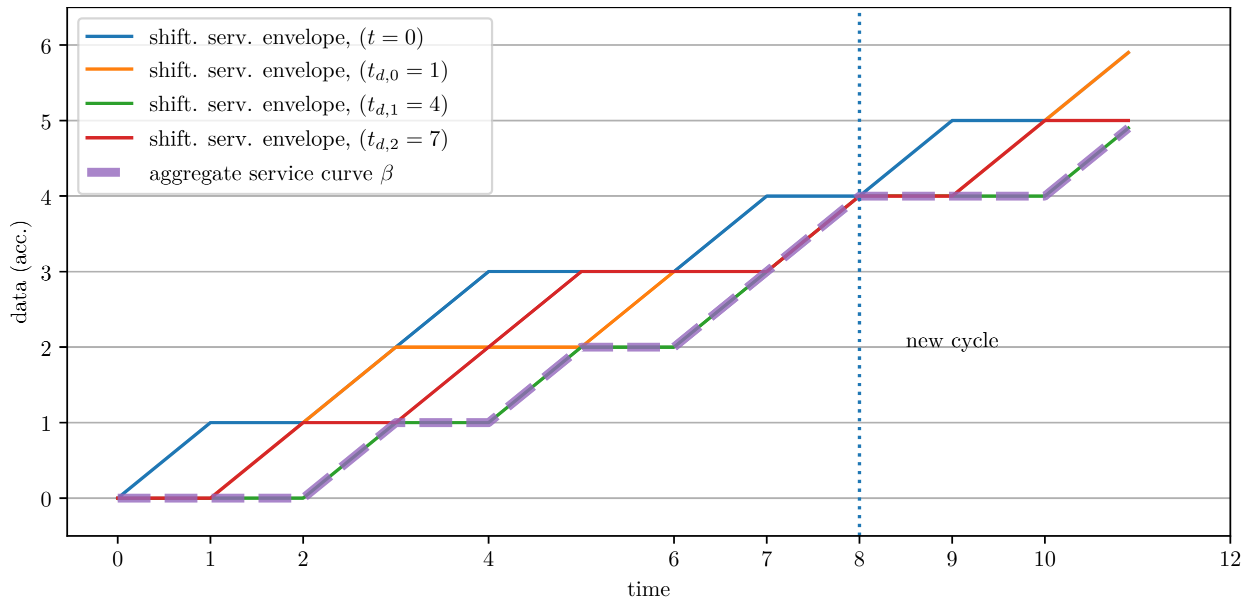


FIGURE 6.10 Direct, time-invariant service curve for the cyclic schedule from Tab. 6.4.

PROOF Recalling the definition of the service curves from Sec. 6.1.3, the infimum of offered service in any backlogged interval of length t equals the value of the time-invariant service curve at time t. Due to the cyclic properties of the time-triggered mechanisms, this lower bound can be found by evaluating the time-variant direct service curve $\beta(s, s+t)$ for $n_{\max}$ intervals with length t using properly chosen starting points s. If the starting point s of the interval with length t for which we evaluate the time-variant service curve $\beta(s, s+t)$ is equal to the time when an enabled interval has just ended, no service is offered until the next enabled interval. This is potentially a worst-case. As an intermediate step, we define the intermediate service curves

$$b_{\mathrm{imd},n}(t) = \begin{cases} \beta\left(0, t\right) & n = 0 \\ \beta\left(\left(t_{\mathrm{offset},n}\right), \left(t + t_{\mathrm{offset},n}\right)\right) & n > 0 \end{cases}. \tag{6.20}$$

The values of $b_{\mathrm{imd},n}(t)$ are obtained by evaluating $\beta(s,t)$ for a fixed s starting at the disabling times of the different enabled intervals n. To make the step from the intermediate service curves to the time-invariant service curve, each intermediate service curve $b_{\mathrm{imd},n}(t)$ is first shifted in the time domain (see Fig. 6.10) to the left by the distance of the last disabling time before $t_{e,n}$ from the origin. In the second step, we apply the infimum over all shifted intermediate service curves. Thus, given arrival function $A(t)$ and departure function $R(t)$, $R(t) \geq A(s) + \beta(t-s)$. ■

Note that for network elements where the service process S gets blocked according to the cyclic schedule (see Thm. 6.1) this service curve is again infeasible to evaluate computationally since $n_{\max} = \infty$. But for asymptotic behaviors of β_{S} as defined in Tab. 6.3, $n_{\max}$ can be bounded to a finite value similarly as in the leftover service curve approach, see Eq. (6.16).

The basic idea of taking the lower bound of several intermediate service curves can be recovered from [Zha+18a], which considers the special case where the intermediate service curves are derived from the traditional TDMA service curve with constant-rate service.

In Fig. 6.10, we illustrate the construction of the direct, time-invariant service curve for the schedule from Tab. 6.4. Fig. 6.10 already indicates that the cost for reducing the bivariate time-variant service curve function to a univariate time-invariant service curve is an under-approximation of the offered service due to the inf-term in Eq. (6.18).

6.4 Empirical Evaluation

Here, we investigate the difference between time-variant and time-invariant service curves with respect to the underestimation of offered service. We think this aspect is worth exploring because both—time-variant and time-invariant Network Calculus—have their own merits and drawbacks. Since the derivation of the time-invariant service curves (see Eq. (6.17)-Eq. (6.18)) involves the combinatorial problem of finding extrema of the respective intermediate curves, we use a numerical approach to assess the impact of the schedule on the difference between the time-variant and time-invariant service curves for the time-triggered network elements.

Since service processes with ultimately constant-rate service are used the most often, we use the most simple service curve $\beta_{\mathrm{S}} = t$ with constant-rate behavior for the service process S of the time-triggered network elements to figure out "how bad it is, to use time-invariant functions to model time-variant systems", focusing on the tightness of the worst-case bounds. Before continuing, we explain the performance metrics, which we use for the comparison of the service curve formulations.

6.4.1 Arrival-Curve-Oblivious Metrics

Since service curves can be interpreted as lower bounds on the actually offered service, the less under-approximation of offered service a service curve exhibits, the less worst-case bounds on backlog and delay are overestimated. Assume service curve β^a and service curve β^b are both supposed to model the same system. We consider service curve β^b more pessimistic than service curve β^a if β^a exceeds β^b for some time, and β^a never rises above β^b. More precisely, β^b is more pessimistic than β^a if

$$\begin{cases} \forall t : \beta^b(t) \leq \beta^a(t) \text{ and } \exists t : \beta^b(t) < \beta^a(t) & \text{(time-invariant)} \\ \forall s,t : \beta^b(s,t) \leq \beta^a(s,t) \text{ and } \exists s,t : \beta^b(s,t) < \beta^a(s,t) & \text{(time-variant)} \end{cases} . \tag{6.21}$$

The service curve can be regarded as means to describe the system, whereas the arrival curve

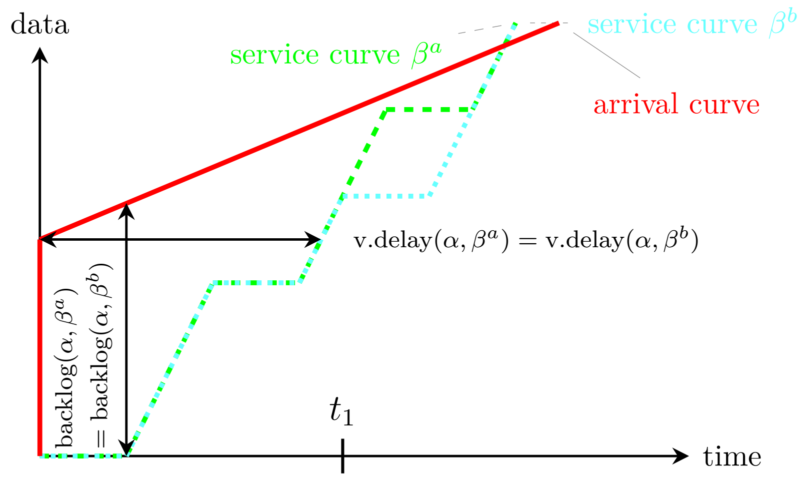


FIGURE 6.11 Different values of offered service resulting from service curve β^a and service curve β^b starting at time t_1 have no impact on worst-case backlog and delay bounds.

is foremost a description of the arriving data traffic. The definition of the arrival curve is tied to a specific application and network scenario and is not in our scope here. However, the shape of the arrival curve determines how much the different ways of modeling the service impact the computed delay and backlog bounds:

For the given arrival curve in Fig. 6.11, the maximum delay and backlog are the same for both service curves even though service curve β^b under-approximates the offered service starting at time t_1 more than β^a.

Figure 6.12 depicts an example where v.delay(α, β^b) − v.delay$(\alpha, \beta^a) = \Delta h$. In this scenario, the largest horizontal distance between the two service curves is the difference in the derived worst-case delay and can be computed with the equations for the virtual delay (Equations (6.8) and (6.9)), replacing the actual arrival curve α with the less pessimistic service curve β^a, and β with the more pessimistic service curve β^b, that is,

$$\Delta h = \text{v.delay}(\beta^a, \beta^b). \tag{6.22}$$

If, for example, a networked application requires that the worst-case (virtual) delay is less than a certain value (indicated by the dashed arrow in Fig. 6.12), we might come to the conclusion that this requirement is not met if we use the more pessimistic modeling that results in β^b, whereas the less pessimistic modeling confirms that the worst-case (virtual) delay is actually less than what is maximally allowed for this application. This means, here, the more pessimistic modeling might result in unnecessary modifications of the system to address this apparent delay requirement violation.

Analogously, we have backlog(α, β^b) − backlog$(\alpha, \beta^a) = \Delta v$ in Fig. 6.13. If we use β^b instead of β^a, then we overestimate the worst-case backlog of the system by Δv. In this case we compute

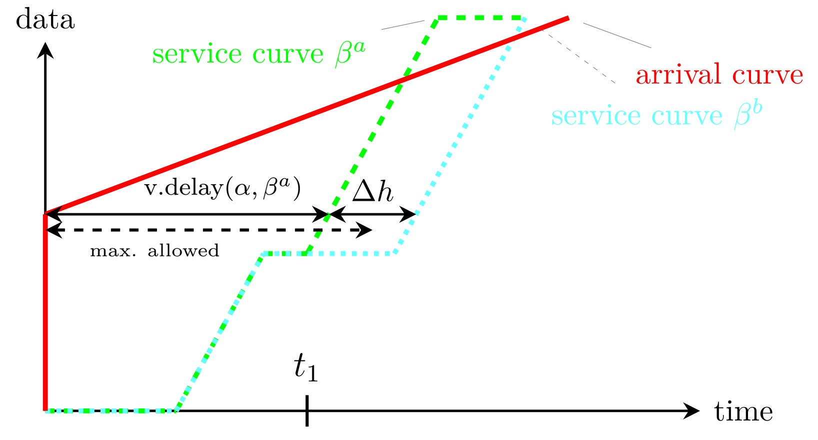


FIGURE 6.12 Different values of offered service resulting from service curve β^a and service curve β^b starting at time t_1 fully affect the worst-case virtual delay bounds.

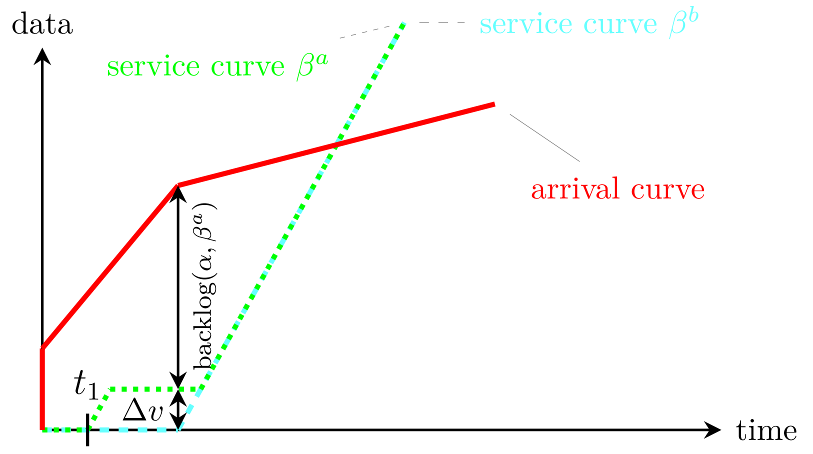


FIGURE 6.13 Different values of offered service resulting from service curve β^a and service curve β^b starting at time t_1 affect worst-case backlog bounds.

the difference in the derived worst-case backlog with the backlog equation (Eq. (6.7)), replacing α with the less pessimistic service β^a curve and β with the more pessimistic service curve β^b, that is,

$$\Delta v = \text{backlog}(\beta^a, \beta^b). \tag{6.23}$$

Similarly, depending on the arrival curve there exist scenarios where the maximum delay and backlog are equal for both service curves even though service curve β^b is more pessimistic than service curve β^a. Using Δh and Δv, we can express the possible difference of the delay and backlog bounds of two service curves in absence of any specific arrival curve. Therefore, our evaluation is not subject to a particularly "good" or "bad" choice of the arrival curve or limited to a specific scenario.

6.4.2 Results

With the Δh and Δv metric, we compare the following two combinations of pairs of service curves that can be mapped to the two types of network elements:

Time-triggered blocking We compare the time-variant service curve from Eq. (6.11) to the time-invariant leftover service curve from Eq. (6.17).

Time-triggered halt-restart We compare the time-variant service curve from Eq. (6.12) to the time-invariant direct service curve from Eq. (6.18).

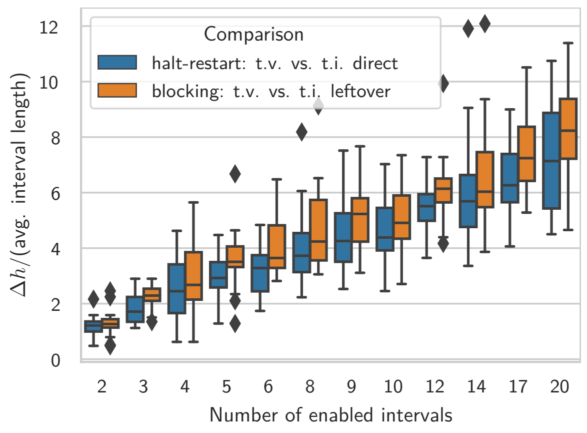
Equation (6.18) can also be used for time-triggered network elements with blocking [Zha+18a]. Nevertheless, we use the network element types to label the combinations throughout this section. To compare a time-variant service curve with a time-invariant service curve, we artificially extend each time-invariant service curve to a time-variant function $\beta_{\text{pseudo t.v.}}(s,t) = \beta_{\text{t.i}}(t-s)$. The evaluation of Δh and Δv is done as described previously with the time-variant versions of the equations for backlog bound (Eq. (6.7)) and virtual delay bound (Eq. (6.9)). Considering the derivation of the time-invariant service curve formulations for both types of network elements, the time-invariant service curves are either equally or more pessimistic than the time-variant service curves (see Figures 6.11 to 6.13: β^a equals the time-variant service curve, β^b equals the time-invariant service curve).

We used our own prototype implementation of Network Calculus operations in Python for the evaluations in this section. For the computation of Δh and Δv, we apply a time discretization with 100 steps per time unit and evaluate the service curve functions with randomly created schedules with a granularity of one time unit. The service curves are evaluated over a window of length t_{cycle} since the offered service per cycle is equal for all service curves for our choice of $\beta_{\text{S}} = t$.

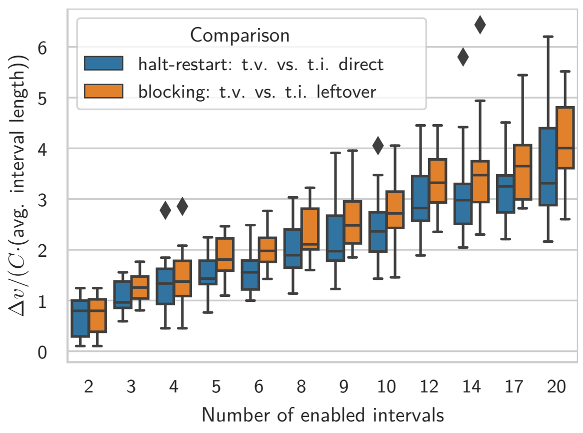
Schedules with Varying Number of Enabled Intervals of Random Length

The schedules for the evaluations presented in Figures 6.14A and 6.14B are created by randomly drawing the length of the enabled intervals and disabled intervals from the discrete range $[1, 100]$ with uniform probability. The cycle time is set to the sum of the lengths of the enabled intervals and disabled intervals.

The number of enabled intervals ranges from 2 to 20, and we evaluate 20 unique schedules for each number of enabled intervals and each type of network element. Due to the random length of the individual enabled intervals, it is not meaningful to interpret the absolute values of Δh and Δv. Therefore, we normalize Δh to the average interval length per schedule. Analogously, we normalize Δv to the amount of data that can be served in the average schedule interval. This normalization is applied per schedule.



A Δh relative to average schedule interval length: different $i_{\max}$.



B Δv relative to data transmittable in the average schedule interval: different $i_{\max}$.

FIGURE 6.14 Evaluation for schedules with enabled intervals and disabled intervals randomly drawn from $[1, 100]$.

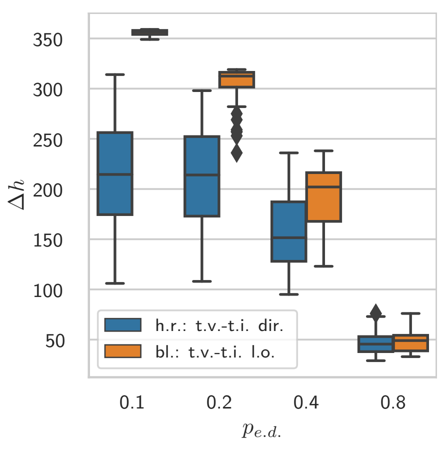
In both figures (Figures 6.14A and 6.14B) we can observe that overall the difference—the normalized values of Δh and Δv—between the time-variant and time-invariant service curve increases with the number of entries of the cyclic schedule. For the evaluated schedules, Δh is on average bigger than the size of an average interval. The outliers in Fig. 6.14A where Δh exceeds the average interval multiple times are from schedules where the enabled intervals are shorter than the difference of the lengths of the disabled intervals.

Thus, the time-invariant service curves can possibly lead to over-estimating the worst-case delay in the order of multiple interval lengths. The different modeling approaches have a larger impact on Δh than on Δv because here all service curves have the same slope in the enabled intervals, which is determined by β_{S}.

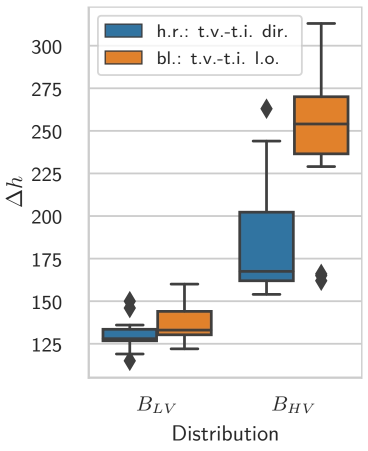
Schedules with Varying Ratio of Enabled Intervals to Cycle Time

While the evaluations in the previous section used random schedules where only the number of schedule entries is fixed, we now fix the percentage $p_{e.d.}$ of the cycle time when the service process is enabled. To be exact, we create 20 unique schedules per $p_{e.d.} = \sum_i (t_{dr,i} - t_{er,i})/t_{\mathrm{cycle}}$ with $p_{e.d.} \in \{0.1, 0.2, 0.4, 0.8\}$, each with $t_{\mathrm{cycle}} = 400$ and 10 enabled intervals.

The results are presented in Fig. 6.15A and show that the mean of Δh reduces with increasing $p_{e.d.}$. This means the less time the enabled intervals take up per cycle, the bigger the benefit of employing the time-variant service curve instead of the time-invariant service curve. The behavior shown in Fig. 6.15A supports the observation from the evaluation with random interval length in the previous section that the time-invariant approaches perform worse if the schedule consists of long disabled intervals with short interspersed enabled intervals such as $p_{e.d.} = 0.1$.



A Δh for schedules with different per-cycle enabled interval ratio.



B Δh for schedules with different interval-length variance.

FIGURE 6.15 Evaluation for random schedules with different creation schemes.

Schedules with Equal Mean Interval Length and Varying Variance of the Interval Lengths

To evaluate the effect of the variance of the enabled intervals, we compute Δh for schedules where all interval lengths are drawn from one of the two binomial distributions, $B_{HV}(N = 1000, p = 0.1)$, or $B_{LV}(N = 125, p = 0.8)$. The parameters are chosen such that the interval lengths have equal mean value $\mu_{HV} = \mu_{LV} = 100$ but different variance $\sigma^2_{HV} = 90$ and $\sigma^2_{LV} = 20$. Effectively, B_{HV} yields schedules with 4.5 times higher variance of the interval lengths compared to B_{LV}. Each schedule comprises 20 enabled intervals for each gate, and we evaluate 16 schedules per configuration.

The results are shown in Fig. 6.15B. The values of Δh are much higher for the data sets with higher variance of the interval lengths. Therefore, we conclude that not only the length of the schedule intervals but also the variance of the schedule intervals influences the differences between the service curves.

6.4.3 Summary of the Evaluation

In all of our evaluations, the time-invariant service curve approaches can lead to vast overestimations of the worst-case delay bounds and the worst-case backlog bounds. The more entries the cyclic schedule contains and the more the interval lengths vary, the stronger can the difference with regards to the worst-case bounds emerge in the analysis of a system. This can be attributed to the construction of the time-invariant service curves where a large variance of the schedule intervals can increase the under-approximation of offered service (see Figures 6.9 and 6.10).

This is intuitive to see if we look at the extreme case where all schedule intervals have the same length. Then the variance of the lengths of the schedule intervals is zero, and the schedule can be reduced to a schedule with just one enabled interval and a cycle length of two enabled intervals. In this case, the difference between the time-variant and the time-invariant service curve is limited to one interval length which is much smaller than the values observed in Figures 6.14 and 6.15.

In the evaluations, the overall difference between the time-variant service curve and the time-invariant service curve for the time-triggered elements with blocking is larger than the difference between the time-variant service curve and the time-invariant service curve for the time-triggered elements with halt-restart. This is caused by the modeling with instantaneous arrivals of the data (see Eq. (6.13)) that emulate the disabled intervals for the actual arrival process A in the construction of the time-invariant leftover service curve. The instantaneous arrivals can result in service being attributed to the virtual arrival process A^v too early. Due to our particular choice of $\beta_S = t$, which is less affected by the cyclic schedule compared to other service curves (for example, rate-latency, where latency $\geq$ interval length), we expect these effects to be even more pronounced for most other service processes.

6.5 Extension to Multi-Hop?

"In theory", Network Calculus already provides the means to evaluate multi-hop scenarios, because we can compute the service curve of a series of network elements with $\beta_1 \otimes \beta_2 \otimes \ldots$ (β_x being the single-hop service curve of the x-th network element). However, in practice, an analytic (symbolic) approach is required to evaluate expressions with multiple steps or operands unless the expression can be computed by pointwise evaluation of the operands. This is due to the search for extrema ($\inf, \sup$) over large time intervals as required for operations such as $\otimes$ which, if no analytic knowledge can be exploited, becomes computationally very expensive, effectively limiting a numerical approach to a single hop.

While for time-invariant Network Calculus (restricted to piecewise-linear, ultimately pseudo-periodic functions), at least the algorithms are published [Bou14] research yet has to provide similarly powerful algorithms for time-variant Network Calculus to make the evaluation of multi-hop scenarios feasible.

6.6 Related Work

Network Calculus has been applied to some time-triggered network elements with intermittent service. TDMA systems have been targeted in [Wan+06a; Gol+07; Kha+14]. In [Wan+06a; Gol+07], insights from the Network Calculus analysis of a TDMA system (distributed embedded system with bus-interconnection [Wan+06a], wireless sensor network [Gol+07]) are used as input for the optimization of the TDMA schedule. In [Kha+14], the tightness of worst-case bounds is improved, taking into account various scheduling policies and packetization of data.

More recently, Ethernet-based real-time networks (TTEthernet, IEEE 802.1 TSN) have been subject to Network Calculus analysis [Zha+17; Zha+18a; Zha+18b]. Concerning the analysis of Ethernet-based real-time networks, the goal is to derive worst-case bounds for the traffic factoring in the impact of various, implementation-specific integration effects. For example, for the analysis of IEEE 802.1 TSN networks, the integration effects of Credit-based Shapers, Time-aware Shapers, and Frame-preemption are considered in [Zha+18b], and in [Zha+18a] the combination of Time-aware Shapers, Length-aware Scheduling, and strict-priority frame-selection policy is considered.

All of the work above has in common that the service curve formulations presented in these papers assume one service interval per cycle and all of the service curve formulations are time-invariant, and can—eventually—be traced back to a peak-rate service process with constant service rate. Besides explicitly anticipating more intricate schedules with multiple service intervals per cycle in our service curve formulations, we show that these approaches are not sufficient for service curves for time-triggered network elements with intermittent service processes with ultimately non-constant service rate.

A different approach is pursued in [Bec+19] where *stochastic*, time-variant Network Calculus is combined with a measurement-based approach to model cellular sleep scheduling. Service intermittence due to "sleeping" is considered a random process in [Bec+19], whereas we assume a repeating, deterministic schedule for service intermittence. In [Bec+15; Bec+19], also the notion of regenerative service processes, where the service process is restarted at regeneration points, is explicitly mentioned. While we started with time-variant service curve formulations (albeit using deterministic Network Calculus with cyclic schedules), we covered systems with and without "restarting" service processes and compared them to time-invariant service curve formulations.

6.7 Summary

In this chapter, we investigated service curves for network elements with service intermittence and resumption triggered at times derived from a cyclic schedule. We identified two archetypes of such network elements, namely, time-triggered network elements with blocking, and time-triggered network elements with halt-restart behavior.

The type of the network element and the service curve are tightly coupled. Therefore, we presented distinct time-variant formulations for each type of time-triggered network element and also showed how to obtain the respective time-invariant service curves. We show that for a generic service process S, the computation of the actual values of the time-invariant service curve of the network element might involve evaluating an arbitrary number of expressions. In our numerical evaluations, we observed that the time-invariant service curves result in large over-estimations of the worst-case bounds compared to the time-variant service curve formulation and we identified influencing factors.

Ultimately, our findings highlight some challenges for analyzing time-triggered network elements with intermittent service with NC: time-invariant service curves suffer from under-approximation when compared to the time-variant service curve and, depending on β_{S}, might even be infeasible to evaluate computationally. Time-variant service curves offer tighter bounds, but their use, for example, for large multi-hop scenarios, is impeded by the lack of efficient algorithms for time-variant Network Calculus.

7 Conclusion

In this thesis, we have presented different methods for traffic planning—considering scheduling and routing—for time-triggered flows in the first part of this thesis (Chap. 2 – Chap. 5), and we have developed service curves for network elements with time-triggered service intermittence to facilitate the analysis of complex networking scenarios within the Network Calculus framework in the second part of this thesis (Chap. 6).

7.1 Summary

In more detail: We have presented an ILP formulation for joint scheduling and route computation in Sec. 4.2. Traffic planning instances where the number of flows and the number of nodes in the network are in the double-digit range, respectively, result in ILPs with millions of constraints and variables. Despite the NP-hardness of the problem, these problem instances can often be solved within reasonable time (seconds to hours) with state-of-the-art equipment.

Our second ILP formulation for joint scheduling and path selection (see Sec. 4.3) uses candidate paths and represents the reserved window in terms of discrete, occupied cells in the transmission cycle. Restricting the set of pre-computed candidate paths can be used to influence the size of the ILP instances and thereby allows trading off between solver runtime and coverage of the solution space.

We introduced the concept of complemental flows for applications in converged networks and extended our ILP formulations to jointly compute optimized routes and schedules for applications with complemental flows in Sec. 4.4. These approaches can be used, for example, for better placement of complemental flows in the network compared to approaches that treat the time-triggered part and then non-time-triggered traffic part separately even with limited solver runtime.

Then we proceeded from a method relying on constraint-programming (ILP) to a conflict-graph-based approach. We presented an approach for solving the traffic planning problem for time-triggered traffic in data networks with conflict graphs in Sections 5.1 and 5.2. We explained how to construct a conflict graph for a given traffic planning problem, and we showed how to solve the traffic planning problem by finding independent vertex sets which cover all flows. With the conflict-graph-based approach, it is comparably cheap to get feasible (partial) solutions, since there exist efficient algorithms for finding independent sets even in large graphs. We presented a proof-of-concept implementation for the conflict-graph-based approach.

We extended the conflict-graph approach for scenarios with dynamic changes in the flow set. To this end, we presented an approach for offensive dynamic traffic planning, which allows reconfiguring active flows to achieve better network utilization in Sec. 5.3. This dynamic traffic planning approach allows us to quantify and control the QoS degradation a flow may suffer during such reconfigurations.

Finally, we investigated service curves for network elements with service intermittence and resumption triggered by a cyclic schedule in Chap. 6. We derived time-variant and time-invariant service-curve formulations for the different types of network elements with time-triggered service intermittence. These service curves can be interpreted as descriptions of the system characteristics. We evaluated the different service curve formulations numerically, using a traffic independent method that allows a more general comparison of the pessimism caused by the different modeling. In our evaluations, we observed that the time-invariant service curves can result in large over-estimations of the worst-case bounds compared to the time-variant service curves.

7.2 Outlook

There are two main arcs along which our work on the traffic planning problem from the first part of this thesis (Chap. 2 – Chap. 5) could be extended: improving the methods for traffic planning and extending the traffic planning problem itself.

Improving the methods for traffic planning is especially promising for the conflict-graph approaches. Interesting directions for future work include investigating memory-access optimized conflict-graph data structures which take into account the dynamic nature of the conflict graph, meta-heuristics that exploit the conflict-graph properties as well as different strategies for the conflict-graph growth.

Developing new exact or approximate algorithms for the maximum colorful independent vertex set problem in the conflict graph is interesting, too. Here, it is especially interesting to investigate algorithms that can be aborted at any time and still yield a valid partial solution, and algorithms for dynamic graph structures which can efficiently exploit the changes in the graph and do not require a complete re-execution.

Another interesting problem in the context of conflict graphs is the question of how to compute optimal traffic plans. This requires us to think about how to formulate an objective in terms of flow configurations, too.

When thinking about the traffic planning problem itself, there is a multitude of possible extensions. We could account for artifacts such as synchronization errors, non-constant delays, etc., which could be mitigated by additional safety margins in the reservation windows. Failures of links and nodes may require us to compute traffic plans with in-built redundancy and add additional requirements on the update protocol for dynamic traffic planning. Relaxations of the zero-queuing assumption will also add new challenges to the traffic planning process, not only with regard to the quality of service guarantees but also with regard to the representation of buffered packets in a feasible way. Finally, an additional degree of freedom can be gained by allowing for more dynamics in the traffic plan, for example, dynamic routing with different packets of an application being forwarded along different paths depending on the current time. Here, it will be interesting how to make traffic planning with these additional degrees of freedom computationally tractable.

It would be also interesting to tie the Network Calculus modeling from Chap. 6 to the traffic planning itself, for example, in an extension to our work on optimized traffic plans for complemental flows. Given that time-variant service curves offer tighter bounds, but their use, for example, for large multi-hop scenarios, is hampered due to the lack of efficient algorithms for time-variant Network Calculus the development of efficient computational methods for time-variant Network Calculus operations is well-motivated, too.

During the course of our work, we also often encountered a seemingly fundamental challenge in networking with regards to the three properties: **q**uality (of the communication service), **u**tilization (of the network resources), and **d**ynamics (of the traffic). Analogously to other results for distributed systems, this insinuates the formulation of the QUD-conjecture: We can optimize networking designs for any two of the three properties quality of service, resource utilization, and traffic dynamics, at the cost of the respective third property. We think that expressing this conjecture in more concrete terms and trying to prove or disprove it can result in interesting insights for the design of future communication networks.

A Appendix

A.1 Joint Scheduling and Route Computation

A.1.1 Plain ILP Formulation

To express implications in the form

$$\text{if (lin. constraint 1) then (lin. constraint 2)} \tag{A.1}$$

we used the if-operator. Eq. (A.1) expresses that the linear constraint 2 only has to be satisfied if linear constraint 1 is satisfied. Similarly, to express disjunctions in the form

$$\text{(lin. constraint 1 or lin. constraint 2)} \tag{A.2}$$

where only one of the constraints needs to apply, we used the or-operator.

In case of bounded variables, the expressions from Eq. (A.1) and Eq. (A.2) can be expressed with a set of plain integer linear inequalities and additional auxiliary variables and parameters. The key idea in resolving the if-operator and the or-operator is to introduce a large enough parameter—commonly referred to as *bigM*—which can be used in combination with an auxiliary variable to force any linear constrain to a true statement. We will not go into the details of ILP modeling here, but we want to give at least some intuition with an example:

Assume that

$$\left(\sum_i a_i \underline{x_i}\right) - b \leq 0 \tag{A.3}$$

shall only be satisfied, if the binary variable $\underline{w}$ has value 1. This can be expressed with the constraint

$$\left(\left(\sum_i a_i \underline{x_i}\right) - b\right) - ((1 - \underline{w}) \cdot M_{\text{aux}}) \leq 0. \tag{A.4}$$

If $\underline{w} = 1$, then Eq. (A.4) reduces to Eq. (A.3). This means the ILP solver has to assign values to the $\underline{x_i}$-variables that satisfy Eq. (A.3). However, if $\underline{w} = 0$, then we have

$$\left(\left(\sum_i a_i \underline{x_i}\right) - b\right) - M_{\text{aux}} \leq 0. \tag{A.5}$$

It is easy to see that we can a priori compute a large enough value for M_{aux} such that Eq. (A.5) will be true for any assignment to the $\underline{x_i}$-variables if we know the range of the values for each $\underline{x_i}$.

If we apply this principle, we can express the ILP formulation from Sec. 4.2 using only linear inequalities.

A.1.2 Scheduling Constraints

To express if and or from Eq. (4.12)–Eq. (4.14) with linear integer inequalities, we need additional auxiliary variables.

We introduce the binary variable $\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell] \in \{0, 1\}$ with $f_1, f_2 \in \mathcal{F}$ and $\ell \in \mathcal{L}$. The constraints (A.6)-(A.8) force $\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell] = 1$ if link ℓ is used by both, f_1 and f_2, else $\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell] = 0$.

$$\begin{aligned}
&\forall f_1 \in \mathcal{F}, \forall f_2 \in \mathcal{F} : f_1 \neq f_2, \forall \ell \in \mathcal{L} : \\
&\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell] \leq \underline{\mathbf{u}_{\text{link}}}[f_1, \ell] && \text{(A.6)} \\
&\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell] \leq \underline{\mathbf{u}_{\text{link}}}[f_2, \ell] && \text{(A.7)} \\
&\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell] \geq \underline{\mathbf{u}_{\text{link}}}[f_1, \ell] + \underline{\mathbf{u}_{\text{link}}}[f_2, \ell] - 1 && \text{(A.8)}
\end{aligned}$$

In other words, $\underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell]$ indicates whether the condition from Eq. (4.12) holds.

To model the disjunction for the order of the reserved windows, we introduce another auxiliary variable $\underline{\mathbf{w}_{\text{resv}}}[f_1, f_2, \ell, a, b] \in \{0, 1\}$ with $f_1, f_2 \in \mathcal{F}$, $\ell \in \mathcal{L}$ and $a \in \mathcal{A}$, $b \in \mathcal{B}$ as defined in Equations (4.15) and (4.16). Then we can constrain the offsets of all flows whose packets traverse an link ℓ such that the reservation windows are placed in mutually exclusive time intervals with

$$\begin{aligned}
&\forall f_1 \in \mathcal{F}, \forall f_2 \in \mathcal{F} : f_1 \neq f_2, \forall \ell \in \mathcal{L}, \forall a \in \mathcal{A}, \forall b \in \mathcal{B} : \\
&\big(\underline{\mathbf{t}_{\text{offset}}}[f_2, \ell] + b \cdot \mathbf{t}_{\text{cycle}}[f_2] + \mathbf{t}_{\text{resv}}[f_2] - \big(\underline{\mathbf{t}_{\text{offset}}}[f_1, \ell] + a \cdot \mathbf{t}_{\text{cycle}}[f_1]\big)\big) \\
&\quad - \big(\big(1 - \underline{\mathbf{w}_{\text{resv}}}[f_1, f_2, \ell, a, b]\big) + \big(1 - \underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell]\big)\big) \cdot M_{\text{aux}} \leq 0 && \text{(A.9)} \\
&\big(\underline{\mathbf{t}_{\text{offset}}}[f_1, \ell] + a \cdot \mathbf{t}_{\text{cycle}}[f_1] + \mathbf{t}_{\text{resv}}[f_1] - \big(\underline{\mathbf{t}_{\text{offset}}}[f_2, \ell] + b \cdot \mathbf{t}_{\text{cycle}}[f_2]\big)\big) \\
&\quad - \big(\underline{\mathbf{w}_{\text{resv}}}[f_1, f_2, \ell, a, b] + \big(1 - \underline{\mathbf{w}_{\text{link}}}[f_1, f_2, \ell]\big)\big) \cdot M_{\text{aux}} \leq 0 && \text{(A.10)}
\end{aligned}$$

We can set the big constant M_{aux}, for instance, to ($M_{\text{aux}} = 2 \cdot t_{\text{hyper}}$).

A.1.3 Joint Scheduling and Routing Constraints

To express the constraints that link the routing and scheduling constraints from Eq. (4.17)-Eq. (4.19) in plain integer linear inequalities, we similarly need two sets of additional auxiliary binary variables.

The variable $\underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] \in \{0, 1\}$ with $f \in \mathcal{F}$ and $\ell_{\mathrm{in}}, \ell_{\mathrm{out}} \in \mathcal{L}$ is used as helper variable to model the implication from Eq. (4.17). The constraints

$$\begin{aligned}
\forall f \in \mathcal{F}, \forall n \in \mathcal{N} :& \\
\forall \ell_{\mathrm{in}} \in \mathcal{L}_{\mathrm{in},n}, \forall \ell_{\mathrm{out}} \in \mathcal{L}_{\mathrm{out},n} :& \\
\underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] \leq \underline{\mathbf{u}_{\mathrm{link}}}[f, \ell_{\mathrm{in}}]& \qquad \text{(A.11)} \\
\underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] \leq \underline{\mathbf{u}_{\mathrm{link}}}[f, \ell_{\mathrm{out}}]& \qquad \text{(A.12)} \\
\underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] \geq \underline{\mathbf{u}_{\mathrm{link}}}[f, \ell_{\mathrm{in}}] + \underline{\mathbf{u}_{\mathrm{link}}}[f, \ell_{\mathrm{out}}] - 1& \qquad \text{(A.13)}
\end{aligned}$$

can be interpreted as follows: if ℓ_{in} is followed by ℓ_{out} on the route of flow f—the route of flow f contains the sequence $\ell_{\mathrm{in}}, \ell_{\mathrm{out}}$—, then $\underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] = 1$, else $\underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] = 0$.

If $(\ell_{\mathrm{in}}, \ell_{\mathrm{out}})$ is a part of the route of flow f, then we have to enforce the relation of the offsets for the reserved windows on subsequent links on the route of each flow f.

The disjunction from Equations (4.18) and (4.19) which, in simplified terms, is required since a reserved window may either occur on the next link ℓ_{out} later in the same cycle or (modulo t_{cycle}) earlier in a later cycle is modeled with another binary auxiliary variable $\underline{\mathbf{w}_{\mathrm{cycle}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]$ with $f \in \mathcal{F}$ and $\ell_{\mathrm{in}}, \ell_{\mathrm{out}} \in \mathcal{L}$.

Combining implication, disjunction, and the equality from Eq. (4.17)–Eq. (4.19), we end up with the four linear inequality constraints

$$\begin{aligned}
\forall f \in \mathcal{F}, \forall n \in \mathcal{N} :& \\
\forall \ell_{\mathrm{in}} \in \mathcal{L}_{\mathrm{in},n}, \forall \ell_{\mathrm{out}} \in \mathcal{L}_{\mathrm{out},n} :& \\
\left(\underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{out}}] - \underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{in}}] - \mathbf{d}_{\mathrm{hop}}[f]\right)& \\
- \left(\underline{\mathbf{w}_{\mathrm{cycle}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] + \left(1 - \underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]\right)\right) \cdot M_{\mathrm{aux}} \leq 0& \qquad \text{(A.14)} \\
\left(\underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{out}}] - \underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{in}}] - \mathbf{d}_{\mathrm{hop}}[f]\right)& \\
+ \left(\underline{\mathbf{w}_{\mathrm{cycle}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}] + \left(1 - \underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]\right)\right) \cdot M_{\mathrm{aux}} \geq 0& \qquad \text{(A.15)} \\
\left(\underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{out}}] + \mathbf{t}_{\mathrm{cycle}}[f] - \underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{in}}] - \mathbf{d}_{\mathrm{hop}}[f]\right)& \\
- \left(\left(1 - \underline{\mathbf{w}_{\mathrm{cycle}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]\right) + \left(1 - \underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]\right)\right) \cdot M_{\mathrm{aux}} \leq 0& \qquad \text{(A.16)} \\
\left(\underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{out}}] + \mathbf{t}_{\mathrm{cycle}}[f] - \underline{\mathbf{t}_{\mathrm{offset}}}[f, \ell_{\mathrm{in}}] - \mathbf{d}_{\mathrm{hop}}[f]\right)& \\
+ \left(\left(1 - \underline{\mathbf{w}_{\mathrm{cycle}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]\right) + \left(1 - \underline{\mathbf{w}_{\mathrm{seq}}}[f, \ell_{\mathrm{in}}, \ell_{\mathrm{out}}]\right)\right) \cdot M_{\mathrm{aux}} \geq 0.& \qquad \text{(A.17)}
\end{aligned}$$

Again, $M_{\mathrm{aux}} = 2 \cdot t_{\mathrm{hyper}}$ is a reasonable choice.

A.1.4 Implementation of ILP Model with Matrices

Depending on the toolchain, it is more natural to implement the ILP model from Sec. 4.2 using matrices derived from the $\boldsymbol{G}_n$, namely, the link-link adjacency matrix $\mathbf{A}_{LL}$ and the node-link incidence matrix $\mathbf{B}_{NL}$, for example, if it is not straightforward to express $\mathcal{L}_{\text{in},n}, \mathcal{L}_{\text{out},n}$ otherwise.

The value of an element of the link-link adjacency matrix $\mathbf{A}_{LL} \in \{0,1\}^{|\mathcal{L}|\times|\mathcal{L}|}$ in row $\ell_{\text{in}} \in \mathcal{L}$ and column $\ell_{\text{out}} \in \mathcal{L}$ is defined by

$$\mathbf{A}_{LL}[\ell_{\text{in}}, \ell_{\text{out}}] = \begin{cases} 1, \text{ if } \mathrm{n}_{\text{in}}(\ell_{\text{in}}) = \mathrm{n}_{\text{out}}(\ell_{\text{out}}) \\ 0, \text{ otherwise} \end{cases} . \tag{A.18}$$

In words, $\mathbf{A}_{LL}[\ell_{\text{in}}, \ell_{\text{out}}] = 1$, if a packet that is received via the link ℓ_{in} at node $n \in \mathcal{N}$ can be transmitted via link ℓ_{out} from the same node n.

The value of an element of the node-link incidence matrix $\mathbf{B}_{NL} \in \{-1,0,1\}^{|\mathcal{N}|\times|\mathcal{L}|}$ in row $n \in \mathcal{N}$ and column $\ell \in \mathcal{L}$ is defined by

$$\mathbf{B}_{NL}[n, \ell] = \begin{cases} 1, \text{ if } \mathrm{n}_{\text{out}}(\ell) = n \\ -1, \text{ if } \mathrm{n}_{\text{in}}(\ell) = n \\ 0, \text{ otherwise} \end{cases} . \tag{A.19}$$

In words, $\mathbf{B}_{NL}$ encodes whether the link ℓ is an outgoing link at node n (then $\mathbf{B}_{NL}[n, \ell] = 1$), or if the link ℓ is an incoming link at n, (then $\mathbf{B}_{NL}[n, \ell] = -1$).

Note that we only need to store the non-zero components of $\mathbf{A}_{LL}$ and $\mathbf{B}_{NL}$, that is, we can use a sparse matrix representation for $\mathbf{A}_{LL}$ and $\mathbf{B}_{NL}$.

With these two matrices, we can replace the sum of elements in $\mathcal{L}_{\text{in},n}$ and $\mathcal{L}_{\text{out},n}$ in the ILP constraints from Sec. 4.2 by sums over matrix components:

Routing Constraint Eq. (4.8)

$$\forall f \in \mathcal{F} : \sum_{\ell \in \mathcal{L}} \mathbf{B}_{NL} \left[\mathbf{n}_{\text{src}}[f], \ell\right] \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell] = 1 \tag{A.20}$$

Routing Constraint Eq. (4.9)

$$\forall f \in \mathcal{F} : \sum_{\ell \in \mathcal{L}} \mathbf{B}_{NL} \left[\mathbf{n}_{\text{dst}}[f], \ell\right] \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell] = -1 \tag{A.21}$$

Routing Constraint Eq. (4.10)

$$\forall f \in \mathcal{F} : \forall n \in \mathcal{N} \setminus \{\mathbf{n}_{\text{src}}[f], \mathbf{n}_{\text{dst}}[f]\} :$$

$$\sum_{\ell_{\text{out}} \in \{\ell \in \mathcal{L} | \mathbf{B}_{NL}[n,\ell]=1\}} \mathbf{B}_{NL}[n, \ell_{\text{out}}] \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out}}] \tag{A.22}$$

$$= - \sum_{\ell_{\text{in}} \in \{\ell \in \mathcal{L} | \mathbf{B}_{NL}[n,\ell]=-1\}} \mathbf{B}_{NL}[n, \ell_{\text{in}}] \cdot \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in}}] \tag{A.23}$$

Joint Routing and Scheduling Constraints Eq. (4.17)–Eq. (4.19)

$$\forall f \in \mathcal{F}, \forall \ell_{\text{in}}, \ell_{\text{out}} \in \{\mathcal{L} \times \mathcal{L} | \ell_{\text{in}} \neq \ell_{\text{out}} \text{ and } \mathbf{A}_{LL}[\ell_{\text{in}}, \ell_{\text{out}}] = 1\} :$$

$$\text{if} (\underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{in}}] + \underline{\mathbf{u}_{\text{link}}}[f, \ell_{\text{out}}] \geq 2) \text{ then} \tag{A.24}$$

$$(\ \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{out}}] = \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}] + \mathbf{d}_{\text{hop}}[f] \tag{A.25}$$

$$\text{or } \ \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{out}}] + \mathbf{t}_{\text{cycle}}[f] = \underline{\mathbf{t}_{\text{offset}}}[f, \ell_{\text{in}}] + \mathbf{d}_{\text{hop}}[f] \ ) \tag{A.26}$$

A.1.5 Evaluation Results

Here, we provide the numerical data for the evaluations in Sec. 4.2.2.

Each table contains the average runtime (column “mean”), and the standard deviation of the runtimes (column “std”). The columns labeled “fin” contain the number of problem instances, where the solver finished in time, that is, either the solver found a solution or declared infeasibility of the problem within the allowed time limit. The columns labeled “infs” contain the number of infeasible problem instances, that is, where no solution exists. The columns labeled “tot” contain the total number of problem instances that the solver attempted to solve. Subtracting the number of finished problem instances from the number of total problem instances yields the number of unsolved problem instances.

The irregular amount of problem instances with random topology is an artifact of the graph generation, which may return disconnected graphs. If the graph generators produced disconnected graphs, we selected the largest connected component.

TABLE A.1 Varying number of flows, $|\mathcal{N}| = 8$, transmission cycles $\in \{50, 100, 200, 400, 800\}$ (HF), and runtime limit of 30 min.

$\lvert\mathcal{F}\rvert$	Line					Ring					Scale-Free					Random				
	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot
2	0.03	0.02	10	0	10	0.06	0.06	10	0	10	0.03	0.01	10	0	10	0.08	0.11	10	0	10
3	0.20	0.23	10	0	10	0.70	0.85	10	0	10	0.14	0.19	10	0	10	0.59	0.70	10	0	10
4	0.46	0.50	10	0	10	1.63	1.15	10	0	10	0.39	0.29	10	0	10	1.57	1.84	10	0	10
5	1.39	0.76	10	0	10	3.34	2.24	10	0	10	1.04	0.58	10	0	10	4.57	2.87	10	0	10
6	2.05	1.31	10	0	10	6.54	3.70	10	0	10	1.49	0.99	10	0	10	7.14	4.95	10	0	10
7	3.55	2.06	10	0	10	10.35	1.79	10	0	10	4.27	2.48	10	0	10	11.86	4.29	10	0	10
8	3.17	1.00	10	0	10	13.19	9.36	10	0	10	7.78	4.34	10	0	10	16.80	3.82	10	0	10
9	7.66	3.26	10	0	10	22.62	7.75	10	0	10	7.84	5.44	10	0	10	26.92	9.23	10	0	10
10	7.68	4.77	10	1	10	28.67	7.42	10	0	10	10.88	4.73	10	0	10	33.18	16.78	10	0	10
11	16.83	7.35	10	0	10	31.59	12.46	10	0	10	12.29	3.49	10	0	10	38.41	5.55	10	0	10
12	18.09	6.08	10	0	10	35.32	14.55	10	0	10	18.04	8.93	10	0	10	69.20	26.69	10	0	10
13	21.23	10.58	10	0	10	64.56	22.18	10	0	10	31.70	18.24	10	0	10	70.46	27.18	10	0	10
14	30.63	11.46	10	1	10	82.12	36.09	10	0	10	22.47	6.78	10	0	10	118.91	51.36	10	0	10
15	36.47	20.20	10	0	10	98.23	36.66	10	0	10	29.87	12.32	10	0	10	154.37	37.26	10	0	10

TABLE A.2 Varying number of flows, $|\mathcal{N}| = 8$, transmission cycles $\in \{1000, 2000, 4000, 8000\}$ (LF), and runtime limit of 30 min.

$\lvert\mathcal{F}\rvert$	Line					Ring					Scale-Free					Random				
	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot
2	0.02	0.00	10	0	10	0.06	0.04	10	0	10	0.05	0.07	10	0	10	0.16	0.12	10	0	10
3	0.12	0.16	10	0	10	0.19	0.12	10	0	10	0.13	0.18	10	0	10	0.31	0.23	10	0	10
4	0.44	0.32	10	0	10	1.23	0.51	10	0	10	0.24	0.20	10	0	10	1.84	0.92	10	0	10
5	1.17	0.65	10	0	10	2.47	0.91	10	0	10	1.47	1.25	10	0	10	4.68	1.64	10	0	10
6	2.86	1.30	10	0	10	3.71	0.93	10	0	10	2.68	1.53	10	0	10	5.10	2.17	10	0	10
7	2.23	1.00	10	0	10	5.12	1.24	10	0	10	2.70	1.82	10	0	10	8.44	2.05	10	0	10
8	4.37	1.39	10	0	10	8.97	2.20	10	0	10	3.42	2.67	10	0	10	11.05	3.98	10	0	10
9	5.47	1.16	10	0	10	11.82	4.68	10	0	10	6.56	2.00	10	0	10	14.29	3.17	10	0	10
10	6.64	2.11	10	0	10	15.73	5.58	10	0	10	8.43	2.74	10	0	10	20.91	4.17	10	0	10
11	10.26	2.84	10	0	10	20.86	3.35	10	0	10	10.29	3.27	10	0	10	25.29	3.74	10	0	10
12	14.47	3.82	10	0	10	29.00	8.70	10	0	10	12.79	2.90	10	0	10	39.51	21.06	10	0	10
13	14.13	3.36	10	0	10	34.57	11.21	10	0	10	17.30	4.92	10	0	10	47.91	8.08	10	0	10
14	16.79	4.56	10	0	10	48.49	14.34	10	0	10	17.80	4.94	10	0	10	52.08	12.49	10	0	10
15	27.02	7.32	10	0	10	56.36	11.48	10	0	10	19.91	2.69	10	0	10	70.42	14.59	10	0	10

TABLE A.3 Varying number of flows, $|\mathcal{N}| = 8$, transmission cycles $\in \{1000, 2000, 4000, 8000\}$ (LF), and runtime limit of 60 min.

$\|\mathcal{F}\|$	Line					Ring					Scale-Free					Random				
	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot
16	31.84	6.68	10	0	10	76.63	21.54	10	0	10	32.99	7.47	10	0	10	104.51	14.60	10	0	10
17	39.08	8.08	10	0	10	103.94	41.51	10	0	10	44.40	12.22	10	0	10	120.71	45.96	10	0	10
18	44.60	10.57	10	0	10	126.88	67.88	10	0	10	47.88	8.20	10	0	10	149.06	36.33	10	0	10
19	49.25	10.30	10	0	10	152.60	68.84	10	0	10	57.01	12.11	10	0	10	179.70	41.53	10	0	10
20	58.21	11.47	10	0	10	191.83	33.25	10	0	10	61.51	16.95	10	0	10	241.87	71.28	10	0	10
21	70.53	15.92	10	0	10	185.89	22.57	10	0	10	62.11	16.91	10	0	10	254.20	141.58	9	0	10
22	87.40	13.19	10	0	10	232.19	40.95	10	0	10	101.42	16.14	10	0	10	370.90	170.34	10	0	10
23	86.84	20.50	10	0	10	247.50	69.91	10	0	10	106.02	22.26	10	0	10	400.43	307.67	10	0	10
24	111.76	26.96	10	0	10	334.76	91.93	10	0	10	119.64	22.37	10	0	10	420.31	202.31	10	0	10
25	116.74	19.19	10	0	10	428.97	224.16	10	0	10	149.12	27.43	10	0	10	449.60	152.85	10	0	10
26	147.02	25.24	10	0	10	354.06	68.11	10	0	10	148.77	30.43	10	0	10	643.39	237.89	10	0	10
27	169.85	25.60	10	0	10	502.09	165.25	10	0	10	203.17	34.09	10	0	10	543.48	134.24	10	0	10
28	196.12	36.78	10	0	10	432.13	92.62	10	0	10	206.94	22.74	10	0	10	937.55	378.95	10	0	10
29	223.86	51.63	10	0	10	564.75	141.50	10	0	10	221.54	40.19	10	0	10	1020.50	356.87	10	0	10
30	245.73	61.84	10	0	10	776.78	269.99	10	0	10	283.55	63.72	10	0	10	1056.51	303.14	9	0	10

TABLE A.4 Varying number of nodes, $|\mathcal{F}| = 7$, transmission cycles $\in \{50, 100, 200, 400, 800\}$ (HF), and runtime limit of 30 min.

$\|\mathcal{N}\|$	Line					Ring					Scale-Free					Random				
	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot
5	1.17	0.53	10	0	10	5.06	2.76	10	0	10	1.62	0.87	10	0	10	3.93	3.90	8	0	8
6	1.54	0.95	10	0	10	7.77	3.10	10	0	10	2.12	1.10	10	0	10	5.79	3.03	12	0	12
7	2.10	0.90	10	0	10	7.72	2.73	10	0	10	2.33	1.33	10	0	10	6.07	6.24	5	0	5
8	2.38	1.58	10	0	10	9.89	5.15	10	0	10	2.42	1.24	10	0	10	10.26	4.89	11	0	11
9	3.53	1.98	10	0	10	11.40	3.43	10	0	10	4.20	1.78	10	0	10	17.48	8.02	10	0	10
10	2.94	2.66	10	0	10	13.12	4.52	10	0	10	4.10	2.50	10	0	10	12.58	7.03	20	0	20
11	3.77	1.56	10	0	10	16.20	7.27	10	0	10	6.14	4.52	10	0	10	13.71	10.30	6	0	6
12	3.96	2.86	10	2	10	18.56	7.29	10	0	10	6.25	2.62	10	0	10	20.68	8.20	12	0	12
13	7.54	3.74	10	0	10	17.98	6.49	10	0	10	5.47	2.95	10	0	10	18.29	8.50	8	0	8
14	7.64	3.74	10	1	10	23.63	11.21	10	0	10	6.47	3.89	10	0	10	27.13	9.46	14	0	14
15	9.77	5.30	10	3	10	22.11	6.92	10	1	10	15.11	5.61	10	0	10	23.59	16.99	10	0	10
16	8.06	3.44	10	1	10	18.71	7.86	10	1	10	9.93	4.37	10	0	10	30.21	18.50	12	1	13
17	12.53	5.93	10	3	10	26.05	10.53	10	1	10	13.30	6.10	10	0	10	32.34	23.98	4	0	4
18	7.20	3.53	10	1	10	28.80	8.59	10	1	10	16.30	10.68	10	0	10	40.30	17.46	10	0	10
19	13.78	9.85	10	2	10	25.55	8.81	10	2	10	17.47	11.89	10	0	10	267.98	462.89	7	0	7
20	15.20	13.01	10	5	10	24.60	7.92	10	1	10	14.41	8.98	10	0	10	579.91	771.72	2	0	2

TABLE A.5 Varying number of nodes, $|\mathcal{F}| = 7$, transmission cycles $\in \{1000, 2000, 4000, 8000\}$ (LF), and runtime limit of 30 min.

$\|\mathcal{N}\|$	Line					Ring					Scale-Free					Random				
	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot
5	1.02	0.50	10	0	10	2.43	0.87	10	0	10	1.09	0.70	10	0	10	2.03	2.18	10	0	10
6	1.59	0.90	10	0	10	3.44	0.94	10	0	10	1.61	1.16	10	0	10	3.84	1.45	12	0	12
7	2.13	1.07	10	0	10	4.94	2.04	10	0	10	2.40	1.48	10	0	10	3.26	2.45	5	0	5
8	2.46	1.53	10	0	10	4.28	1.51	10	0	10	2.92	1.12	10	0	10	10.06	3.87	12	0	12
9	3.93	1.89	10	0	10	5.86	1.46	10	0	10	2.84	0.93	10	0	10	10.76	3.54	15	0	15
10	5.23	2.36	10	0	10	9.14	4.56	10	0	10	3.34	1.86	10	0	10	7.96	4.06	13	0	13
11	4.93	1.87	10	0	10	8.15	2.66	10	0	10	3.97	1.48	10	0	10	12.37	6.07	12	0	12
12	5.53	2.52	10	0	10	9.47	3.60	10	0	10	5.42	2.44	10	0	10	16.75	4.24	7	0	7
13	4.91	2.58	10	0	10	10.42	4.35	10	0	10	6.99	2.13	10	0	10	17.60	7.28	10	0	10
14	6.69	3.70	10	0	10	12.68	3.13	10	0	10	8.34	3.65	10	0	10	23.56	11.10	16	0	16
15	7.48	2.45	10	0	10	14.06	3.91	10	0	10	7.03	4.04	10	0	10	18.80	7.96	14	0	14
16	8.88	3.50	10	0	10	14.21	6.13	10	0	10	7.78	3.74	10	0	10	25.36	10.55	7	0	7
17	9.24	4.29	10	0	10	15.68	5.84	10	0	10	10.23	4.41	10	0	10	27.51	22.11	3	0	3
18	10.85	4.13	10	0	10	17.42	4.65	10	0	10	9.21	4.21	10	0	10	40.37	10.31	10	0	10
19	11.19	3.86	10	0	10	18.97	7.45	10	0	10	10.09	3.93	10	0	10	45.91	13.02	5	0	5
20	13.56	4.35	10	0	10	21.46	9.00	10	0	10	10.39	6.55	10	0	10	54.48	10.05	4	0	4

TABLE A.6 Varying number of nodes, $|\mathcal{F}| = 7$, transmission cycles $\in \{1000, 2000, 4000, 8000\}$ (LF), and runtime limit of 60 min.

$\|\mathcal{N}\|$	Line					Ring					Scale-Free					Random				
	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot	mean [s]	std [s]	fin	infs	tot
21	11.88	5.10	10	0	10	24.47	5.67	10	0	10	13.16	5.74	10	0	10	41.93	16.31	12	0	12
22	11.84	6.32	10	0	10	29.31	14.83	10	0	10	19.09	5.37	10	0	10	62.66	20.97	8	0	8
23	13.81	3.83	10	0	10	23.97	7.74	10	0	10	15.01	7.25	10	0	10	64.87	29.10	7	0	7
24	15.70	8.27	10	0	10	24.02	6.03	10	0	10	15.68	7.86	10	0	10	102.07	70.18	15	0	15
25	16.72	5.64	10	0	10	31.57	6.42	10	0	10	12.94	5.74	10	0	10	94.52	88.72	13	0	13
26	16.59	5.24	10	0	10	27.50	14.98	10	0	10	15.04	6.79	10	0	10	52.22	25.84	10	0	11
27	17.38	7.09	10	0	10	35.31	7.13	10	0	10	23.08	10.05	10	0	10	80.31	72.07	10	0	10
28	15.70	10.20	10	0	10	39.21	24.98	10	0	10	24.13	11.18	10	0	10	258.12	434.34	8	0	8
29	19.04	7.63	10	0	10	35.95	9.02	10	0	10	20.39	9.24	10	0	10	109.70	22.44	4	0	4
30	24.79	7.77	10	0	10	45.12	15.22	10	0	10	20.99	12.23	10	0	10	339.23	409.29	9	0	9
31	17.12	5.39	10	0	10	38.67	10.35	10	0	10	28.87	11.78	10	0	10	238.52	312.44	10	0	10
32	22.10	8.99	10	0	10	46.43	16.13	10	0	10	22.67	12.87	10	0	10	138.26	35.99	9	0	9
33	24.99	11.63	10	0	10	55.61	22.75	10	0	10	30.29	13.91	10	0	10	313.89	238.65	7	0	7
34	23.48	9.38	10	0	10	53.57	20.63	10	0	10	27.49	14.50	10	0	10	145.48	77.56	6	0	7
35	24.59	7.53	10	0	10	55.90	17.79	10	0	10	33.87	12.15	10	0	10	267.12	198.14	7	0	8
36	26.03	10.95	10	0	10	64.76	42.79	10	0	10	44.53	8.51	10	0	10	361.76	426.08	3	0	4

A.2 Greedy Flow Heap Heuristic

The Greedy Flow Heap Heuristic (GFH) is a heuristic that searches for an independent vertex set in a vertex-colored graph. The GFH has been developed by Heiko Geppert in the context of the joint work on dynamic traffic planning [Fal+22]. The GFH is employed to compute a traffic plan that maximizes the objective given in Def. 5.9.

Here, we give a more detailed description of the GFH. Remember, GFH is an iterative greedy approach. It splits $\mathit{Active}\mathcal{F}(\mathcal{P})$ and $\mathit{Req}\mathcal{F}(\mathcal{P})$ into flow subsets and successively processes the different subsets in an order that is supposed to maximize the objective function (see Eq. (5.6)). For every flow set, GFH iteratively selects the next flow, computes ratings for the flow's configurations in $\boldsymbol{G}_c(\mathcal{P}')$, and adds the configuration with the best rating to an intermediary set of conflict-free configurations $\mathcal{C}$. During the GFH execution, $\mathcal{C}$ is always an independent vertex set in $\boldsymbol{G}_c(\mathcal{P}')$. For the same flow f, there may be multiple configurations in the final $\mathcal{C}$ returned by the GFH, but we can only include one of these (we need only one route and phase) in a traffic plan. In this case, we arbitrarily select one configuration for every flow with multiple configurations in $\mathcal{C}$ since all configurations in $\mathcal{C}$ are conflict-free.

Before explaining the greedy strategy and how configurations are rated, we (re-)introduce some terminology:

- $\mathcal{C}$ *admits* flow f if it contains at least one candidate configuration for f.
- A configuration $c \in \boldsymbol{G}_c(\mathcal{P}')$ is *shadowed* if at least one of its neighbors is included in $\mathcal{C}$. Thus, adding a configuration to $\mathcal{C}$ shadows all its neighbors. Shadowed configurations are excluded from being added to $\mathcal{C}$ since they conflict, that is, are connected by an edge, with at least one of the configurations in $\mathcal{C}$.
- A configuration is *solitary* if it has no neighbors in $c \in \boldsymbol{G}_c(\mathcal{P}')$ and thus cannot conflict with any other configuration.
- A configuration c is *eligible* for selection by the GFH algorithm if and only if c is neither shadowed nor $c \in \mathcal{C}$.

In each run, the GFH attempts to find a $\mathcal{C}$ that admits all flows as follows: First, $\mathcal{C}$ is initialized by adding all solitary configurations. Thus, $\mathcal{C}$ already admits all flows with solitary configurations without reducing the solution space. Then, configurations for the remaining flows are iteratively added to $\mathcal{C}$. We will discuss the configuration selection strategy in Appendix A.2.1 and the configuration rating in Appendix A.2.2. If $\mathcal{C}$ does not admit all flows after its first run, the GFH has a *re-run* mechanism which attempts to find a $\mathcal{C}$ that admits more flows by additional *re-runs* with modified starting conditions. The number of re-runs (default $n_{\text{re-runs}} = 3$) can be parameterized. We discuss the re-run technique in Appendix A.2.3. Finally, the complexity of GFH is briefly stated in Appendix A.2.4.

A.2.1 Configuration Selection Strategy

GFH being a greedy algorithm, the quality of the solution, that is, how many flows can ultimately be admitted, relies on the strategy we use to add configurations to $\mathcal{C}$. GFH uses a hierarchical, iterative strategy, first deciding on the next flow to process and then selecting a configuration for that flow. Remember that any active flow contributes more to the objective than all new flows taken together. To prevent that any configuration of a new flow shadows any active flow's configuration such that an active flow may not be admitted to $\mathcal{C}$, we thus first process all flows in $Active\mathcal{F}(\mathcal{P})$, before searching for a configuration for new flows from $Req\mathcal{F}(\mathcal{P})$.

Algorithm A.1: addConfigPerFlow

```
  input   : flows set $Search\mathcal{F}(\mathcal{P}')$, conflict-free configurations $\mathcal{C}$
1 $\forall f \in Search\mathcal{F}(\mathcal{P}')$ : check admits($f, \mathcal{C}$) ;
2 create heap of not admitted flows ;
3 while $heap \neq \emptyset$ do
4     $f_{\min} \leftarrow$ pop $f_{\min}$ with least eligible configurations from heap ;
5     $c_{\text{sel}} \leftarrow$ eligible $c$ of color $f_{\min}$ with smallest result from computeShadowRating($c$) ;
6     add $c_{\text{sel}}$ to $\mathcal{C}$, update eligibility of neighbors of $c_{\text{sel}}$;
7     update heap (remove completely shadowed flows and reorder) ;
8 end
9 return updated $\mathcal{C}$ ;
```

The pseudo-code to process a particular (sub-)set of flows, say $Search\mathcal{F}(\mathcal{P}') \subseteq Active\mathcal{F}(\mathcal{P}) \cup Req\mathcal{F}(\mathcal{P})$, is given in the method addConfigPerFlow in Alg. A.1. We start by creating a min-heap that contains all flows from $Search\mathcal{F}(\mathcal{P}')$ not already admitted by $\mathcal{C}$ (see Alg. A.1, line 1). The flows in the min-heap are sorted according to the number of remaining eligible configurations such that flows with less eligible configurations are on top of the heap. Ties are broken by the higher total degree of the configurations in $\text{cand}(f, \mathcal{P}')$, because a high total degree suggests a high chance that all configurations of the flow will soon become ineligible. The unique flow identifier is used as final tie-breaker to get a deterministic algorithm.

Next, the heap is iteratively processed: The flow on top of the heap—the one with the least remaining eligible configurations—is removed from the heap, and we add the flow's best-rated (see Appendix A.2.2) configuration to $\mathcal{C}$. When a configuration is added to $\mathcal{C}$, the heap is updated. Besides the flow $f_{\min}$ which gets admitted by adding c_{sel} to $\mathcal{C}$, we remove any flow which has become completely shadowed from the heap, too. In other words, in each iteration of the while-loop the heap contains only flows with more than one remaining eligible configuration, and—because we remove at least one flow from the heap in each iteration of the while-loop—addConfigPerFlow is guaranteed to terminate. The remaining flows in the heap might be reordered (see Alg. A.1, line 7) since the number of eligible configurations could have changed.

A.2.2 Rating Configurations

The *shadowRating* value of a configuration is intended to capture the "cost" of selecting that configuration in terms of the remaining solution space. For example, adding a configuration to $\mathcal{C}$ that shadows huge portions of the conflict graph is "expensive". Conversely, the cost is lower, the fewer neighbors are shadowed. If neighbors are shadowed, it is preferable to shadow configurations of those flows with lots of remaining eligible configurations. This intuition is encoded in the computeShadowRating pseudo-code given in Alg. A.2.

Algorithm A.2: computeShadowRating

```
   input : configuration $c$
1  shadowRating = 0 ;
2  $\mathcal{F}_{\text{neig}}$ ← set of flows of all neighbors of $c$ ;
3  foreach $f_n \in \mathcal{F}_{neig}$ do
4      shadowCount ← number of eligible configurations of flow $f_n$ in neighborhood of $c$ ;
5      eligibleCount ← total number of eligible configurations for $f_n$ ;
6      $\delta$ ← shadowCount / eligibleCount ;
7      if $\delta = 1$ then
8          shadowRating+=$\alpha$ ;
9      else
10         shadowRating+=$\delta$ ;
11     end
12 end
13 return shadowRating of $c$ ;
```

We compute for each flow f_n with a configuration in the 1-hop neighborhood of c the share δ of this flow's remaining eligible configurations that would be shadowed by picking c. Configurations that are already shadowed are not taken into account. The *shadowRating* ordinarily amounts to the sum over the δ values of c's neighborhood. If adding c to $\mathcal{C}$ shadows all remaining eligible configurations of a flow ($\delta = 1$), a very large constant α (default $\alpha = 1000$) is added instead to discourage the selection of a configuration c which shadows the remaining configuration(s) of another flow. Note that the method computeShadowRating can be executed in parallel since we only have to read the neighbors of c. Further, the *eligibleCount* calculation (see Alg. A.2, line 5) can be optimized via caching so that *eligibleCount* has to be determined only once for each execution of Alg. A.2.

A.2.3 Re-run Mechanism

The re-run mechanism provided by the GFH (see Alg. A.3) can be used to improve the number of flows admitted by $\mathcal{C}$ if $\mathcal{C}$ does not admit all flows after the first run. In principle, a single run in the GFH corresponds to executing the body of the loop in Alg. A.3 (starting at line 3) once. In each run, we call addConfigPerFlow once for every flow subset, using it as parameter $\mathit{Search}\mathcal{F}(\mathcal{P}')$.

Algorithm A.3: GFH

```
input   : active flow set $\mathit{Active}\mathcal{F}(\mathcal{P})$, new flow set $\mathit{Req}\mathcal{F}(\mathcal{P})$, $n_{\text{re-runs}}$
1  $\mathcal{C} \leftarrow \emptyset$ ;                                        // init. $\mathcal{C}$ (first run)
2  cache $\leftarrow \emptyset$ ;
3  for $i = 0; i \leq n_{\mathit{re\text{-}runs}}; i{+}{+}$ do
4      $\mathit{naActive}\mathcal{F} \leftarrow \mathit{Active}\mathcal{F}(\mathcal{P}) \setminus \mathcal{C}$ ;
5      $\mathit{aActive}\mathcal{F} \leftarrow \mathit{Active}\mathcal{F}(\mathcal{P}) \bigcap \mathcal{C}$ ;
6      $\mathit{naReq}\mathcal{F} \leftarrow \mathit{Req}\mathcal{F}(\mathcal{P}) \setminus \mathcal{C}$ ;
7      $\mathit{aReq}\mathcal{F} \leftarrow \mathit{Req}\mathcal{F}(\mathcal{P}) \bigcap \mathcal{C}$ ;
8      $\mathcal{C} \leftarrow \{v \in \mathcal{V} : \deg(v) = 0\}$ ;             // $\mathcal{C} \leftarrow$ solitary configurations
9      $\mathcal{C} \leftarrow$ addConfigPerFlow($\mathit{naActive}\mathcal{F}, \mathcal{C}$) ;
10     $\mathcal{C} \leftarrow$ addConfigPerFlow($\mathit{aActive}\mathcal{F}, \mathcal{C}$) ;
11     $\mathcal{C} \leftarrow$ addConfigPerFlow($\mathit{naReq}\mathcal{F}, \mathcal{C}$) ;
12     $\mathcal{C} \leftarrow$ addConfigPerFlow($\mathit{aReq}\mathcal{F}, \mathcal{C}$) ;
13     if $\mathcal{C}$ admits $(\mathit{Active}\mathcal{F}(\mathcal{P}) \cup \mathit{Req}\mathcal{F}(\mathcal{P}))$ then
14         return $\mathcal{C}$ ;
15     else
16         store $\mathcal{C}$ in cache ;
17     end
18 end
19 return $\text{argmax}_{\mathcal{C} \in \text{cache}}$ TrafficPlanningObjective($\mathcal{C}$) ;
```

Remember that the order of the flow subsets for which we execute addConfigPerFlow implicitly assigns priorities to those subsets. Due to the greedy strategy, the earlier we include a flow f from $\boldsymbol{G}_c(\mathcal{P}')$ in $\mathit{Search}\mathcal{F}(\mathcal{P}')$ the likelier it is that a configuration for f is added to $\mathcal{C}$. We take advantage of this in the re-run mechanism to prioritize those flows which were not admitted to $\mathcal{C}$ in the previous run. Note that we still have to account for the higher importance of active flows compared to new flows, see Eq. (5.6). To account for this, we split both, $\mathit{Active}\mathcal{F}(\mathcal{P})$ and $\mathit{Req}\mathcal{F}(\mathcal{P})$, into two subsets each: one subset contains all flows which were **a**dmitted by $\mathcal{C}$ in the previous run (prefixed by "a": $\mathit{aActive}\mathcal{F}$, $\mathit{aReq}\mathcal{F}$ in Alg. A.3), and another subset contains those flows which could **n**ot be **a**dmitted previously (prefixed by "na": $\mathit{naActive}\mathcal{F}$, $\mathit{naReq}\mathcal{F}$ in Alg. A.3). Before each run, we reset $\mathcal{C}$ (see Alg. A.3, line 8). Then, we first process the two subsets corresponding to $\mathit{Active}\mathcal{F}(\mathcal{P})$, before processing the two subsets corresponding to $\mathit{Req}\mathcal{F}(\mathcal{P})$ (see Alg. A.3 lines 9–12). To be exact, we call addConfigPerFlow on the four flow subsets in the following order: 1) flows from $\mathit{Active}\mathcal{F}(\mathcal{P})$ previously not admitted by $\mathcal{C}$, 2) flows from $\mathit{Active}\mathcal{F}(\mathcal{P})$ previously admitted by $\mathcal{C}$, 3) flows from $\mathit{Req}\mathcal{F}(\mathcal{P})$ previously not admitted by $\mathcal{C}$, and 4) flows from $\mathit{Req}\mathcal{F}(\mathcal{P})$ previously admitted by $\mathcal{C}$.

The first run constitutes a special case where the flows admitted by $\mathcal{C}$ are not the result of a previous run but contain only those flows with solitary configurations. Solitary configurations do not shadow any eligible configuration and consequently cannot affect the results during the re-runs. Therefore, by convention, we do not count the first run as a re-run. Hence, the loop can be executed up to $n_{\text{re-runs}} + 1$ times. Re-runs are performed either until $\mathcal{C}$ admits all flows, or we run out of re-runs (default $n_{\text{re-runs}} = 3$). This can improve the objective value, but it is not guaranteed. If we use up all re-runs, the GFH algorithm returns the $\mathcal{C}$ with the highest objective value seen so far.

A.2.4 Complexity

The worst-case runtime complexity of a single GFH run, that is, adding the solitary configurations to $\mathcal{C}$ and performing a single run of addConfigPerFlow for all subsets, can be stated as $\mathcal{O}(\mathcal{E}+\mathcal{V}\cdot\mathcal{F})$. Here, $\mathcal{E}$ is the number of conflicts, $\mathcal{V}$ is the number of configurations in the conflict graph, and $\mathcal{F}$ represents the number of flows.

The re-run mechanism adds another factor $r' = n_{\text{re-runs}} + 1$ resulting in a simplified overall complexity of $\mathcal{O}(r' \cdot \mathcal{E} + r' \cdot \mathcal{V}\mathcal{F})$.

For more details, refer to [Fal+22].

Bibliography

[Ata+19] A. Atallah, G. Bany Hamad, and O. Ait Mohamed. “Routing and Scheduling of Time-Triggered Traffic in Time Sensitive Networks.” In: *IEEE Transactions on Industrial Informatics* (2019), pp. 1–1. ISSN: 1941-0050. DOI: 10.1109/TII.2019.2950887.

[Bac+01] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity An Algebra for Discrete Event Systems*. Web. 2001. URL: https://www.rocq.inria.fr/metalau/cohen/documents/BCOQ-book.pdf.

[Bar+99] A.-L. Barabási and R. Albert. “Emergence of Scaling in Random Networks.” In: *Science* 286.5439 (1999), pp. 509–512. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.286.5439.509. pmid: 10521342.

[Bec+15] N. Becker and M. Fidler. “A Non-stationary Service Curve Model for Performance Analysis of Transient Phases.” In: *2015 27th International Teletraffic Congress*. 2015 27th International Teletraffic Congress. 2015, pp. 116–124. DOI: 10.1109/ITC.2015.21.

[Bec+19] N. Becker and M. Fidler. “A Non-Stationary Service Curve Model for Estimation of Cellular Sleep Scheduling.” In: *IEEE Transactions on Mobile Computing* 18.1 (2019), pp. 28–41. ISSN: 1558-0660. DOI: 10.1109/TMC.2018.2821133.

[Bel+16] P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gómez, and D. Salvagnin. “On Handling Indicator Constraints in Mixed Integer Programming.” In: *Computational Optimization and Applications* 65.3 (2016), pp. 545–566. ISSN: 0926-6003, 1573-2894. DOI: 10.1007/s10589-016-9847-8.

[Ber+01] G. Bernat, A. Burns, and A. Liamosi. “Weakly Hard Real-Time Systems.” In: *IEEE Transactions on Computers* 50.4 (2001), pp. 308–321. ISSN: 0018-9340. DOI: 10.1109/12.919277.

[Bez+17] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. “Julia: A Fresh Approach to Numerical Computing.” In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. eprint: https://doi.org/10.1137/141000671.

[Bon+14] S. Bondorf and J. B. Schmitt. "The DiscoDNC - A Comprehensive Tool for Deterministic Network Calculus." In: *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)*. 2014.

[Bou+01] J.-Y. L. Boudec and P. Thiran. *Network Calculus A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science 2050. Springer Berlin Heidelberg, 2001.

[Bou+09] A. Bouillard, L. Jouhet, and E. Thierry. *Service Curves in Network Calculus: Dos and Don'ts*. RR-7094. INRIA, 2009. URL: `https://hal.inria.fr/inria-00431674/document`.

[Bou+10] A. Bouillard, L. Jouhet, and É. Thierry. "Comparison of Different Classes of Service Curves in Network Calculus." In: *IFAC Proceedings Volumes*. 10th IFAC Workshop on Discrete Event Systems 43.12 (2010), pp. 306–311. ISSN: 1474-6670. DOI: `10.3182/20100830-3-DE-4013.00051`.

[Bou14] A. Bouillard. "Algorithms and Efficiency of Network Calculus." Habilitation à diriger des recherches. Ecole Normale Supérieure (Paris), 2014. URL: `https://hal.inria.fr/tel-01107384`.

[Bou96] J.-Y. L. Boudec. *Network Calculus Made Easy*. Technical Report EPFL-DI 96/218. ECOLE POLYTECHNIQUE FEDERALE, LAUSANNE (EPFL), 1996.

[Bou98] J.-Y. L. Boudec. "Application of Network Calculus to Guaranteed Service Networks." In: *IEEE Transactions on Information Theory* 44.3 (1998), pp. 1087–1096. ISSN: 0018-9448. DOI: `10.1109/18.669170`.

[Boy+16] M. Boyer, H. Daigmorte, N. Navet, and J. Migge. "Performance Impact of the Interactions between Time-Triggered and Rate-Constrained Transmissions in TTEthernet." In: *8th European Congress on Embedded Real Time Software and Systems*. TOULOUSE, France, 2016. URL: `https://hal.archives-ouvertes.fr/hal-01255939`.

[Bra+87] R. Braden and J. Postel. *Requirements for Internet Gateways*. Request for Comments RFC 1009. Internet Engineering Task Force, 1987. 54 pp. DOI: `10.17487/RFC1009`.

[Bra89] R. T. Braden. *Requirements for Internet Hosts - Communication Layers*. Request for Comments RFC 1122. Internet Engineering Task Force, 1989. 116 pp. DOI: `10.17487/RFC1122`.

[Bro+17] S. Bromberger, J. Fairbanks, and o. contributors. "JuliaGraphs/LightGraphs.Jl: An Optimized Graphs Package for the Julia Programming Language." In: (2017). DOI: `10.5281/zenodo.889971`.

[Cha+02] C. S. Chang, R. L. Cruz, J. Y. Le Boudec, and P. Thiran. “A Min, + System Theory for Constrained Traffic Regulation and Dynamic Service Guarantees.” In: *IEEE/ACM Transactions on Networking* 10.6 (2002), pp. 805–817. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.804824.

[Cha+99] C.-S. Chang and R. L. Cruz. “A Time Varying Filtering Theory for Constrained Traffic Regulation and Dynamic Service Guarantees.” In: *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM ’99)*. INFOCOM ’99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Vol. 3. 1999, pp. 63–70. DOI: 10.1109/INFCOM.1999.749253.

[Che+04] R. Chellappa, G. Qian, and Q. Zheng. “Vehicle Detection and Tracking Using Acoustic and Video Sensors.” In: *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*. 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing. Vol. 3. 2004, pp. III-793 –III-796. DOI: 10.1109/ICASSP.2004.1326664.

[Cra+16a] S. S. Craciunas and R. S. Oliver. “Combined Task- and Network-Level Scheduling for Distributed Time-Triggered Systems.” In: *Real-Time Systems* 52.2 (2016), pp. 161–200. ISSN: 0922-6443, 1573-1383. DOI: 10.1007/s11241-015-9244-x.

[Cra+16b] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner. “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. 2016, pp. 183–192. ISBN: 978-1-4503-4787-7. DOI: 10.1145/2997465.2997470.

[Cra+17] S. S. Craciunas and R. S. Oliver. *An Overview of Scheduling Mechanisms for Time-sensitive Networks*. École d’Été Temps Réel 2017. Paris, France, 2017, p. 7.

[Cru91] R. L. Cruz. “A Calculus for Network Delay. Part I.+II.” In: *IEEE Transactions on Information Theory* 37.1 (1991), pp. 114–141.

[Dan+13] M. Daniel-Cavalcante and R. Santos-Mendes. “Modular Methodology for the Network Calculus in a Time-Varying Context.” In: *IEEE Transactions on Information Theory* 59.10 (2013), pp. 6342–6356. ISSN: 0018-9448. DOI: 10.1109/TIT.2013.2272870.

[DAr+07] A. D’Ariano, M. Pranzo, and I. A. Hansen. “Conflict Resolution and Train Speed Coordination for Solving Real-Time Timetable Perturbations.” In: *IEEE Transactions on Intelligent Transportation Systems* 8.2 (2007), pp. 208–222. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2006.888605.

[De +14] J. A. R. De Azua and M. Boyer. “Complete Modelling of AVB in Network Calculus Framework.” In: *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*. RTNS ’14. New York, NY, USA: ACM, 2014, 55:55–55:64. ISBN: 978-1-4503-2727-5. DOI: 10.1145/2659787.2659810.

[Deu] Deutsche Forschungsgemeinschaft. *DFG - GEPRIS - Integrierte Reglerentwurfsverfahren Und Kommunikationsdienste Für Digital Vernetzte Regelungssysteme*. URL: https://gepris.dfg.de/gepris/projekt/285825138.

[Die+12] J. Diemer, D. Thiele, and R. Ernst. “Formal Worst-Case Timing Analysis of Ethernet Topologies with Strict-Priority and AVB Switching.” In: *7th IEEE International Symposium on Industrial Embedded Systems (SIES’12)*. 7th IEEE International Symposium on Industrial Embedded Systems (SIES’12). 2012, pp. 1–10. DOI: 10.1109/SIES.2012.6356564.

[Dju+07] P. Djukic and S. Valaee. “Link Scheduling for Minimum Delay in Spatial Re-Use TDMA.” In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications. 2007, pp. 28–36. DOI: 10.1109/INFCOM.2007.12.

[Dun+17] I. Dunning, J. Huchette, and M. Lubin. “JuMP: A Modeling Language for Mathematical Optimization.” In: *SIAM Review* 59.2 (2017), pp. 295–320. DOI: 10.1137/15M1020575.

[Dür+14] F. Dürr and T. Kohler. *Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches*. TR 2014/04. Stuttgart: Institute of Parallel and Distributed Systems, University of Stuttgart, 2014, p. 2014.

[Dür+16] F. Dürr and N. G. Nayak. “No-Wait Packet Scheduling for IEEE Time-sensitive Networks (TSN).” In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS ’16. Brest, France: ACM, 2016, pp. 203–212. ISBN: 978-1-4503-4787-7. DOI: 10.1145/2997465.2997494.

[Fal+18] J. Falk, F. Dürr, and K. 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. Hakodate, Japan, 2018, pp. 136–146. DOI: DOI10.1109/RTCSA.2018.00025.

[Fal+19a] J. Falk, F. Dürr, S. Linsenmayer, S. Wildhagen, B. Carabelli, and K. Rothermel. “Optimal Routing and Scheduling of Complemental Flows in Converged Networks.”

In: *Proceeding of the 27th International Conference on Real-Time Networks and Systems*. RTNS'19. Toulouse, France, 2019. DOI: 10.1145/3356401.3356415.

[Fal+19b] J. Falk, F. Dürr, and K. Rothermel. "Modeling Time-Triggered Service Intermittence in Network Calculus." In: *Proceedings of the 27th International Conference on Real-Time Networks and Systems* (Toulouse, France). RTNS '19. New York, NY, USA: ACM, 2019, pp. 90–100. ISBN: 978-1-4503-7223-7. DOI: 10.1145/3356401.3356411.

[Fal+19c] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel. "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++." In: *2019 International Conference on Networked Systems (NetSys)*. 2019 International Conference on Networked Systems (NetSys). 2019, pp. 1–8. DOI: 10.1109/NetSys.2019.8854500.

[Fal+20] J. Falk, F. Dürr, and K. Rothermel. "Time-Triggered Traffic Planning for Data Networks with Conflict Graphs." In: *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). Sydney, Australia: IEEE, 2020. DOI: 10.1109/RTAS48715.2020.00-12.

[Fal+21] J. Falk, H. Geppert, F. Dürr, S. Bhowmik, and K. Rothermel. *Dynamic QoS-Aware Traffic Planning for Time-Triggered Flows with Conflict Graphs*. Technical report computer science Technical Report No. 1. University of Stuttgart, Institute of Parallel and Distributed Systems, Distributed Systems: University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, 2021, p. 23. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=TR-2021-01&engl=1.

[Fal+22] J. Falk, H. Geppert, F. Dürr, S. Bhowmik, and K. Rothermel. "Dynamic QoS-Aware Traffic Planning for Time-Triggered Flows in the Real-Time Data Plane." In: *IEEE Transactions on Network and Service Management* 19.2 (2022), pp. 1807–1825. ISSN: 1932-4537. DOI: 10.1109/TNSM.2022.3150664.

[Fat+15] B. Fateh and M. Govindarasu. "Joint Scheduling of Tasks and Messages for Energy Minimization in Interference-Aware Real-Time Sensor Networks." In: *IEEE Transactions on Mobile Computing* 14.1 (2015), pp. 86–98. ISSN: 1536-1233. DOI: 10.1109/TMC.2013.81.

[Fid10] M. Fidler. "Survey of Deterministic and Stochastic Service Curve Models in the Network Calculus." In: *IEEE Communications Surveys Tutorials* 12.1 (2010), pp. 59–86. ISSN: 1553-877X. DOI: 10.1109/SURV.2010.020110.00019.

[Fis+85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. “Impossibility of Distributed Consensus with One Faulty Process.” In: *Journal of the ACM* 32.2 (1985), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121.

[Gaï+08] M. E. M. B. Gaïd, D. Simon, and O. Sename. “A Design Methodology for Weakly-Hard Real-Time Control.” In: 17th IFAC World Congress (IFAC WC 2008). 2008, p. 7. URL: https://hal.inria.fr/inria-00269209/document.

[Gar+79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* A Guide to the Theory of NP-completeness. New York, NY: Freeman, 1979. ISBN: 0-7167-1044-7.

[Gav+18] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop. “AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN.” In: *IEEE Access* 6 (2018), pp. 75229–75243. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2883644.

[Gol+07] N. Gollan and J. Schmitt. “Energy-Efficent TDMA Design Under Real-Time Constraints in Wireless Sensor Networks.” In: *Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems.* MASCOTS ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 80–87. ISBN: 978-1-4244-1854-1. DOI: 10.1109/MASCOTS.2007.23.

[Gur19] L. Gurobi Optimization. “Gurobi Optimizer Reference Manual.” In: (2019). URL: http://www.gurobi.com.

[Gur21] L. Gurobi Optimization. “Gurobi Optimizer Reference Manual.” In: (2021). URL: http://www.gurobi.com.

[Hag+08] A. A. Hagberg, D. A. Schult, and P. J. Swart. “Exploring Network Structure, Dynamics, and Function Using NetworkX.” In: *Proceedings of the 7th Python in Science Conference.* Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, 2008, pp. 11–15.

[Har+11] W. E. Hart, J.-P. Watson, and D. L. Woodruff. “Pyomo: Modeling and Solving Mathematical Programs in Python.” In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[Har+17] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola. *Pyomo–Optimization Modeling in Python.* 2nd ed. Vol. 67. Springer Science & Business Media, 2017.

[He+18] Q. He, D. Yuan, and A. Ephremides. “Optimal Link Scheduling for Age Minimization in Wireless Systems.” In: *IEEE Transactions on Information Theory* 64.7 (2018), pp. 5381–5394. ISSN: 0018-9448. DOI: 10.1109/TIT.2017.2746751.

[Hel+20a] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr. "On the Performance of Stream-based, Class-based Time-aware Shaping and Frame Preemption in TSN." In: *2020 IEEE International Conference on Industrial Technology (ICIT)*. 2020 IEEE International Conference on Industrial Technology (ICIT). 2020, pp. 298–303. DOI: 10.1109/ICIT45562.2020.9067122.

[Hel+20b] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr. "Scaling TSN Scheduling for Factory Automation Networks." In: *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*. 2020 16th IEEE International Conference on Factory Communication Systems (WFCS). 2020, pp. 1–8. DOI: 10.1109/WFCS47810.2020.9114415.

[Hel+21] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen. "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*. Ed. by ACM. Nantes,France: ACM (Online), 2021. DOI: 10.1145/3453417.3453421.

[Hsu+09] C.-C. Hsu, K.-F. Lai, C.-F. Chou, and K. C.-J. Lin. "ST-MAC: Spatial-Temporal MAC Scheduling for Underwater Sensor Networks." In: *IEEE INFOCOM 2009*. IEEE INFOCOM 2009. 2009, pp. 1827–1835. DOI: 10.1109/INFCOM.2009.5062103.

[IBM17] IBM Corporation. "IBM ILOG CPLEX Optimization Studio CPLEX User's Manual Version 12 Release 8." In: (2017), p. 596.

[IEE+18] IEEE Computer Society and IEEE Microwave Theory and Techniques Society. "IEEE Standard for Air Interface for Broadband Wireless Access Systems." In: *IEEE Std 802.16-2017 (Revision of IEEE Std 802.16-2012)* (2018), pp. 1–2726. DOI: 10.1109/IEEESTD.2018.8303870.

[IEE16] IEEE Computer Society. "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), pp. 1–57. DOI: 10.1109/IEEESTD.2016.7440741.

[IEE18a] IEEE Computer Society. "IEEE Standard for Ethernet." In: *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)* (2018), pp. 1–5600. DOI: 10.1109/IEEESTD.2018.8457469.

[IEE18b] IEEE Computer Society. "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks." In: *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), pp. 1–1993. DOI: 10.1109/IEEESTD.2018.8403927.

[IEE20a] IEEE Computer Society. "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), pp. 1–421. DOI: 10.1109/IEEESTD.2020.9121845.

[IEE20b] IEEE Instrumentation and Measurement Society. "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems." In: *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)* (2020), pp. 1–499. DOI: 10.1109/IEEESTD.2020.9120376.

[IEE21a] IEEE 802.1 Working Group. *IEC/IEEE 60802 TSN Profile for Industrial Automation |*. IEC/IEEE 60802 TSN Profile for Industrial Automation. 2021. URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[IEE21b] IEEE Computer Society. "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." In: *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), pp. 1–4379. DOI: 10.1109/IEEESTD.2021.9363693.

[Ind19] Industrial Internet Consortium. *Time Sensitive Networks for Flexible Manufacturing Testbed Characterization and Mapping of Converged Traffic Types.* 2019. URL: https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Char_Mapping_of_Converged_Traffic_Types_Whitepaper_20180328.pdf.

[ISO+96] ISO/IEC JTC 1, Information Technology and ITU-T. "ISO/IEC 7498-1:1994(E)." In: *ISO* (1996). URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/02/20269.html.

[Jam+97] S. Jamin, S. J. Shenker, and P. B. Danzig. "Comparison of Measurement-Based Admission Control Algorithms for Controlled-Load Service." In: *, Proceedings IEEE INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution.* , Proceedings IEEE INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution. Vol. 3. 1997, 973–980 vol.3. DOI: 10.1109/INFCOM.1997.631035.

[Jia+10] L. Jiang, D. Shah, J. Shin, and J. Walrand. "Distributed Random Access Algorithm: Scheduling and Congestion Control." In: *IEEE Transactions on Information Theory* 56.12 (2010), pp. 6182–6207. ISSN: 0018-9448, 1557-9654. DOI: 10.1109/TIT.2010.2081490.

[Jin+19] X. Jin, A. Saifullah, C. Lu, and P. Zeng. "Real-Time Scheduling for Event-Triggered and Time-Triggered Flows in Industrial Wireless Sensor-Actuator Networks." In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications.* IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. Paris, France: IEEE, 2019, pp. 1684–1692. ISBN: 978-1-72810-515-4. DOI: 10.1109/INFOCOM.2019.8737373.

[Kar+11] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. "Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions." In: *IEEE Communications Surveys Tutorials* 13.4 (Fourth 2011), pp. 584–616. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.061411.00019.

[Kha+14] D. D. Khanh and A. Mifdaoui. "Timing Analysis of TDMA-Based Networks Using Network Calculus and Integer Linear Programming." In: *2014 IEEE 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS).* Paris, France: IEEE, 2014, pp. 21–30. ISBN: 978-1-4799-5610-4. DOI: doi.ieeecomputersociety.org/10.1109/MASCOTS.2014.12.

[Koc04] T. Koch. "Rapid Mathematical Programming." Technische Universität Berlin, 2004. 217 pp. URL: https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/834.

[Kon+18] O. Kondrateva, H. Döbler, H. Sparka, A. Freimann, B. Scheuermann, and K. Schilling. "Throughput-Optimal Joint Routing and Scheduling for Low-Earth-Orbit Satellite Networks." In: *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS).* 2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS). 2018, pp. 59–66. DOI: 10.23919/WONS.2018.8311663.

[Kro+21] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng. "Joint Routing and Scheduling for Large-Scale Deterministic IP Networks." In: *Computer Communications* 165 (2021), pp. 33–42. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2020.10.016.

[KUK16] KUKA. *Hello Industrie 4.0 _we Connect You*. 2016. URL: https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kukaindustrie40en.pdf?rev=a1e19f08c4fc4f9ebacdb8ba748c1d46.

[Lau+16] S. M. Laursen, P. Pop, and W. Steiner. "Routing Optimization of AVB Streams in TSN Networks." In: *SIGBED Rev.* 13.4 (2016), pp. 43–48. ISSN: 1551-3688. DOI: 10.1145/3015037.3015044.

[Li+10] H. Li, Y. Cheng, C. Zhou, and P. Wan. "Multi-Dimensional Conflict Graph Based Computing for Optimal Capacity in MR-MC Wireless Networks." In: *2010 IEEE 30th International Conference on Distributed Computing Systems*. 2010 IEEE 30th International Conference on Distributed Computing Systems. 2010, pp. 774–783. DOI: 10.1109/ICDCS.2010.58.

[Li+19] Z. Li, H. Wan, Z. Pang, Q. Chen, Y. Deng, X. Zhao, Y. Gao, X. Song, and M. Gu. "An Enhanced Reconfiguration for Deterministic Transmission in Time-Triggered Networks." In: *IEEE/ACM Transactions on Networking* 27.3 (2019), pp. 1124–1137. ISSN: 1558-2566. DOI: 10.1109/TNET.2019.2911272.

[Lin+19] S. Linsenmayer, B. W. Carabelli, F. Dürr, J. Falk, F. Allgöwer, and K. Rothermel. "Integration of Communication Networks and Control Systems Using a Slotted Transmission Classification Model." In: *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC). 2019, pp. 1–6. DOI: 10.1109/CCNC.2019.8651811.

[Lub85] M. Luby. "A Simple Parallel Algorithm for the Maximal Independent Set Problem." In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (Providence, Rhode Island, USA). STOC '85. New York, NY, USA: ACM, 1985, pp. 1–10. ISBN: 978-0-89791-151-1. DOI: 10.1145/22145.22146.

[Mas+02] A. Mascis and D. Pacciarelli. "Job-Shop Scheduling with Blocking and No-Wait Constraints." In: *European Journal of Operational Research* (2002), p. 20.

[Nay+15] N. G. Nayak, F. Dürr, and K. Rothermel. "Software-Defined Environment for Reconfigurable Manufacturing Systems." In: *2015 5th International Conference on the Internet of Things (IOT)*. 2015 5th International Conference on the Internet of Things (IOT). 2015, pp. 122–129. DOI: 10.1109/IOT.2015.7356556.

[Nay+16] N. G. Nayak, F. Dürr, and K. 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. 2016, pp. 193–202. ISBN: 978-1-4503-4787-7. DOI: 10.1145/2997465.2997487.

[Nay+18a] N. G. Nayak, F. Dürr, and K. Rothermel. "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks." In: *IEEE Transactions on Industrial Informatics* 14.5 (2018), pp. 2066–2075. ISSN: 1941-0050. DOI: 10.1109/TII.2017.2782235.

[Nay+18b] N. G. Nayak, F. Dürr, and K. Rothermel. "Routing Algorithms for IEEE802.1Qbv Networks." In: *SIGBED Rev.* 15.3 (2018), pp. 13–18. ISSN: 1551-3688. DOI: 10.1145/3267419.3267421.

[Oli+18] R. S. Oliver, S. S. Craciunas, and W. 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 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). 2018, pp. 13–24. DOI: 10.1109/RTAS.2018.00008.

[Ope16] Open Networking Foundation. *SDN Architecture 1.1.* TR (Technical Reference). Palo Alto, CA 94303: Open Networking Foundation, 2016, p. 59. URL: https://opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf.

[Pah+18] M. Pahlevan and R. 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).* 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA). Vol. 1. 2018, pp. 337–344. DOI: 10.1109/ETFA.2018.8502515.

[Pah+19a] M. Pahlevan, J. Schmeck, and R. Obermaisser. "Evaluation of TSN Dynamic Configuration Model for Safety-Critical Applications." In: *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BD-Cloud/SocialCom/SustainCom).* 2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom). 2019, pp. 566–571. DOI: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00086.

[Pah+19b] M. Pahlevan, N. Tabassam, and R. Obermaisser. "Heuristic List Scheduler for Time Triggered Traffic in Time Sensitive Networks." In: *ACM SIGBED Review* 16.1 (2019), pp. 15–20. ISSN: 1551-3688, 1551-3688. DOI: 10.1145/3314206.3314208.

[Pal+13] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel. "On Optimal Scheduling in Duty-Cycled Industrial IoT Applications Using

IEEE802.15.4e TSCH." In: *IEEE Sensors Journal* 13.10 (2013), pp. 3655–3666. ISSN: 1530-437X, 1558-1748. DOI: 10.1109/JSEN.2013.2266417.

[Pan+21] Z. Pang, X. Huang, Z. Li, S. Zhang, Y. Xu, H. Wan, and X. Zhao. "Flow Scheduling for Conflict-Free Network Updates in Time-Sensitive Software-Defined Networks." In: *IEEE Transactions on Industrial Informatics* 17.3 (2021), pp. 1668–1678. ISSN: 1941-0050. DOI: 10.1109/TII.2020.2998224.

[Pei14] T. P. Peixoto. "The Graph-Tool Python Library." In: *figshare* (2014). DOI: 10.6084/m9.figshare.1164194.

[Pop+16] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner. "Design Optimisation of Cyber-Physical Distributed Systems Using IEEE Time-Sensitive Networks." In: *IET Cyber-Physical Systems: Theory Applications* 1.1 (2016), pp. 86–94. ISSN: 2398-3396. DOI: 10.1049/iet-cps.2016.0021.

[Poz+16] F. Pozo, G. Rodriguez-Navas, W. Steiner, and H. Hansson. "Period-Aware Segmented Synthesis of Schedules for Multi-hop Time-Triggered Networks." In: *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2016, pp. 170–175. DOI: 10.1109/RTCSA.2016.42.

[Pri+18] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl. "Dynamic Real-time Orchestration of I4.0 Components Based on Time-Sensitive Networking." In: *Procedia CIRP*. 51st CIRP Conference on Manufacturing Systems 72 (2018), pp. 910–915. ISSN: 2212-8271. DOI: 10.1016/j.procir.2018.03.174.

[Pri07] D. D. S. Price. "A General Theory of Bibliometric and Other Cumulative Advantage Processes." In: *Journal of the American Society for Information Science* 27.5 (2007), pp. 292–306. ISSN: 0002-8231. DOI: 10.1002/asi.4630270505.

[PRO+16] PROFIBUS Nutzerorganisation e. V. (PNO) and PROFIBUS & PROFINET International (PI). *PROFIBUS System Description - Technology and Application.* 2016. URL: https://www.profibus.com/index.php?eID=dumpFile&t=f&f=52380&token=4868812e468cd5e71d2a07c7b3da955b47a8e10d.

[Raa+17] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner. "Runtime Reconfiguration of Time-Sensitive Networking (TSN) Schedules for Fog Computing." In: *2017 IEEE Fog World Congress (FWC)*. 2017 IEEE Fog World Congress (FWC). 2017, pp. 1–6. DOI: 10.1109/FWC.2017.8368523.

[Rea] RealTime-at-Work (RTaW). *RTaW-Pegase Helps Design Safe and Optimized Critical Embedded Networks*. RealTime-at-Work (RTaW). URL: http://www.realtimeatwork.com/software/rtaw-pegase/.

[Rei+02] M. Reisslein, K. W. Ross, and S. Rajagopal. "A Framework for Guaranteeing Statistical QoS." In: *IEEE/ACM Transactions on Networking* 10.1 (2002), pp. 27–42. ISSN: 1063-6692. DOI: 10.1109/90.986511.

[Rei+12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. "Abstractions for Network Update." In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 323–334. ISBN: 978-1-4503-1419-0. DOI: 10.1145/2342356.2342427.

[Sad+20] O. Sadio, I. Ngom, and C. Lishou. "Design and Prototyping of a Software Defined Vehicular Networking." In: *IEEE Transactions on Vehicular Technology* 69.1 (2020), pp. 842–850. ISSN: 1939-9359. DOI: 10.1109/TVT.2019.2950426.

[Sch+07] H. Schioler, H. P. Schwefel, and M. B. Hansen. "CyNC: A MATLAB/SimuLink Toolbox for Network Calculus." In: *Proceedings of the 2Nd International Conference on Performance Evaluation Methodologies and Tools*. ValueTools '07. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, 60:1–60:10. ISBN: 978-963-9799-00-4. URL: http://dl.acm.org/citation.cfm?id=1345263.1345340.

[Sch+17] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl. "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. 2017, pp. 8–17. ISBN: 978-1-4503-5286-4. DOI: 10.1145/3139258.3139289.

[Sch+20] E. Schweissguth, D. Timmermann, H. Parzyjegla, P. Danielis, and G. Mühl. "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 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2020, pp. 1–11. DOI: 10.1109/RTCSA50079.2020.9203662.

[Smi+17] F. Smirnov, M. Glaß, F. Reimann, and J. Teich. "Optimizing Message Routing and Scheduling in Automotive Mixed-Criticality Time-Triggered Networks." In: *Proceedings of the 54th Annual Design Automation Conference 2017*. DAC '17. Austin, TX, USA: ACM, 2017, 48:1–48:6. ISBN: 978-1-4503-4927-7. DOI: 10.1145/3061639.3062298.

[Ste+09] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan. “TTEthernet Dataflow Concept.” In: *2009 Eighth IEEE International Symposium on Network Computing and Applications.* 2009 Eighth IEEE International Symposium on Network Computing and Applications (NCA). Cambridge, MA, USA: IEEE, 2009, pp. 319–322. DOI: 10.1109/NCA.2009.28.

[Ste+13] W. Steiner and B. Dutertre. “The TTEthernet Synchronisation Protocols and Their Formal Verification.” In: *International Journal of Critical Computer-Based Systems* 4.3 (2013), p. 280. ISSN: 1757-8779, 1757-8787. DOI: 10.1504/IJCCBS.2013.058398.

[Ste+15] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz. “Beware of the Hidden! How Cross-Traffic Affects Quality Assurances of Competing Real-Time Ethernet Standards for in-Car Communication.” In: *2015 IEEE 40th Conference on Local Computer Networks (LCN).* 2015 IEEE 40th Conference on Local Computer Networks (LCN). 2015, pp. 1–9. DOI: 10.1109/LCN.2015.7366277.

[Ste+18] W. Steiner, S. S. Craciunas, and R. S. Oliver. “Traffic Planning for Time-Sensitive Communication.” In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 42–47. ISSN: 2471-2825. DOI: 10.1109/MCOMSTD.2018.1700055.

[Ste10] W. Steiner. “An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks.” In: *2010 31st IEEE Real-Time Systems Symposium.* 2010 31st IEEE Real-Time Systems Symposium. 2010, pp. 375–384. DOI: 10.1109/RTSS.2010.25.

[Ste11] W. Steiner. “Synthesis of Static Communication Schedules for Mixed-Criticality Systems.” In: *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops.* 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops. 2011, pp. 11–18. DOI: 10.1109/ISORCW.2011.12.

[Sye+19] A. Syed and G. Fohler. “Efficient Offline Scheduling of Task-Sets with Complex Constraints on Large Distributed Time-Triggered Systems.” In: *Real-Time Systems* 55.2 (2019), pp. 209–247. ISSN: 1573-1383. DOI: 10.1007/s11241-018-9320-0.

[Tel+16] N. E. H. Tellache and M. Boudhar. “The Two-Machine Flow Shop Problem with Conflict Graphs.” In: *IFAC-PapersOnLine.* 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016 49.12 (2016), pp. 1026–1031. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2016.07.577.

[Wan+06a] E. Wandeler and L. Thiele. "Optimal TDMA Time Slot and Cycle Length Allocation for Hard Real-Time Systems." In: *Asia and South Pacific Conference on Design Automation, 2006.* Asia and South Pacific Conference on Design Automation, 2006. 2006, pp. 479–484. DOI: `10.1109/ASPDAC.2006.1594731`.

[Wan+06b] E. Wandeler and L. Thiele. *Real-Time Calculus (RTC) Toolbox.* 2006. URL: `http://www.mpa.ethz.ch/Rtctoolbox`.

[Wro97] J. Wroclawski <jtw@lcs.mit.edu>. *RFC 2211: Specification of the Controlled-Load Network Element Service.* 1997. URL: `https://tools.ietf.org/html/rfc2211`.

[Zha+17] L. Zhao, P. Pop, Q. Li, J. Chen, and H. Xiong. "Timing Analysis of Rate-Constrained Traffic in TTEthernet Using Network Calculus." In: *Real-Time Systems* 53.2 (2017), pp. 254–287. ISSN: 0922-6443, 1573-1383. DOI: `10.1007/s11241-016-9265-0`.

[Zha+18a] L. Zhao, P. Pop, and S. S. Craciunas. "Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus." In: *IEEE Access* 6 (2018), pp. 41803–41815. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2018.2858767`.

[Zha+18b] L. Zhao, P. Pop, Z. Zheng, and Q. Li. "Timing Analysis of AVB Traffic in TSN Networks Using Network Calculus." In: *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). 2018, pp. 25–36. DOI: `10.1109/RTAS.2018.00009`.

The End.

This document was compiled from LaTeX sources in October 2022.