Scheduling & Routing Time-triggered Traffic in Time-sensitive Networks

Von der Graduate School of Excellence advanced Manufacturing Engineering
der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

von

Naresh Ganesh Nayak

aus Kochi, Indien

Hauptberichter:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Mitberichter:	Prof. Dr.-Ing. Dirk Timmermann
Tag der mündlichen Prüfung:	08.11.2018

Graduate School of Excellence
advanced Manufacturing Engineering (GSaME)
der Universität Stuttgart
2018

ACKNOWLEDGEMENTS

As I sit down to write my doctoral thesis, I would like to acknowledge the support of the many people without whom the entire research presented in thesis would not have been possible. First and foremost, I would like to express my heartfelt gratitude to my doctoral supervisor, Prof. Dr. Kurt Rothermel, for providing me the opportunity to be a part of the Distributed Systems Group at the Institute of Parallel and Distributed Systems (IPVS), University of Stuttgart. He expressed his confidence in my abilities when I was myself plagued with several self-doubts. The many fruitful discussions I had with him were instrumental in the successful completion of this thesis. Apart from showing me the ropes to research, he was also always there for me as a fatherly figure to discuss several other problems that I encountered as a foreigner in Germany.

I would also like to thank Prof. Dr. Dirk Timmermann for agreeing to be a part of the examination commission and reviewing this thesis.

Next, I would like to thank Dr. Frank Dürr for the significant role that he played in my research. The various activities relating to software-defined networking and time-sensitive networking that Frank drove at the institute benefited me significantly. His unflinching support to my research ideas and approaches during tough times helped me hold my belief in the topic of this thesis. I really feel lucky to have had Frank around as my mentor.

I was also fortunate enough to have had a set of friendly and talented bunch of colleagues at IPVS. In particular, I would like to thank Thomas Kohler and Martin Brodbeck for helping me out with the networking infrastructure available at IPVS. Had it not been for them, I would have required several additional weeks to get my head around our SDN testbed. Special thanks to Ben Carabelli for translating the abstract of this thesis into German and being my go-to guy for issues arising in LaTeX. My sincere thanks to Jonathan Falk for helping me acquire a high proficiency in German language within a short timespan. I am sure it was annoying for him to help me with

the grammatical rules all the time. Thank you Sukanya Bhowmik for proof reading this manuscript and for being a good friend all this time. I would also like to thank Adnan Tariq, Zohaib Riaz, Florian Berg, David Schäfer, Thomas Bach, Christoph Dibak, Christian Mayer, Johannes Kässinger, Otto Bibartiu, Ahmad Slo, Saravana Palanisamy, Henriette Roeger, and Ruben Mayer for the friendly and pleasant environment during my time at IPVS. A big thank-you to Eva Strähle and Corinna Noltenius for helping me with all the administrative issues.

Finally, I would like to thank all my family members for their support in this journey. I dedicate this thesis to my late father Ganesh Nayak. I guess, I can't thank my mother, Preetha Nayak, enough for the sacrifices she made for me. Without her support, it would not have been possible for me to even think about writing a doctoral thesis. Special thanks to my wife Sowmya Shenoy and my sisters Neha and Rekha Nayak for putting up with my tantrums during this time. Thank you Rahul Pai and Aiswarya Pai for being my only family in Stuttgart and for hosting me frequently over weekends. Last but not least, I would like to thank my uncle Viswanath Kini for the significant role he has played in my life. He has not only inspired me to strive for excellence but also to be a good human being. I am indeed blessed to be a part of this family.

I would like to acknowledge the financial support from the German Research Foundation (DFG) through their grants to the Graduate School of Excellence - advanced Manufacturing Engineering (GSaME) at University of Stuttgart.

ABSTRACT

The application of recent advances in computing, cognitive and networking technologies in manufacturing has triggered the so-called fourth industrial revolution, also referred to as Industry 4.0. Smart and flexible manufacturing systems are being conceived as a part of the Industry 4.0 initiative to meet the challenging requirements of the modern day manufacturers, e.g., production batch sizes of one. The information and communication technologies (ICT) infrastructure in such smart factories is expected to host heterogeneous applications ranging from the time-sensitive cyber-physical systems regulating physical processes in the manufacturing shopfloor to the soft real-time analytics applications predicting anomalies in the assembly line. Given the diverse demands of the applications, a single converged network providing different levels of communication guarantees to the applications based on their requirements is desired.

Ethernet, on account of its ubiquity and its steadily growing performance along with shrinking costs, has emerged as a popular choice as a converged network. However, Ethernet networks, primarily designed for best-effort communication services, cannot provide strict guarantees like bounded end-to-end latency and jitter for real-time traffic without additional enhancements. Two major standardization bodies, viz., the IEEE Time-sensitive Networking (TSN) Task Group (TG) and the IETF Deterministic Networking (DetNets) Working Group are striving towards equipping Ethernet networks with mechanisms that would enable it to support different classes of real-time traffic. In this thesis, we focus on handling the time-triggered traffic (primarily periodic in nature) stemming from the hard real-time cyber-physical systems embedded in the manufacturing shopfloor over Ethernet networks. The basic approach for this is to schedule the transmissions of the time-triggered data streams appropriately through the network and ensure that the allocated schedules are adhered with. This approach leverages the possibility to precisely synchronize the clocks of the network participants, i.e., end systems and switches, using time synchronization protocols like the IEEE 1588 Precision Time Protocol (PTP). Based on the capabilities of the network participants,

the responsibility of enforcing these schedules can be distributed. An important point to note is that the network utilization with respect to the time-triggered data streams depends on the computed schedules. Furthermore, the routing of the time-triggered data streams also influences the computed transmission schedules, and thus, affects the network utilization. The question however remains as to how to compute transmission schedules for time-triggered data streams along with their routes so that an optimal network utilization can be achieved.

We explore, in this thesis, the scheduling and routing problems with respect to the time-triggered data streams in Ethernet networks. The recently published IEEE 802.1Qbv standard from the TSN-TG provides programmable gating mechanisms for the switches enabling them to schedule transmissions. Meanwhile, the extensions specified in the IEEE 802.1Qca standard or the primitives provided by OpenFlow, the popular south-bound software-defined networking (SDN) protocol, can be used for gaining an explicit control over the routing of the data streams. Using these mechanisms, the responsibility of enforcing transmission schedules can be taken over by the end systems as well as the switches in the network. Alternatively, the scheduling can be enforced only by the end systems or only by the switches. Furthermore, routing alone can also be used to isolate time-triggered data streams, and thus, bound the latency and jitter experienced by the data streams in absence of synchronized clocks in the network.

For each of the aforementioned cases, we formulate the scheduling and routing problem using Integer Linear Programming (ILP) for static as well as dynamic scenarios. The static scenario deals with the computation of schedules and routes for time-triggered data streams with *a priori* knowledge of their specifications. Here, we focus on computing schedules and routes that are optimal with respect to the network utilization. Given that the scheduling problems in the static setting have a high time-complexity, we also present efficient heuristics to approximate the optimal solution. With the dynamic scheduling problem, we address the modifications to the computed transmission schedules for adding further or removing already scheduled time-triggered data streams. Here, the focus lies on reducing the runtime of the scheduling and routing algorithms, and thus, have lower set-up times for adding new data streams into the network.

DEUTSCHE ZUSAMMENFASSUNG

Die Anwendung jüngster Fortschritte der Rechner- und Netzwerktechnologien in der Fertigung hat die sogenannte vierte industrielle Revolution eingeläutet, die auch als Industrie 4.0 bezeichnet wird. Im Rahmen dieser Initiative werden derzeit intelligente und flexible Fertigungssysteme erdacht, die den anspruchsvollen Anforderungen heutiger Fertigungsunternehmen gerecht werden sollen, wie beispielsweise der Produktion von Chargen der Größe eins. Die Informations- und Kommunikationsinfrastruktur (engl. *information and communication technologies*, ICT) in solchen intelligenten Fabriken muss dafür heterogene Anwendungen unterstützen, die von echtzeitkritischen cyber-physikalischen Systemen für die Maschinenregelung in der Fertigung bis zu Datenanalyseanwendungen mit weichen Echtzeitanforderungen wie z.B. für die Erkennung von Anomalien im Produktionsprozess reichen. Angesichts der verschiedenen Anforderungen dieser Anwendungen ist ein einheitliches Netzwerk wünschenswert, das den Anwendungen unterschiedliche Grade an Kommunikationsgarantien entsprechend ihres Bedarfs liefert.

Ethernet hat sich aufgrund seiner weiten Verbreitung, stetig steigenden Performance und sinkenden Kosten als Technologie der Wahl für vereinheitlichte Netzwerke hervorgetan. Allerdings können Ethernet-Netzwerke, die in erster Linie für Best-Effort-Kommunikationsdienste entwickelt wurden, zunächst nicht ohne weiteres harte Garantien wie Begrenzungen der Ende-zu-Ende-Latenz und des Jitter für Echtzeitkommunikation bieten. Jedoch arbeiten bereits zwei bedeutende Standardisierungsgruppen an entsprechenden Erweiterungen für Ethernet, welche Mechanismen zur Unterstützung unterschiedlicher Klassen von Echtzeitverkehr vorsehen, nämlich die IEEE Time-sensitive Networking (TSN) Task Group (TG) sowie die IETF Deterministic Networking Working Group. In dieser Dissertation konzentrieren wir uns auf den – vorwiegend periodischen – zeitgesteuerten (engl. *time-triggered*) Datenverkehr in Ethernet-Netzwerken, der beispielsweise von den cyber-physischen Echtzeitsystemen in der Fertigung erzeugt wird. Der grundlegende Ansatz hierzu umfasst die Berechnung

eines geeigneten Übertragungszeitplans, oder *Schedules*, für zeitgesteuerte Datenströme durch das Netzwerk und Mechanismen zur Gewährleistung, dass dieser vordefinierte Plan eingehalten wird. Unser Ansatz macht von der Möglichkeit Gebrauch, die Uhren aller Kommunikationsteilnehmer, also der Endsysteme und Switches, mithilfe von Protokollen wie dem IEEE 1588 Precision Time Protocol (PTP) präzise zu synchronisieren. Die Zuständigkeit für die Einhaltung des Schedules kann auf die Kommunikationsteilnehmer entsprechend ihrer Fähigkeiten verteilt werden. Hierbei ist es wichtig anzumerken, dass die Netzauslastung bezüglich der zeitgesteuerten Datenströme wesentlich vom Scheduling abhängt. Darüber hinaus hat das Routing des zeitgesteuerten Verkehrs einen Einfluss auf die möglichen Schedules, und wirkt sich daher ebenfalls auf die erreichbare Netzauslastung aus. Dadurch ergibt sich die Frage, wie die Schedules für zeitgesteuerte Datenströme zusammen mit deren Routen derart berechnet werden können, dass eine optimale Netzauslastung erzielt wird.

In dieser Dissertation untersuchen wir Scheduling- und Routingprobleme für zeitgesteuerte Datenströme in Ethernet-Netzwerken. Der kürzlich veröffentlichte IEEE-Standard 802.1Qbv der TSN-TG stellt programmierbare Gating-Mechanismen zur Verfügung, die Ethernet-Switches eine zeitlich geplante Paketübertragung erlauben. Gleichermaßen können die im IEEE-Standard 802.1Qca spezifizierten Erweiterungen oder auch die Primitiven des weit verbreiteten software-defined networking (SDN) Protokolls OpenFlow dazu verwendet werden, explizit die Kontrolle über das Routing von Datenströmen zu übernehmen. Unter Verwendung dieser Mechanismen kann die Zuständigkeit für die Durchsetzung von Übertragungs-Schedules von Endsystemen sowie Switches im Netzwerk übernommen werden. Alternativ kann das Scheduling auch rein durch Endsysteme oder rein durch Switches vollzogen werden. Außerdem kann schon allein durch das Routing eine Isolation von zeitgesteuerten Datenströmen erreicht werden, und dadurch eine Beschränkung der Latenz und des Jitters dieser Datenströme, selbst ohne den Einsatz präzise synchronisierter Uhren im Netzwerk.

Für jeden der zuvor genannten Fälle formulieren wir das Scheduling- und Routingproblem als ganzzahliges lineares Optimierungsproblem (engl. *integer linear program*, ILP) für statische sowie dynamische Szenarien. Das statische Szenario umfasst die Berechnung von Schedules und Routen für eine Menge zeitgesteuerter Datenströme, deren Spezifikationen *a priori* bekannt sind. In diesem Fall konzentrieren wir uns auf die Berechnung von Schedules und Routen welche zu einer optimalen Netzauslastung führen. Da die entwickelten Algorithmen zur Lösung des statischen Schedulingproblems durch eine hohe Zeitkomplexität gekennzeichnet sind, stellen wir außerdem effiziente Heuristiken zur Approximation der optimalen Lösung vor. Beim dynamischen Schedulingproblem befassen wir uns hingegen mit der Modifikation bereits berechneter Schedules beim Hinzufügen neuer oder beim Entfernen bestehender zeitgesteuerter Datenströme. Dabei wird eine Laufzeitreduktion der Scheduling- und Routingalgorithmen angestrebt, und damit eine Verkürzung des Verbindungsaufbaus für neu hinzukommende Datenströme.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

1.1 Motivation

The deployment of cyber-physical systems and automation technologies to implement production processes in a shopfloor has revolutionized manufacturing. These systems enable control of production processes by means of sensors and actuators embedded in the manufacturing shopfloor and networked over an information and communication technologies (ICT) infrastructure. Notably, they have triggered a paradigm shift from mass production to production of highly customized products. For instance, Daimler, a leading automotive manufacturer, offers today around 10^{21} and 10^{24} variants of its Mercedes C-Class and E-Class models, respectively [1]. These technologies have also facilitated manufacturers to offer a wider range of products in their portfolio leading to a subsequent decrease in the duration of their product life-cycles. An example from the automotive industry is the reduction of the average duration of product life-cycles of street cars from 10.6 years in the 1970s to 5.6 years in the early 2000s [2]. Given that modern day manufacturers are striving to achieve a production batch size of one along with rolling product releases, these trends are only expected to rise sharply in the coming years. New manufacturing systems are being conceptualized in order to achieve these goals at costs not significantly higher than that of mass production [3] [4].

The advent of Internet of Things (IoT) has led to the networking of sensing, actuating, and computing elements in the manufacturing shopfloor over the internet, bringing in key enabling technologies like Machine Learning, Artificial Intelligence, Data Analytics, etc., into the shopfloor [5] [6] [7]. Thanks to these technologies, we are standing on the cusp of the so-called fourth industrial revolution, also labelled as *Industry 4.0* in popular literature [8]. Industry 4.0 envisions intelligent manufacturing environments with

self-adaptive machines which reconfigure themselves to efficiently meet volatile production goals stemming from changing market demands, regulations, rapid innovations, etc., [9].

The ICT infrastructures in such a "smart factory", usually consisting of computing, storage, and network resources, are expected to host a number of heterogeneous applications, each demanding different levels of service guarantees from the underlying infrastructure. On the one hand are extremely time-sensitive cyber-physical systems where network delay (including the delay from packet losses) and jitter impacts the quality of control of cyber-physical systems. An example for such a system are the machines in an automotive shopfloor which can potentially fail when two consecutive packets are lost [10]. In extreme cases these failures could lead to a severe damage to the machines or can even be fatal. Another example of such time-sensitive systems are the isochronous motion control appliances in industrial automation which demand bounded jitter to the order of microseconds for stability [11]. On the other hand are soft real-time applications, for instance, an industrial analytics application using machine learning techniques for optimizing production output [12], or complex event processing (CEP) applications detecting anomalies in the production line [7] [13].

Providing required quality of service (QoS) to the heterogeneous applications executing in the ICT infrastructure is typically easier, if dedicated resources are allocated for different kind of applications, e.g., deployment of dedicated machines with real-time operating systems and highly engineered field-bus networks providing computing and communication capabilities respectively for the time-sensitive cyber-physical systems hosted in the infrastructure. The modern automation pyramid which classifies the infrastructure resources into several levels ranging from field level to enterprise level based on their functionalities and capabilities is a classical example of such dedicated resource provisioning. However, the aspect of reconfigurability in smart factories cannot be limited to the mechanical components alone and must also cover its ICT infrastructure which ultimately hosts the applications executing on top. Hence, dedicated resource provisioning schemes would not be suitable for deployment in smart factories. A converged ICT infrastructure which provisions resources to the applications based on their requirements is needed to handle such heterogeneity.

To achieve converged ICT infrastructures targeted at manufacturing scenarios, we propose a software-defined approach inspired from the basic principles of software-defined networking (SDN). These principles can be further applied to the compute and storage domains of the infrastructure to create a software-defined manufacturing environment.

1.1.1 Software-defined Manufacturing Environment

The paradigm of software-defined networking primarily aims to increase the flexibility of networking with two basic operating principles [14]. Firstly, it clearly separates the

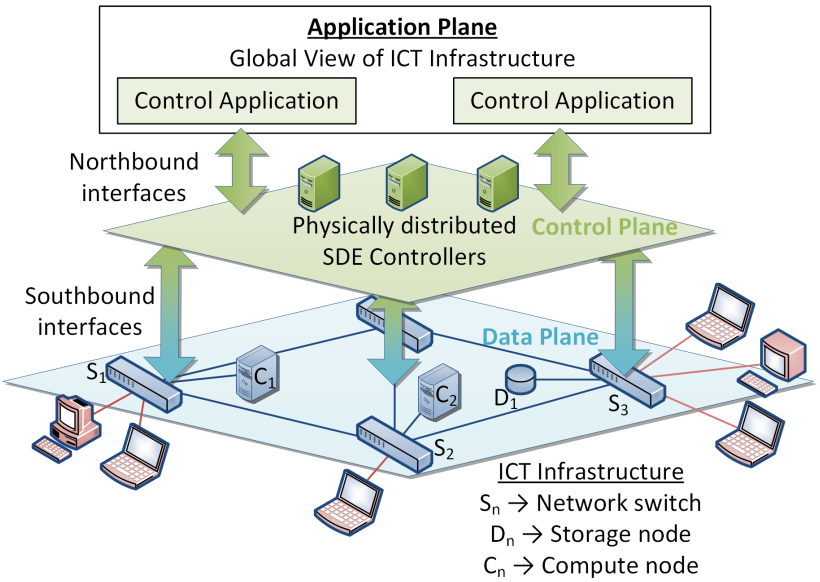


Figure 1.1: Software-defined Manufacturing Environment

basic network functionality of forwarding (network data plane) from network configuration and management (network control plane). The network data plane is implemented "in hardware" by network switches, while the network control plane is outsourced to standard hosts, i.e., SDN controller(s), implementing the logic to configure the network, e.g., by populating the forwarding tables of switches. Secondly, SDN enables a logically centralized control plane which has a global view onto the underlying network infrastructure simplifying the implementation of control logic significantly. For instance, SDN enables deployment of centralized routing algorithms with a global view on the network topology, traffic statistics, etc., to improve network utilization instead of implementing a distributed routing algorithm [15]. The logical centralization here does not imply that the control plane is also physically centralized. Usually, it is distributed to several hosts to increase availability, performance, and fault tolerance. With SDN, the network can be shaped, if required dynamically, to meet the requirements of the applications executing in the infrastructure with respect to communication.

These principles can be translated from the networking domain to compute and storage domains by mainly separating their core functionality from their management and configuration. For instance, the computational processes executed by virtual machines can be seen as a typical task of the data plane, while the management of virtual machines (creation, migration, placement, termination, etc.) would be handled by the control plane. Figure 1.1 shows the architecture of such a software-defined manufacturing environment.

Architecturally, the software-defined manufacturing environment consists of three layers, viz., the data plane, the control plane, and the application plane. The data plane in a software-defined manufacturing environment (SDME) consists not only of the stor-

age nodes, the compute nodes, and the network switches, but also of the sensors and the actuators embedded in the manufacturing shopfloor. The control plane is hosted on physically distributed controllers but maintains a logically centralized view on the data plane, while the applications executing in the infrastructure are on the application plane. The control plane communicates with the data plane and the application plane using what are known as the southbound and the northbound interfaces, respectively. The control plane gathers applications requirements using northbound interfaces and provisions the infrastructure resources to the applications based on its holistic view of the data plane and the application requirements. The data plane is then configured by the control plane using the southbound interfaces to enforce the provisioning of resources. For instance, if new sensors and actuators are plugged-in into the infrastructure for creating a new cyber-physical system, the control plane allocates required computational resources for the cyber-physical systems controller (different from the controllers hosting the control plane of the infrastructure) and configures the network to connect the respective computing node with the sensors and the actuators while ensuring that the latency and jitter requirements of the cyber-physical systems are satisfied.

Although this concept in general improves flexibility, it is still an open question on how to provide desired QoS guarantees to the applications, i.e., how exactly to configure the data plane for satisfying the real-time requirements of all the applications while exploiting the global view of the infrastructure available with the software-defined approach. In this thesis, we focus on the networking domain of the software-defined manufacturing environment and explore the various challenges in configuring the underlying communication network in a software-defined manufacturing environment based on the applications executing in the infrastructure. We exploit the latest developments in the networking technologies—Software-defined Networking and Time-sensitive Networking [16]—for managing and controlling complex networks foreseen in smart factories. Overall, we seek to develop a "Time-sensitive Software-defined Network", a network architecture along with required algorithms, that automatically configures itself based on the capabilities of the data plane elements of the underlying network and the requirements of the hosted applications. Such an architecture can become one cornerstone for a software-defined manufacturing environment and can be complemented by additional concepts for the compute and the storage domains in the future.

1.1.2 The Time-sensitive Networking Initiative

As already mentioned, the cyber-physical systems deployed in manufacturing shopfloors are hard real-time systems demanding guarantees like upper bounds on the network latency and latency variance (also referred to as jitter). For instance, computerised numerical control (CNC) machines used in production processes like milling, sawing, drilling etc., demand network latencies to the order of 250 μs–1 ms with jitter limited to a few microseconds [17]. Traditionally, highly engineered field-bus systems

have been used to provide these stringent communication guarantees for such time-sensitive systems [18] [19]. State-of-the-art field-bus technologies like SERCOS-III [20], ProfiNET [21], etc., developed over Ethernet, can handle up to three different classes of real-time traffic along with non-real-time traffic over the same medium [11]. However, being incompatible with each other, different field-bus technologies cannot be operated on the same physical medium without losing real-time properties. This hinders the goal of realising a single converged network as required by the smart factories instead of the multitude of networks classified into multiple levels in the current model of the automation pyramid.

The proliferation of the IEEE 802.3, i.e., Ethernet, and IP networks, and its steadily growing performance along with shrinking costs has led to the emergence of these networking technologies as a natural choice for industrial automation as well. However, these technologies, initially designed for providing best-effort communication services, cannot provide required real-time guarantees for time-sensitive traffic like the time-triggered data streams (primarily periodic in nature) stemming from manufacturing systems like the CNC machines in a shopfloor [22]. Realizing the need for equipping these networking technologies to also handle real-time traffic in addition to best-effort traffic, two major standardization bodies—the IEEE and the IETF—have set out to standardize extensions enabling the usage of Ethernet as a converged network. The IEEE Time-sensitive Networking (TSN) Task Group (TG) [16] is developing standards for time-synchronized low latency streaming services for Ethernet networks, while the IETF Deterministic Networking (DetNets) Working Group [23] is targeting time-sensitive communication over Layer 3 routed networks. Both these groups consider a separation of concerns and logical centralization of control logic, like in Software-defined Networking, a promising option [24] [25].

The TSN-TG has already published several extensions to the IEEE 802.1 standard and, at the time of writing this thesis, was actively working on several others. Prominent among the recently published standards are the ones which standardize hardware extensions in Ethernet switches to handle scheduled traffic, i.e., the time-triggered data streams. The IEEE 802.1Qbv [26] specifies a programmable gating mechanism which regulates the connection of the queues of an egress port with the physical medium of the port and can be used for scheduling transmissions of data streams at the egress ports of a switch. The IEEE 802.1Qca [27] provides mechanisms for explicit control over the forwarding of traffic, if required over non-shortest paths. Furthermore, the availability of frame pre-emption mechanisms enable high priority traffic to commence by interrupting transmissions of low priority traffic. The availability of these capabilities in the data plane elements opens up additional possibilities with respect to configuration of the network to handle time-triggered data streams. However, these standards stop short of specifying concrete methods to compute schedules or routes for time-triggered traffic, providing network operators the complete freedom to choose/develop their own approaches to these scheduling and routing problems.

For optimal utilization of the network by the applications relying on time-triggered communication, there is a need of algorithms for computing good schedules and routes for time-triggered data streams. In this thesis, we formulate and solve the problem of scheduling and routing of time-triggered data streams based on the availability of these aforementioned IEEE extensions in Ethernet networks.

1.2 Scheduling & Routing of Time-triggered Traffic in Ethernet

Network delay in Switched-Ethernet networks comprises propagation delay, processing delay, transmission delay, and queuing delay. Propagation delay in a local area network with predefined diameter is bounded by physics, thus, deterministic and very small (order of nanoseconds). The processing delays in commodity switches are in the order of microseconds or lower and have been shown to be almost constant for a given set of matching fields [28]. The transmission delay depends on the bit-rate of the link and is bounded and deterministic for constant bit-rate traffic. Thus, the bounding of queuing delays, which is often non-deterministic and could be unbounded, is the key to achieving deterministic and bounded network delays and jitter for real-time traffic. Furthermore, bounding of queuing delays eliminate packet losses occurring due to overflowing queues, provided that the bounds are low enough.

The basic approach to bound queuing delays targeting time-sensitive periodic communication, e.g., a constant bit-rate sensor data stream, in local area networks (LAN) is to schedule the transmission of the packets through the network. Naturally, these schedules must be effectively enforced by ensuring that the scheduled data streams (also referred to as time-triggered data streams) are isolated either temporally or spatially from each other and from other interfering traffic in the network. This idea leverages the possibility to precisely synchronize the clocks of the network participants, i.e., the end systems (hosts) and the network elements (switches) using time synchronization protocols like the IEEE 1588 Precision Time Protocol (PTP) or the IEEE 802.1AS (Standard for Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks) [29].

Based on the capabilities of the network participants, the scheduling can be enforced in different parts of the network [30]. The hosts require synchronized clocks for enforcing the schedules for time-triggered traffic, while the switches additionally need primitives like the enhancements specified in the IEEE 802.1Qbv standard for handling scheduled traffic in Ethernet. The scheduling may thus be implemented only on the switches, only on the hosts, or on the switches as well as the hosts. Enforcing scheduling in an increased number of participants, ideally on the switches as well as on the hosts, yields higher network utilization with respect to the scheduled traffic while providing tighter bounds on the end-to-end latencies and jitter experienced by the packets of

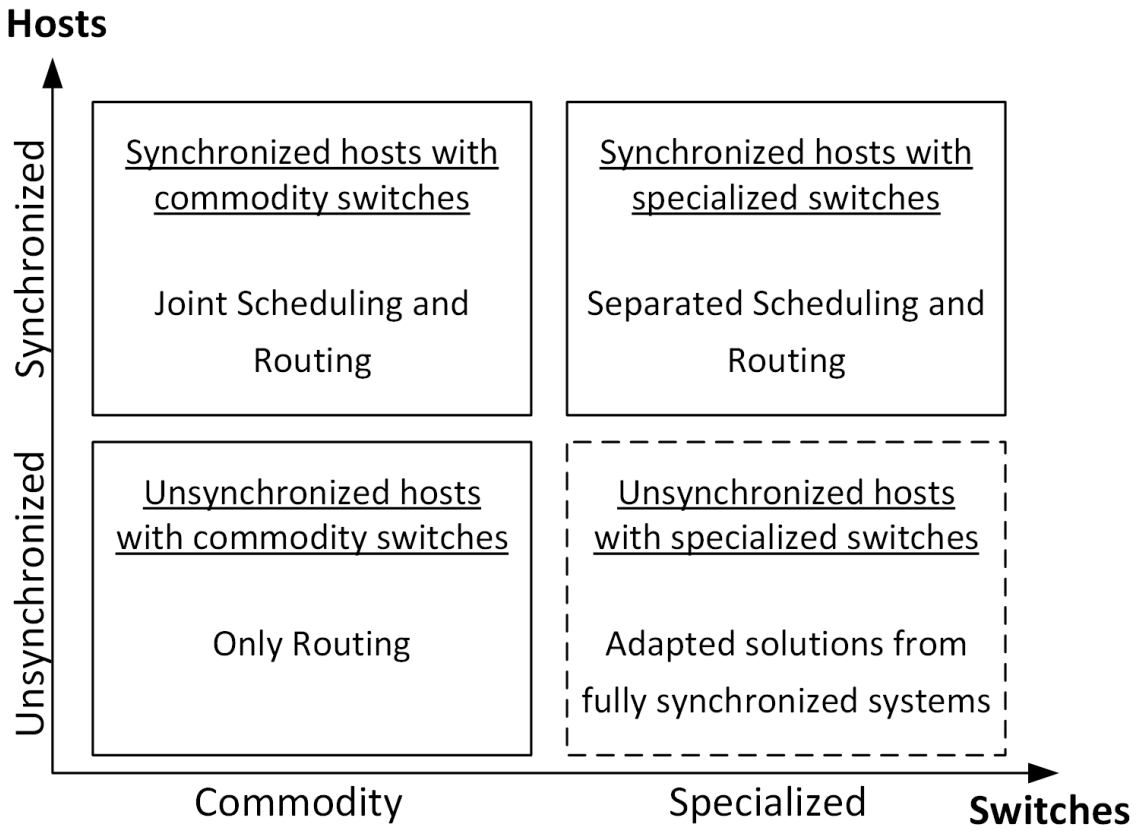


Figure 1.2: Solution space for handling time-triggered data streams in Ethernet

the corresponding data streams. However, this also incurs additional costs on account of the specialized hardware features required to be implemented in the switches. In networks where these enhancements are not available in the switches, the scheduling can be enforced only on the hosts.

One of the main prerequisites for computing transmission schedules for time-triggered data streams is the knowledge of the paths over which the streams are routed. However, the routes of the streams also affect the computed schedules. In the worst case, the paths over which the data streams are routed may not yield any feasible solution. Thus, the routing of time-triggered data streams is as important as its scheduling. Ideally, it would be preferable to compute the schedules and the routes of the time-triggered data streams jointly in a bid to compute solutions optimal with respect to the network utilization. However, as we will see in the later chapters, this is not always possible given the high time-complexities of both these problems.

We classify the solution space for handling of time-triggered data streams in Ethernet into four cases based on the capabilities of the hosts and switches, as shown in the Figure 1.2. First, we consider the networks in which the end systems as well as switches are synchronized and the switches are equipped with enhancements to enforce

transmission schedules for the time-triggered traffic. Here, transmission scheduling can be enforced at every hop resulting in tight bounds on the end-to-end latencies and the jitter experienced by the time-triggered data streams. The need for specialized switches, however, drives up the incurred costs. For the second case, we relax the requirement for synchronized clocks on the end systems. Such end systems could be low cost sensors which are not equipped with network interfaces capable of time-stamping incoming data packets, an essential requirement for precise clock synchronization. With unsynchronized clocks on the end systems, the transmissions can be scheduled only on the switches. Naturally, the offered bounds on the latency and jitter are also slightly relaxed.

Next, we consider the networks where only end systems are capable of scheduling transmissions while the switches are used only for forwarding. Such situations can be foreseen, for instance, during the incremental introduction of switches with additional enhancements for handling scheduled data streams in a live network. Such scenarios could also arise for networks in infrastructures where the quantum of scheduled traffic to be supported is too small a percentage of the total traffic to justify the additional costs incurred for specialized switches.

Finally, we also consider networks in which neither are any network participants synchronized nor are the switches equipped with extensions to schedule transmissions. Here, scheduling as an approach to provide communication guarantees is not an option. The only possibility here is to route the data streams for spatially isolating them in order to bound the queuing delays they incur.

1.2.1 Challenges

While there exists different possibilities to enforce transmission schedules for time-triggered traffic, developing suitable scheduling and routing algorithms for the purpose is rather challenging.

Constrained Optimization Problems

The scheduling problems in each of the aforementioned scenario are, in fact, constrained optimization problems and cannot be solved trivially. The transmission schedules for time-triggered traffic must be so constructed that they fulfil their respective end-to-end latency constraints and respect the network limitations/properties, e.g., FIFO ordering in the queues of the switches, while maximizing the amount of time-triggered traffic that can be accommodated in the network. Moreover, we seek to minimize the cumulative queuing delays incurred by the time-triggered traffic, and hence, base our solutions on the "no-wait" principle, i.e., we strive to schedule transmissions of packets no later than they arrive at a particular node.

Overall, the resulting scheduling problems have a high time complexity, with most of them being NP-hard, thus requiring efficient heuristics for practical usage.

Modelling Hardware Capabilities

In this thesis, we rely on the usage of the programmable gating mechanisms specified in the IEEE 802.1Qbv (cf. Section 2.2.1) for enforcing the schedules on the switches. Thus, we are required to compute the transmission schedules in the form of gating programs for each of the ports of the switches in the network.

For networks in which scheduling is enforced on the end systems only while the switches merely perform the forwarding of packets, we are required to specifically consider in the scheduling model the effects of the best-effort traffic in transit on the time-triggered data streams. The lack of explicit control over the forwarding of best-effort traffic using commodity hardware must be accordingly modelled.

Incremental Updations

Developing algorithms that can compute transmission schedules for a given set of time-triggered data streams in a network topology is not sufficient for practical deployment. We also need mechanisms for modifying the existing schedules without affecting the other time-triggered data streams along with fast algorithms that can compute the changes that must be made to the schedules. After all, we seek to develop converged architectures for usage in smart factories where manufacturing systems may be modified any time.

We, hence, have to consider two variants of the scheduling problem for time-triggered traffic, viz., the static and the dynamic scheduling problem. In the static variant, the time-triggered data streams to be scheduled are known *apriori* and do not change at runtime, while the dynamic scheduling problem is aimed at scenarios in which the data streams to be scheduled in the network may change, for instance, due to connection/disconnection of new end systems into/from the network. The algorithms for the static scheduling problem aim to compute optimal solutions while the algorithms for the dynamic scheduling problem are engineered to deliver solutions faster.

1.2.2 Approach

Our approach to compute transmission schedules and routes for each of the aforementioned cases is as follows.

The high complexity of the scheduling problem for networks in which transmission schedules for time-triggered data streams is enforced on the switches as well as on the end system make it extremely challenging for additionally integrating routing in the problem formulation. One option to keep the problems tractable is to separate the

routing and scheduling stages for such networks, i.e., route the time-triggered data streams independently before computing the schedules based on the computed routes. However, for this, we would require specialized routing algorithms that are aware of the scheduling aspect to route the time-triggered data streams before computing their schedules. This separation of scheduling and routing stages can be further extended for usage in networks where the end systems are not synchronized, and thus, cannot enforce schedules.

We combine the scheduling and routing aspects for the networks in which only the end systems enforce scheduling. This is, too an extent, mandated by the scheduling model used in such scenarios to result in improved network utilization.

In networks where there are no synchronized clocks, we route the time-triggered streams over disjoint paths, i.e., spatially isolate the streams.

1.3 Scientific Contributions of this Thesis

In this thesis, we introduce scheduling and routing algorithms for computing routes and schedules for time-triggered traffic in Ethernet networks depending on where the schedules can be enforced. We present algorithms for computing optimal and heuristic solutions to the static scheduling problems which we extend in the next step to also compute incremental schedules for the dynamic scheduling problems. In particular, the contributions of this thesis are:

1. We compute transmission schedules for time-triggered data streams in networks where end systems are synchronized and switches are compliant with the IEEE 802.1Qbv standard by mapping the packet scheduling problem to *No-wait Job-shop Scheduling Problem (NW-JSP)*, a well-known problem from operations research. The resulting *No-wait Packet Scheduling Problem (NW-PSP)* can be formulated as an Integer Linear Program (ILP). We also propose a heuristic optimization algorithm based on the Tabu-search meta-heuristic for an efficient schedule computation. Moreover, we show how to further optimize calculated schedules by means of a *schedule compression* algorithm to reduce bandwidth wastage due to the presence of guard bands isolating the scheduled data streams and best-effort traffic. Finally, we also extend our solution for computing incremental schedules for the network and discuss the deployment of the solution in networks where end systems cannot participate in scheduling due to lack of synchronized clocks.

   These contributions are mainly based on [31]. The author of this thesis was primarily responsible for development of the meta-heuristics and concepts relating to the schedule compression. Overall, the author of this thesis contributed around 30 % of the total content in this paper.

2. For computation of transmission schedules for time-triggered data streams in networks equipped with the programmable gating mechanisms using the NW-PSP, the routes for the data streams must be available *a priori*. We show that the algorithm used for computing routes for the time-triggered data streams impacts their schedulability and identify parameters which can be used as heuristics for developing routing algorithms that are aware of the subsequent scheduling process. Based on these heuristics, we propose ILP-based routing algorithms for improving the schedulability of time-triggered data streams by up to 60 % and 30 % compared to shortest path routing and equal cost multi-pathing (ECMP), respectively.

   These contributions are based on [32]. The author of this thesis was instrumental in developing the concepts and for its evaluation. Overall, the author contributed around 60 % of the total content of this paper.

3. We introduced Time-sensitive Software-defined Network (TSSDN), an SDN-based network architecture, that can provide real-time guarantees for time-triggered traffic using a data plane consisting of commodity SDN hardware switches and synchronized end systems. Based on TSSDN, we formulate the joint scheduling and routing problem for time-triggered data streams in networks where scheduling can only be enforced on the end systems and not on the switches as an ILP. By adjusting the candidate paths over which time-triggered data streams can be routed, our approach can be adapted to explore the entire solution space for searching the optimum solution or to quickly compute heuristic solutions which may be slightly sub-optimal. Furthermore, we also show how the computed schedules can be adhered with using high speed packet processing frameworks, like Intel's Data Plane Development Kit [33] or netmap [34].

   These contributions are based on [35]. The author of this thesis developed the initial concepts for this work and was responsible for the evaluations. The author also significantly contributed in the write-up of the paper. To summarize, the author's total contribution in the content of this paper is around 70 %.

4. We also introduce the dynamic scheduling problem in TSSDN networks where the time-triggered data streams could be scheduled in the network dynamically without causing any disturbances to the data streams already scheduled in the network. Our heuristics for this dynamic scheduling problem approximate the optimal solutions of the corresponding static scheduling problem. We also propose optimizations to reduce the runtime of these algorithms to under a second for networks of realistic sizes.

   These contributions are primarily based on [36]. The author of this thesis contributed with ideas for this paper and was also responsible for the implementation and evaluation of the developed concepts. He also participated significantly in the

write-up of the paper. The overall contribution of the author to this publication is around 60 %.

5. Finally, we introduce the routing problem for time-triggered data streams in unsynchronized networks where scheduling cannot be enforced on any of the network participants. Here, we route the data streams over disjoint paths, i.e., spatially isolate the streams, and use in-network prioritization to bound the non-deterministic queuing delays for these data streams. We provide efficient algorithms for computing the routes for the time-triggered data streams.

   This work is based on [37]. The author's contribution to this work is about 60 % of the total content of this publication.

The contributions in this thesis were supported in many places by the guidance and efforts of Prof. Dr. Kurt Rothermel and Dr. Frank Dürr. Several student theses supervised by the author of thesis (in part as well as completely) also contributed towards the progress of this research presented in this thesis [38] [39] [40].

1.4 Graduate School of Excellence - advanced Manufacturing Engineering (GSaME)

The research presented in this thesis has been conducted within the framework of the Graduate School of Excellence advanced Manufacturing Engineering (GSaME), an excellence initiative at the University of Stuttgart. GSaME is a central scientific institution of the university, principally funded by the German Research Foundation (Deutsche Forschungsgemeinschaft) since 2007. GSaME brings together various disciplines ranging from mechanical and production engineering and computer sciences to business management in a bid to create interdisciplinary research groups working towards holistic development of concepts for the manufacturing systems of the future.

GSaME is divided into six research clusters, each with a specific focus areas. This thesis lies within the cluster C2 - *Informations- und Kommunikationstechnologien für die Produktion* in the GSaME organization. This cluster focuses on the enhancement of information processing and communication technologies for smart factories. In particular, this cluster aims to develop novel methods and architectures to bring in cutting edge information technologies into the manufacturing shopfloor.

Inline with the goals of the research cluster, we seek to utilize latest communication technologies for handling the traffic stemming from manufacturing systems in this thesis. The interdisciplinary nature of GSaME is also reflected in this thesis. In our research, we deal with problems relating to novel communication technologies foreseen to be deployed in the manufacturing systems of the future. We borrow well-established methods and results from the operations research branch of business management and

apply it to solve optimization problems for efficiently using the communication infrastructure in smart factories.

1.5 Structure of this Thesis

This thesis is structured as follows. In Chapter 2, we present a brief background of the networking technologies that are used in this thesis. In particular, we introduce the extensions to the IEEE 802.1Q standards relevant for handling scheduled traffic like time-triggered data streams. We also present a brief summary of OpenFlow, a popular SDN southbound protocol, used for implementing the routing algorithms in this thesis.

Chapter 3 presents our unified network model, specifically our assumptions pertaining to the network data plane and the network control plane, along with our model of time-triggered data streams.

In Chapter 4, we introduce the mapping of the scheduling problem in the networks compliant with the IEEE 802.1Qbv standard to the No-wait Job-shop Scheduling Problem (NW-JSP) and present heuristics based on Tabu-search for efficiently solving this problem. This chapter also presents the usage of Intel's Data Plane Development Kit (DPDK) for precise adherence to the computed schedules by the end systems.

Chapter 5 introduces the routing problem, i.e., the impact of routing time-triggered data streams on the computed schedules, for the networks compliant with the IEEE 802.1Qbv enhancements. It also presents ILP based formulations for computing routes for time-triggered data streams in such networks.

The architecture of TSSDN and the static scheduling and routing problem in networks where the schedules are enforced at the end systems alone is presented in Chapter 6. For this problem, we not only present an ILP formulation that computes the optimal schedules and routes, but also two heuristic solutions by restricting the number of paths over which time-triggered streams can be routed.

The dynamic scheduling problem in TSSDN is presented in Chapter 7. We present optimized scheduling algorithms that can compute the incremental schedules and routes for time-triggered data streams in realistic scenarios under a second.

In Chapter 8, we introduce the routing problem for time-triggered data streams in networks where the network participants cannot enforce schedules due to lack of synchronized clocks. The chapter also presents efficient algorithms based on meta-heuristic approaches to compute these routes for time-triggered data streams.

We conclude the thesis in Chapter 9 with a brief discussion of our contributions and an outlook towards future research in this direction.

Summary

With the advent of Industrial Internet of Things (IIoT), there is a growing desire to have converged networks for transporting time-sensitive traffic along with best-effort traffic in manufacturing systems. The proliferation of Ethernet along with its rapid pace of development has made it a strong contender for this purpose. However, Ethernet being designed for providing best-effort communication services, is not suitable for handling real-time traffic without further enhancements. An idea to provide bounded end-to-end latency and jitter for time-triggered data streams, for instance, a stream of samples transmitted by a sensor, in Ethernet is to schedule the traversal of such streams through the network such that the indeterministic queuing delay encountered by these streams is bounded. The computation of these schedules must take into account the capabilities of the network participants, i.e., who would be enforcing the schedules. Another major factor that affects the computation of schedules for time-triggered data streams is the paths over which these streams are routed.

This thesis mainly deals with different scheduling and routing problems, most of which have a high time complexity, with respect to the handling of time-triggered data streams in Ethernet networks. In particular, it classifies the solution space for scheduling the time-triggered data streams into four parts based on where the computed schedules are to be enforced. This thesis provides scheduling and routing solutions for each of these different scenarios.

CHAPTER 2

BACKGROUND

This chapter mainly presents different networking technologies and relevant IEEE standards that we refer to in this thesis for implementing routing and scheduling of time-triggered data streams.

2.1 Software-defined Networking

As already mentioned in Chapter 1, Software-defined networking (SDN) aims to improve flexibility of computer networks. The SDN paradigm is based on two main principles, viz., the separation of the network control plane from the network data plane and the logical centralization of the control plane with a global view on the data plane. The control plane interacts and configures the data plane using standardized interfaces known as the southbound protocols. OpenFlow, from the Open Networking Foundation, is one of the most popular southbound protocols and is on the path to become the de-facto SDN southbound protocol [41].

OpenFlow is a communication protocol between the data plane elements (switches) and the network controller hosting the control plane. Using OpenFlow, the network controller can program the routing tables of the switches, thus influencing the routes of the traffic in the data plane. In this thesis, we use OpenFlow for routing time-triggered data streams based on their computed schedules and routes in the Time-sensitive Software-defined Network (TSSDN) presented in Chapter 6. Though OpenFlow is well-documented, this section briefly describes the working of the protocol for the sake of completeness.

2.1.1 OpenFlow

OpenFlow switches process received packets based on the entries in its routing tables, also known as flow tables. Each entry (cf. Table 2.1) in the flow table represents a forwarding rule consisting of match fields, instructions and additional information for packet processing. Match fields store the values for the layer 2–4 header fields of the network stack which must match with those of the packet being processed for the rule to apply. If the rule applies for the packet being processed, then the actions specified in the instructions field will be executed on the packet. The possible actions range from rewriting some header fields like the source or destination MAC addresses to forwarding the packet over a certain egress port or even dropping the packet.

Additional information includes counters which store the number of packets that matched the corresponding flow table entry and were accordingly processed. The priority value is used to resolve conflicts when multiple flow table entries are applicable for any of the packets. The timeouts specify the amount of time for which the flow table entry is valid. The timeouts can be specified in terms of maximum amount of time for which the rule exists (hard time-out) or in terms of idle time after which the rule expires (soft time-out). OpenFlow also supports wildcarding of fields to have coarse-grained matches for the flow tables, thus enabling multiple flows to match on the same rule. It also allows chaining of actions to create a pipeline for more complex operations in the data plane.

Match Fields	Instructions	Counters	Priority	Timeouts	Cookie

Table 2.1: Flow table entries in an OpenFlow switch [42].

The OpenFlow protocol allows the SDN controller to connect to OpenFlow switches over a secure channel and read/write flow table entries, and thus, influence the routing of traffic through the network. The possibility of dynamically updating the flow tables brings its own set of challenges, e.g., the consistency of flow table entries across the switches in the network [43].

Recent versions of OpenFlow (since v1.4) provide transactional (all or nothing) semantics for updating flow table rules to avoid these consistency issues. We exploit these features for dynamically setting up time-triggered data streams in TSSDN (cf. Chapter 7) while avoiding problems for the existing data streams.

2.2 Time-sensitive Networking

The Institute of Electrical and Electronics Engineers (IEEE) set out in 2005 to equip Ethernet networks with extensions that would enable its usage for handling different

classes of real-time traffic along with best-effort traffic. The IEEE 802.1 AVB Task Group, now rechristened as the Time-sensitive Networking (TSN) Task Group, was created for this purpose. At the time of writing this thesis, the TSN Task Group had published a series of supplements for the existing IEEE 802.1Q standard, each targeting incorporation of specific real-time features in Ethernet [16]. Furthermore, several other extensions in draft stage were also intensively being worked upon for final publication.

The architecture of these extensions is influenced by the SDN paradigm of separation of concerns and centralized network configuration [25], i.e., these extensions also rely on a centralized network controller dealing with network configuration. A non-exhaustive list of TSN standards broadly classified into four categories is as follows:

1. Clock Synchronization
    - *IEEE 802.1AS* [29] - Timing and clock synchronization: Provides accurate clocks to the order of nanoseconds.
2. Scheduling of Traffic
    - *IEEE 802.1Qav* [44] - Forwarding and Queuing Enhancements for Time-sensitive Streams: Specifications for providing performance guarantees to allow for time-sensitive traffic like real-time audio video streams.
    - *IEEE 802.1Qbv* [26] - Enhancements for Scheduled Traffic: Time-aware shapers targeting real-time communication with deterministic bounds.
    - *IEEE 802.1Qbu* [45] - Frame pre-emption: Suspension of transmission of non-time-critical frames to transmit time-sensitive traffic.
    - *IEEE 802.1Qci* [46] - Per Stream Filtering and Policing: Detection and mitigation of potentially disruptive transmissions.
3. Bridging/Routing
    - *IEEE 802.1Qca* [27] - Path Control and Reservation: Provides explicit path control, bandwidth reservation, and redundancy for data streams.
    - *IEEE 802.1CB* [47] - Frame Replication and Elimination for Reliability: Provides mechanisms for frame replication on disjoint paths, sequence numbering and duplicate elimination.
4. Configuration
    - *IEEE 802.1Qat* [48] & *IEEE 802.1Qcc* [49] - Stream Reservation Protocol along with Enhancements and Performance Improvements: Provides protocols, procedures and managed objects for bridges and end stations.

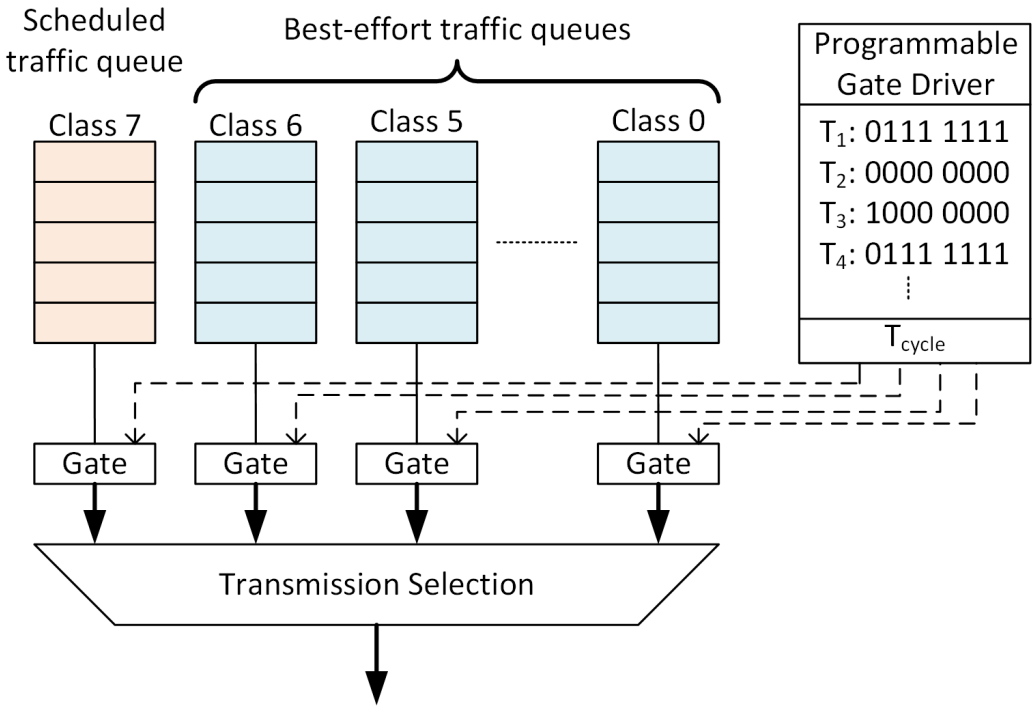


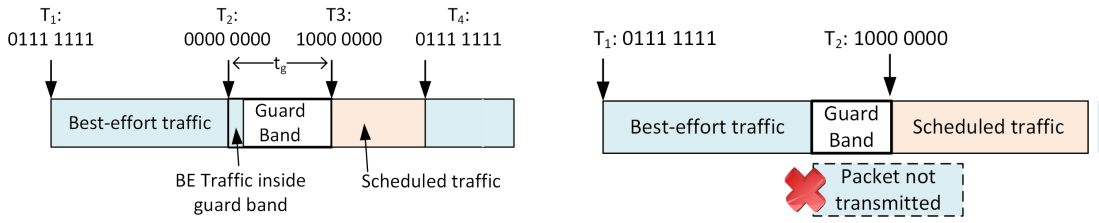
Figure 2.1: Programmable gating mechanism as an enhancement for handling scheduled traffic [16].

In this section, we briefly introduce two of these extensions which we use for handling time-triggered traffic in this thesis.We summarize the enhancements specified in the IEEE 802.1Qbv standard for handling scheduled traffic along with the functioning of the IEEE 802.1Qbu standard which enables frame preemption in Ethernet.

2.2.1 IEEE 802.1Qbv - Enhancements for Scheduled Traffic

The IEEE 802.1Qbv [26] is an extension for the IEEE 802.1Q published in 2016 which specifies the enhancements in Ethernet for handling scheduled traffic. This standard introduces a time-based programmable gating mechanism (cf. Figure 2.1), also referred to as time-aware shapers, that controls which of the queues at the egress port are considered for transmission selection.

The gating mechanism is to be programmed, for instance, over Simple Network Management Protocol (SNMP), with a sequence of gate events, each consisting of a relative time-stamp (represented as T_i in Figure 2.1) to the previous event in the sequence and a bit-mask indicating the queues which are to be considered for transmission selection till the next event. Packets in a queue are considered for transmission only if the corresponding gate is in "open" state. For instance, in Figure 2.1, after time T_2 all gates are closed (represented as state 0), while after time T_3 only the gate for traffic class 7 is open (represented as state 1). The programmed sequence is continually repeated after a



(a) Explicit Guard Bands (b) Implicit Guard Bands

Figure 2.2: Approaches to isolate scheduled traffic from best-effort traffic in IEEE 802.1Qbv networks

pre-programmed duration, denoted as T_{cycle}, resulting in a cyclic pattern of gate events for the queues of an egress port. Moreover, these cyclic patterns of gate events of the ports for all switches can be precisely aligned using clock synchronization protocols. It may be interesting to note that the switches equipped with these enhancements are no different than usual Ethernet switches implementing priority based forwarding when the gates of all queues are open.

In order to effectively deploy these enhancements to handle scheduled traffic, one or more queues per port is exclusively reserved to handle time-triggered data streams, e.g., queue corresponding to traffic class 7 is reserved for scheduled traffic in Figure 2.1. For now, we assume that the best-effort traffic uses all the other queues of the ports. The scheduled traffic is directed into the corresponding queue(s), for instance, by means of Priority Code Point (PCP) field in the IEEE 802.1Q VLAN header. The gating mechanism is then used to isolate this traffic from all the other traffic emerging from the other queues by means of the so-called "guard bands". The guard bands ensure that, on opening the gates corresponding to the scheduled traffic, the transmission of packets belonging to the time-triggered streams can immediately commence. The guard bands are necessitated by the fact that the closing of any gate does not have an impact on the transmission of the packet being currently transmitted even if it belongs to the queue whose gate is being closed. The switch continues its transmission till the current packet is completely transmitted. To ensure that the port is free for transmission when the gate for scheduled traffic is opened, guard bands avoid starting the transmission of any new packet belonging to any other classes of traffic immediately prior to the opening of the gate for scheduled traffic. Hence, the width of guard bands, t_g, must be at least equal to the time required for serializing an MTU-sized packet, i.e., 1500 bytes for Ethernet, on the port of the switch. Obviously, a significant amount of bandwidth, if not all, enclosed in these guard bands is wasted.

The guard bands are created by closing the gates for best-effort traffic at least time t_g in advance before opening the gate for scheduled traffic. For instance, in Figure 2.2a, the gates for best-effort traffic are closed at time T_2 (the current packet being transmitted is however allowed to continue to completion), though the gate for scheduled

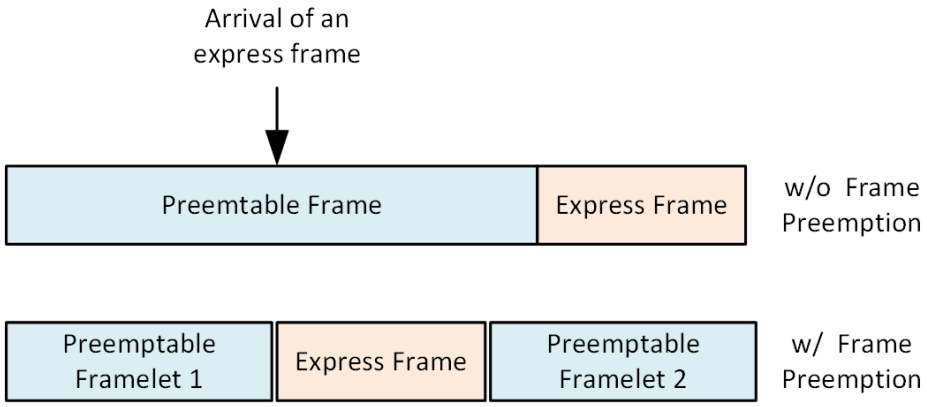


Figure 2.3: Frame preemption as in IEEE 802.1Qbu

traffic opens only at time T_3. This approach requires explicit computation of guard bands during schedule generation to appropriately program the gate-drivers. An alternative approach specified in the standard is to transmit a best-effort packet only if its transmission would finish before the corresponding gate closes, e.g., a packet is not transmitted in Figure 2.2b as its transmission will not finish before the closure of the gate. While this approach does not need explicit computation of guard bands, they are implicitly created, as shown in Figure 2.2b. The approach with implicit guard bands is slightly better than the one with explicit guard bands as it enables the utilization of bandwidth wherever possible till the closure of the gate. However, bandwidth wastage associated with the guard bands between the scheduled traffic and other traffic cannot be completely avoided.

2.2.2 IEEE 802.1Qbu - Frame Pre-emption

IEEE 802.1Qbu [45] is an extension to the IEEE 802.1Q standard dealing with frame pre-emption in IEEE 802.3 Ethernet networks. With IEEE 802.1Qbu, the queues of an egress port can be configured as an express queue or a preemptable queue. Ethernet frames from the express queues are designated as express frames, while those in the preemptable queues are designated as preemptable frames. An express frame ready for transmission can preempt a preemptable frame being transmitted. The preemptable frame resumes with its transmission after the express frame finishes. These express frames themselves cannot be preempted during the transmission. This also implies that configuring a queue as express queue raises its priority above that of all other queues configured as preemptable.

As shown in Figure 2.3, in absence of preemption, an express Ethernet frame can be delayed by the amount of time required to transmit the remaining part of a preemptable Ethernet frame. In worst cases, this delay would be equal to the time required for transmitting an MTU-sized packet. With frame preemption, an express frame is sent

between the framelets of the preemptable frame being transmitted. This dramatically reduces the delay for the express frame. It must be noted that the framelets of the preemptable frames must also adhere with sizing constraints of the Ethernet standard, i.e., framelets of sizes less than 64 bytes are disallowed. Thus, frames of sizes less than 128 bytes (including the 4 bytes of CRC) cannot be preempted despite being preemptable frames. Furthermore, the last 64 bytes of the transmission of preemptable frames cannot be preempted. Thus, overall, the frame preemption standard guarantees delay bounds between 64–128 byte transmission times per hop for express frames given that the frame is next in its queue for transmission.

To benefit from frame preemption while handling scheduled traffic, it is configured to traverse through express queues, while all other traffic traverses through preemptable queues. This enables lower bandwidth wastage stemming from the guardbands associated with the IEEE 802.1Qbv enhancements by reducing their width to 128 byte transmission times instead of the time required to serialize an MTU sized packet.

While the frame pre-emption in Ethernet was designed to work in tandem with the programmable gating mechanism to reduce the network bandwidth inside the guard band, it may also be used in isolation to simply reduce the waiting time for higher priority traffic. This particular feature can be potentially used to improve the performance of TSSDN, as we discuss in Chapter 6.

CHAPTER 3

SYSTEM MODEL

This chapter introduces a unified system model for the scheduling and routing algorithms for time-triggered traffic in Ethernet networks presented in this thesis. We base our system model, shown in Figure 3.1, on the paradigm of software-defined networking, and hence, divide the model into two parts, viz., the data plane and the control plane. The data plane and the control plane models are presented in Section 3.1 and 3.2, respectively, while the modelling of time-triggered traffic is handled in Section 3.3.

It must be noted that this unified system model presents aspects which are common across all the scheduling and routing algorithms presented in the thesis. Additional assumptions and restrictions, if any, are specified in the respective chapters.

3.1 Data Plane

The data plane in our system model consists of end systems (hosts) and network elements (switches). The network elements are full-duplex layer-2 Ethernet switches with priority queues (as specified in the IEEE 802.1Q standard) typically available in commodity switches. We use these queues for isolating the time-triggered traffic from the other kinds of traffic like shaped traffic, best-effort traffic etc. The diameter of the network, i.e., the longest shortest-path between nodes, is limited, e.g. to 7 hops, in accordance to the definition of local area networks (LAN) in the IEEE 802.1D standard. Furthermore, the switches provide interfaces, e.g., OpenFlow, by means of which the routing of the traffic in the network can be influenced. For networks with synchronized switches, the clocks of the switches across the network are synchronized using state-of-the-art clock synchronization algorithms like the IEEE 802.1AS protocol or the IEEE 1588 Precision Time Protocol. The synchronization of switches enable the enforcement of transmission schedules for time-triggered traffic in the switches, i.e., in-network. We

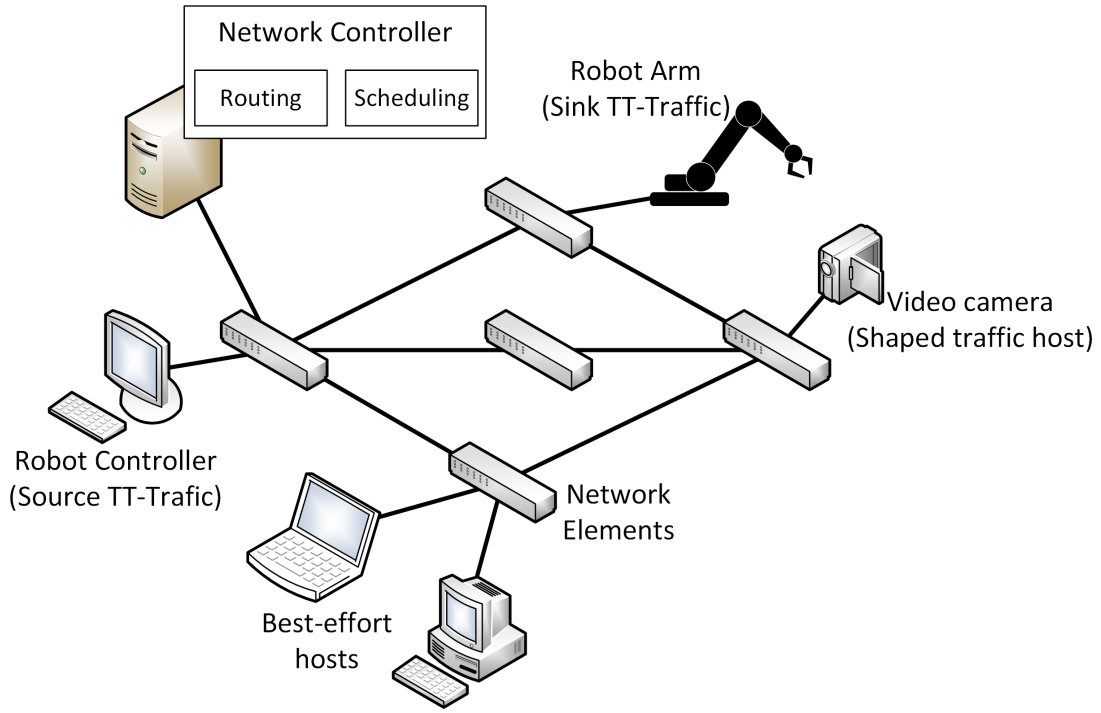


Figure 3.1: Unified System Model

mainly focus on the programmable gating mechanism specified in the IEEE 802.1Qbv extension for this purpose.

The end systems (also known as hosts) in our network model are basically data sources and sinks for the network. As we focus on time-triggered traffic in the networks, only the end systems acting as sources and sinks of time-triggered traffic play a role in the scheduling algorithms, e.g., sensors and actuators in the manufacturing shopfloor. The sources of time-triggered traffic transmit application data units encapsulated in UDP packets with a constant bit-rate. Given that the transmission of time-triggered traffic follows a schedule, properties like flow control, congestion control, etc. (usually offered by the transport layer of the network stack) can be integrated into the generated schedules by the scheduling algorithms. The source hosts label time-triggered traffic using the appropriate prioritization mechanisms available for instance using the Priority Code Point (PCP) in IEEE 802.1Q (VLAN) header. In networks requiring synchronization, the clocks of the hosts can be synchronized using appropriate protocols. Additionally, source hosts in such networks are allocated transmission schedules for time-triggered traffic which they precisely adhere to, if required using specialized packet processing frameworks. This implies that these source hosts take into consideration the transmission schedules to adjust their individual task schedules such that the application data units to be sent are ready in time.

3.2 Control Plane

In software-defined networking, the control plane is primarily responsible for configuring the data plane based on its global knowledge. In this thesis, we assume a centralized control plane which computes transmission schedules and routes for the time-triggered traffic based on its knowledge of the network traffic and network topology while considering the capabilities of the data plane.

As shown in Figure 3.1, the control plane is hosted on a standard server, referred to as a network controller. The network controller communicates with the switches using well-defined configuration protocols, e.g., the popular SDN southbound protocol OpenFlow, Simple Network Management Protocol (SNMP), etc. Overall, the network controller has a logically centralized view on the data plane, including the network topology (can be gathered using the Link Layer Discovery Protocol (LLDP)), traffic, etc., which facilitates implementation of control logic like computation of transmission schedules and routes for time-triggered traffic. In this thesis, we restrict our focus to centralized network controller and do not consider the problem of control plane distribution.

3.3 Time-triggered Traffic

In this thesis, we restrict time-triggered traffic to periodic flows (we refer to data streams also as flows), i.e., consecutive transmissions of the stream by the source host are exactly separated by a pre-defined time-period. The time-triggered paradigm is well suited for control systems using sensors with fixed sampling periods or actuators requiring inputs within given time intervals. While event-triggered control systems are shown to have better performance, they are still not in wide usage as the events may be triggered at arbitrary times leading to some amount of indeterminism [50] [51]. Moreover, event-based paradigm can be adapted to be used over a time-triggered infrastructure by constraining the events to periodically occurring time-slots [52]. Thus, we model traffic stemming from such event-based control systems in the network as a time-triggered flow where the period of the flow is the minimal inter-arrival time of the events for the corresponding flows, i.e., such flows are scheduled into the network considering the pessimistic case.

For the scheduling and routing algorithms presented in this thesis, time-triggered flows are represented by a unique tuple consisting of its source host, destination host(s), its period, and the number of bytes the source sends during each period.

We base our scheduling approaches on the "no-wait" principle, i.e., we strive that time-triggered traffic does not incur any queuing delays in the network. Thus, we do not account for flow deadlines in our model. The total delay (cumulative sum of processing, transmission, and propagation delays) that time-triggered flows would incur in our system model would be completely dependent on the paths over which they are routed. Given that we consider networks of limited sizes (diameter restricted

to 7 hops), the total delay incurred by time-triggered flows is also limited. While we do not focus explicitly on the individual flow deadlines, our routing approaches can be easily extended for this purpose.

CHAPTER 4

SCHEDULING IN NETWORKS WITH IEEE 802.1QBV EXTENSIONS

4.1 Introduction

In this chapter, we look at the scheduling problem for time-triggered flows in networks compliant with the IEEE 802.1Qbv standard. Here, we assume that the network participants including the end systems are fully synchronized and the switches are equipped with a programmable gating mechanism (cf. Section 2.2.1) between the output queues and the transmission selection module of the ports. Although the IEEE 802.1Qbv standard specifies the working of the programmable gating mechanism elaborately, the computation of gating schedules based on the time-triggered flows to be accommodated in the network is out of the scope of the standard. Furthermore, the computation of these gating schedules is far from trivial as it is imperative to compute per-hop schedules for time-triggered flows while respecting the constraints of the underlying hardware to benefit from the possibility of enforcing schedules on switches.

We tackle this scheduling problem by mapping it to the No-wait Job-shop Scheduling Problem (NW-JSP) [53], a well-known problem from the field of operations research [31]. To this end, we show how NW-JSP can be adapted to a No-wait Packet Scheduling Problem (NW-PSP) for computing compact gating schedules while yielding minimum network delay for time-triggered flows. We also show that instances of NW-PSP can be formulated as Integer Linear Programs (ILP) for calculating exact solutions. However, given the complexity of the scheduling problem, the ILP formulations for NW-PSP do not scale beyond very small problem instances. Therefore, we also propose a heuristic optimization algorithm based on the Tabu search metaheuristic that allows for efficient schedule calculation. Finally, we also show how to further optimize the computed schedules through a schedule compression technique for

reducing the number of entries in the gating program and the corresponding bandwidth wastage stemming from the resulting guard bands separating the scheduled traffic from other traffic.

While the gating mechanisms in the switches can be used for ensuring schedule compliance for the time-triggered flows on the switches, no special mechanisms are available for the end systems, e.g., the source host, to adhere with the computed schedules. Deviations from the schedules at the source hosts will result in problems not only for the packets of the corresponding flows but also for other scheduled traffic in the network. Given that the queues at output ports of switches have FIFO semantics, arriving at any of the network nodes earlier than the computed schedule may result in blocking of a link for the packets actually scheduled to traverse over the link at that point of time. Our evaluations show that socket APIs provided by modern operating systems are inadequate for the end systems to comply with the schedules with sufficient precision. To solve the problem with schedule adherence, we present a proof-of-concept implementation showing that the source hosts of time-triggered flows can accurately comply with the computed transmission schedule using specialized high speed packet processing frameworks like Intel's Data Plane Development Kit (DPDK) [33].

This chapter is structured as follows. We present the differences to the unified system model with respect to this scheduling problem and the concrete problem statement in Section 4.2. In Section 4.3, we describe the mapping between NW-PSP and NW-JSP and present the corresponding ILP formulation. In Section 4.4, we present an efficient Tabu search algorithm for NW-PSP along with our approach to compress schedules. The usage of packet processing frameworks for schedule adherence is presented in Section 4.5. Finally, we present the relevant evaluation results and a brief discussion on extending these approaches for networks with unsynchronized end systems in Sections 4.6 and 4.7, respectively.

4.2 System Model & Problem Statement

4.2.1 System Model

For modelling the scheduling problem in networks compliant with the IEEE 802.1Qbv standards, we specify additional conditions for the network data plane and the time-triggered traffic in the unified system model specified in Chapter 3.

In the data plane, we assume that all switches are equipped with the extensions specified in the IEEE 802.1Qbv standard. We further assume that only one queue is reserved per output port for scheduled traffic in all switches. While the standard does not forbid using multiple queues for scheduled traffic, we refrain from using more than one queue as they are usually limited in number, and hence, a valuable resource in a converged network. We also assume that the queue reserved for scheduled traffic throughout the network belongs to the same class, e.g., queue belonging to traffic class 7. This

limitation can be set aside for switches allowing modification of the Priority Code Point (PCP) field in the VLAN tag to correspond with the traffic class of the queue reserved for scheduled traffic in the switch at the next hop. Furthermore, we presume a store-and-forward behaviour of the switches, i.e., the packets are fully received by the switches before being processed (lookup of flow tables to decide the output port) and forwarded. Finally, we also require that the clocks of all end systems and switches are synchronized using the IEEE 802.1AS protocol.

We model time-triggered flows as periodic flows (unicast as well as multicast) with a constant time-period and payload size, i.e., the source host of a flow transmits fixed amount of data periodically. We also add additional restrictions on the time-triggered flows with respect to their time-periods. We require that all flows have time-periods which are integral multiples of what we refer to as the "base-period", t_{bp}. Here, base-period refers to the minimum transmission period that can be supported in the network. To handle flows with arbitrary periods, we reduce their time-periods to the nearest integral multiple of base-period.

4.2.2 Problem Statement

The computation of per-hop schedules for time-triggered traffic in time-sensitive networks, in general, is equivalent to bin packing problem and is thus NP-hard [54]. Given a set of time-triggered flows along with their routes, schedules which result in minimal end-to-end delays for packets belonging to the time-triggered flows are preferred. Additionally, the scheduling algorithm must also consider the number of gating events required to enforce the schedule and the resulting bandwidth wastage from the corresponding guard bands. It must be noted that the gating events resulting from transmission scheduling can only be reduced and not completely eliminated. For this, the scheduling algorithm must compute schedules such that gate opening events lead to transmission of several packets of scheduled traffic.

To this end, we compute *transmission schedules with minimal duration* (also known as makespan) for a set of time-triggered flows known *a priori*, as reduced makespan also imply fewer gate opening events. Further, we compress the schedules using a procedure designed to explicitly reduce the number of gate opening events.

4.3 Mapping Packet Scheduling to Job-shop Scheduling

In this section we show how we adapt a well-studied scheduling problem from the field of operations research to model transmission scheduling in networks compliant with the IEEE 802.1Qbv standard.

4.3.1 Background: Job-shop Scheduling (JSP)

To define the packet scheduling problem, we first briefly introduce the *Job-shop Scheduling Problem (JSP)*. JSP is a well known scheduling problem from operational research. Informally, JSP can be described as follows. Given is a set of machines and set of jobs, each consisting of a sequence of operations like milling, drilling, etc. that have to be executed on the machines in a given sequence. Each operation can be executed on exactly one machine of the set of machines, e.g., a drilling operation on a drilling machine, and operations take a defined time duration to be completed. Moreover, each machine can only process one operation at a time. The JSP is a constrained optimization problem that tries to schedule each operation on the corresponding machine such that no more than one operation is processed at the same time on any machine (constraint). The schedule is expressed by the starting times of the operations for each job. Moreover, JSP tries to minimize the so-called *makespan* of processing, where makespan is defined as the maximum of the finishing times of the last operations for all jobs, i.e., the goal is to finish the set of jobs as fast as possible (objective). Other objectives can be defined such as minimum tardiness, which are however not useful with respect to the packet scheduling problem that we discuss here.

The *No-wait Job Shop Scheduling Problem (NW-JSP)* adds an additional constraint: after a job is started, it cannot be interrupted, i.e., it must run to completion without any time gaps between the processing of the operations of the job. With this constraint, the schedule can be defined by the starting time of the first operation of each job alone. The no-wait property is important in many manufacturing scenarios. For instance, an iron has to be forged immediately after heating it up, and as we see later this also has a meaning for modelling our packet scheduling problem.

It is worth to note that JSP is NP-complete and known as one of the most difficult combinatorial optimization problems [55], and NW-JSP is also NP-hard [56].

4.3.2 The No-wait Packet Scheduling Problem (NW-PSP)

We now investigate how to adapt NW-JSP to calculate the transmission schedules for time-triggered flows with respect to the gating mechanism. We call the corresponding problem the *No-wait Packet Scheduling Problem (NW-PSP)*. The overall goal is to compute the times when packets should be injected into the network by the network interface controllers (NIC) at the source hosts and the times for opening and closing the gates for scheduled packets on the corresponding ports of the switches en route its destination.

The basic idea of mapping NW-JSP to NW-PSP is to map switches and NICs to machines performing forwarding operations on packets. A time-triggered flow is then a job corresponding to a sequence of transmission operations, one for each NIC/switch

along the given path of the flow. Packets should be forwarded immediately without delay, which intuitively corresponds to the no-wait property of NW-JSP.

In order to come up with a formal formulation of NW-PSP, we need to refine this basic idea further. In particular, the mapping "one switch, one machine" is too coarse-grained since typically a switch can forward several packets in parallel. So we need to ask the question: Which operations can or cannot be performed in parallel with respect to forwarding of packets?

The network delay is more complex than the processing times for the operations of the jobs in the manufacturing shopfloor. Network delay can be broken down into several types of delays, viz., the propagation delay of signals along the link, the processing delay for deciding on which port to forward an incoming packet based on the flow tables, the queuing delay of a packet in the queue of an outgoing port, and the transmission delay to serialize the packet on the wire. Note that the time intervals for propagation, processing, queuing, and transmission of each packet are ordered strictly sequentially and do not overlap since we assume a store-and-forward behaviour for switches rather than cut-through forwarding. Before a packet can be processed, it must be received completely and put into an internal buffer. Packet processing then inspects the header fields and decides into which outgoing queue should the packet be enqueued. When processing is finished, the packet can be transmitted on the outgoing link as soon as it is at the head of the queue and the corresponding gate is open.

Due to physical restrictions, two packets cannot be transmitted over the same outgoing port at the same time as their electrical signals could potentially interfere with each other. Therefore, the transmission intervals of two packets using the same outgoing port must not overlap, e.g., the transmission of packets belonging to Flow 2 and Flow 3 over Port 5 of the same switch in Figure 4.1. However, a switch can process several packets received in parallel from different incoming ports at the same time as shown for Flow 1 and Flow 2 in Figure 4.1. On similar lines, a NIC cannot transmit two packets at the same time.

Based on this observation, *switch ports* and *NICs* now correspond to machines of NW-JSP. The set of all switch ports and NICs in the network is denoted as $P \equiv \{P_1, \ldots, P_m\}$. *Time-triggered flows* correspond to jobs, and we denote the set of flows as $F \equiv \{F_1, \ldots, F_n\}$. Each flow F_i consists of a sequence of *transmission operations* $O_i \equiv (O_{i,1}, \ldots, O_{i,n_i})$, with one transmission operation for each outgoing port along the given path of the flow. Relation $\mathcal{R} : O \mapsto P$ maps each transmission operation to an outgoing switch port, i.e., relation $\mathcal{R}$ corresponds the mapping of the operations of the jobs to the machines. Sequence O_i together with relation $\mathcal{R}$ are parameters of NW-PSP defining the route of a flow through the network.

Similar to the machines processing operations of a job, switch ports and NICs perform the transmission of packets of a flow. Note that according to this model, an operation

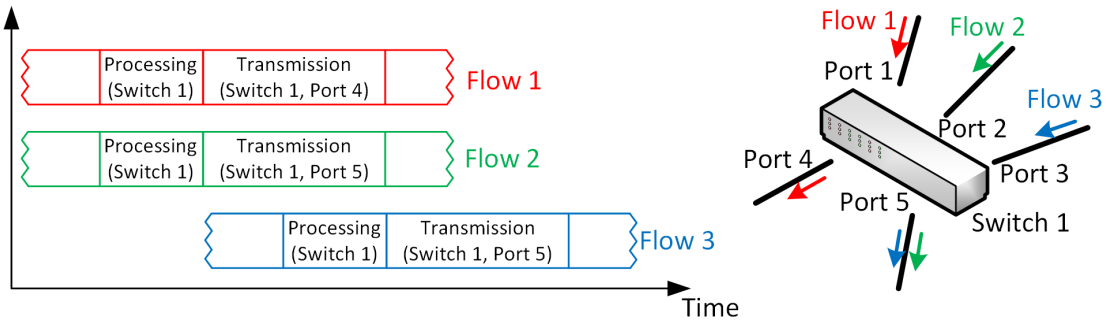


Figure 4.1: Timeline of forwarding three packets of three different flows over one switch. No queuing delay is shown since packets are forwarded immediately with no-wait packet scheduling.

only includes the transmission. Packet processing and propagation are not parts of the operation. The processing time of a job then corresponds to the transmission delay of a packet, which is defined by the given packet size and the data rate of the corresponding switch port or NIC. Parameter $d_{i,j}^{trans}$ defines the transmission delay for each transmission operation $O_{i,j}$ of flow F_i.

In addition to the transmission delay, we also have to consider propagation, processing and queuing delays. Due to the no-wait property, the queuing delay is per-definition zero. Furthermore, without loss of generality, we assume the same processing delay, d^{proc}, and propagation delay, d^{prop}, for all switch ports and NICs in the network. This simplifying assumption could be easily relaxed by defining individual processing and propagation delays for each switch port and NIC, respectively.

In contrast to adjacent operations of a NW-JSP job, in NW-PSP adjacent transmission operations on neighbouring switches of a flow cannot be processed "back-to-back" without time gap. After one switch port has completed the transmission of the last bit of the packet, the bit first has to propagate to the neighbouring switch, and then the packet needs to be processed before the transmission on the next outgoing port can start. Therefore, we need to consider the propagation and processing delays preceding each transmission interval. Let $D_{i,j}$ define the cumulative network delay (processing, propagation, and transmission delay) up to and including transmission operation $O_{i,j}$ of flow F_i:

$$D_{i,j} = (j-1)(d^{prop} + d^{proc}) + \sum_{k=1,\dots,j} d_{i,k}^{trans} \tag{4.1}$$

Then the essential constraint that two conflicting transmission operations $O_{i,k}$ and $O_{j,l}$, belonging to flows F_i and F_j respectively, using the same outgoing port must not

overlap in time can be defined as follows:

$$t_j - t_i \geq D_{i,k} - (D_{j,l-1} + d^{prop} + d^{proc}) \quad \text{OR} \tag{4.2}$$

$$t_i - t_j \geq D_{j,l} - (D_{i,k-1} + d^{prop} + d^{proc}) \tag{4.3}$$

Here, t_i and t_j denote the start of the packet transmission on the source hosts for flows F_i and F_j, respectively. Informally, these constraints ensure that before the first bit of a packet from flow F_i is transmitted over a switch port, the last bit of the packet from flow F_j must have been transmitted over this port, or vice versa. $t_j + D_{j,l-1} + d^{prop} + d^{proc}$ defines the time when the packet is enqueued in the queue of the outgoing port after having transmitted this packet over the previous port, propagating the packet to the next switch, and processing the packet there. Note that packets are transmitted immediately after processing without waiting in the outgoing queue of the port (no-wait forwarding property of NW-PSP). Thus, packets travel through the network "non-stop" with *minimum possible network delay*. For this very reason, we also do not consider the deadlines of the flows, i.e., the maximum end-to-end latency that packets belonging to a flow may suffer in the network, while modelling the scheduling problem. Another inherent advantage of no-wait forwarding is that the *queue size of scheduled traffic is minimal*, and that the switches can potentially dedicate this memory to best-effort traffic.

The set of all flow start times $T \equiv \{t_1, \ldots, t_n\}$, i.e., the time of injecting packet at source NIC of the corresponding flows, are the variables of NW-PSP. Given these start times, we can calculate packet schedules for transmitting packets on all switches (gate opening times) lying in the route of the flow. A transmission operation $O_{i,k}$ needs to be scheduled at time $t_{i,k} = t_i + D_{i,k} - d_i^{trans}$ on port $\mathcal{R}(O_{i,k})$. This schedule is repeated in each cycle, where $t = 0$ defines the start of a cycle and t_{bp} is the length of the scheduling cycle.

Note that simply repeating the schedule in each cycle only works because of our assumption that all flows have the period which are integral multiples of the base-period, t_{bp}, which is also the length of the scheduling cycle. The source hosts of flows with periods higher than the base-period will not transmit during all scheduling cycles, i.e., a flow with period twice the base-period will transmit every alternate scheduling cycle.

The objective of NW-PSP is to *minimize the flowspan*, C_{max}, which we define as the equivalent to the NW-JSP makespan. Let $C_i = t_i + (n_i)(d^{prop} + d^{proc}) + \sum_{k=1,\ldots,n_i} d_{i,k}^{trans}$ be the finishing time of flow F_i. Then, the *flowspan* $C_{max} = \max\{C_i | i \in \{1, \ldots, n\}\}$ is the finishing time of the flow finishing last.

Minimizing the flowspan results in compact schedules where the time-triggered flows are not distributed across the whole cycle, but are bunched towards the beginning of the scheduling cycle. While we discuss other effects of this optimization objective later in Section 4.7, it may be interesting to note that if the flowspan of the computed

schedule is greater than the base-period, t_{bp}, then the set of time-triggered flows may not schedulable. This is because transmissions of time-triggered flows from one cycle may interfere with the transmissions from the next cycle. Thus, flowspan also provides a metric for evaluating the schedulability of a set of time-triggered flows in a given topology. We would exploit this property of the flowspan to compute routes for time-triggered flows in Chapter 5.

4.3.3 Integer Linear Program for NW-PSP

Similar to the orginal NW-JSP, NW-PSP can be also formulated as an Integer Linear Program (ILP).

The ILP formulation for this problem is as follows:

$$\text{Minimize } C_{max} \tag{4.4}$$

Subject to the following constraints:

$$\begin{gathered} \forall \{O_{i,k}, O_{j,l}\} \in K \\ t_j - t_i - D_{i,k} + D_{j,l-1} + d^{prop} + d^{proc} \leq \quad c\, x_{i,k,j,l} \end{gathered} \tag{4.5}$$

$$\begin{gathered} \forall \{O_{i,k}, O_{j,l}\} \in K \\ t_i - t_j - D_{j,l} + D_{i,k-1} + d^{prop} + d^{proc} \leq \quad c\,(1 - x_{i,k,j,l}) \end{gathered} \tag{4.6}$$

The constraints of Equation 4.5 and 4.6 correspond to the constraints of Equation 4.2 and 4.3, respectively, after translating the disjunctive form to a conjunctive form as required by ILPs. To this end, we introduce binary variables $x_{i,k,j,l} \in \{0, 1\}$ for each pair of conflicting forwarding operations $\{O_{i,k}, O_{j,l}\}$ with the same outgoing port. Set $K = \{\{O_{i,k}, O_{j,l}\} \,|\, \mathcal{R}(O_{i,k}) = \mathcal{R}(O_{j,l}) \wedge O_{i,k} \neq O_{j,l}\}$ defines all such pairs of conflicting forwarding operations. Here, c is a large constant (virtually infinity). Depending on the value of x, either the first or second constraint is effective with a right-hand side value of zero. The ineffective constraint is then evaluating to true since "infinity" is greater than anything, ensuring the correct semantic of the conjunctive form.

4.4 Heuristics for NW-PSP

Given that NW-JSP can be reduced to NW-PSP, the NW-PSP is a NP-hard problem. Therefore, we cannot expect to find exact solutions efficiently using the ILP formulation from the previous section for larger scenarios with many time-triggered flows. In this section, we present heuristics for efficiently solving the NW-PSP based on the Tabu search algorithm for NW-JSP from [56]. We also use the exact solutions generated from the ILP formulations as a benchmark for our heuristics.

For the ease of modelling, we make a few changes to the mapping between NW-PSP and NW-JSP. Instead of using a single machine to model a switch port, we now model it using multiple machines: one for processing of incoming packets on the port (this models the processing and the propagation delay incurred by the packet), and one for transmitting the packets going out on the port as explained in Section 4.3, i.e., we now use additional machines to account for processing and propagation delays of packets. Thus, a time-triggered flow is now modelled to consist of a sequence of operations (transmission and processing) executed by different machines. After a machine transmits a packet on a link, another machine responsible for the input port on the next switch in its path processes it. While this mapping makes it easier to apply Tabu search for flowspan optimization of NW-PSP, this model also has the inherent shortcoming that processing of incoming packets on the same port cannot be achieved in parallel as in the unmodified problem formulation. However, such situations would not arise as the transmission delays, typically, dominate the processing delays in commodity switches.

With the no-wait constraints also applicable for NW-PSP, the overall schedule can be specified by the start times for each of the flows. The schedule for all the constituent operations of a flow can be calculated from its start time. Hence, it suffices if the heuristic is specifically targeted for computing the start times of the flows. To this end, we split the NW-PSP into a *time-tabling* problem and a *sequencing* problem based on the approach in [56]. The time-tabling problem deals with the computation of the start times for all the time-triggered flows belonging to a *totally ordered set of flows*. The sequencing problem, on the other hand, deals with *totally ordering the set of flows being scheduled* such that the given time-tabling algorithm results in a schedule with minimal flowspan. In the following, we describe the used time-tabling algorithm (based on a greedy approach) and the sequencing algorithm (based on Tabu search).

4.4.1 Time-tabling Problem

Given a totally ordered set of flows (solution of the corresponding sequencing problem), we use a greedy approach to solve the time-tabling problem for computing start-times of the flows. Consider a totally ordered set of flows, $FO \equiv \{F_1, F_2, \ldots, F_n | F_i \rightarrow F_j; \forall i \leq j\}$. The time-tabling algorithm, presented in Algorithm 1, allocates the earliest possible starting time (Line 4) for each flow in set FO based on their order, one flow at a time, subject to the constraints imposed by the starting times of all the preceding flows, i.e., no machine (switch port/NIC) should be scheduled to process or transmit packets belonging to more than one flow at the same time. The algorithm sets initial start time of the flow being scheduled as 0, and increases it in steps (size based on set of conflicting operations of preceding flows) till all the operations of the current flow are scheduled without any conflicts with the preceding flows. Based on the start times of the flows and its cumulative times for executing its operations, the flowspan is computed (Line 6).

While earliest possible starting time of flows as a heuristic do not always yield optimal solutions, they approximate schedules with minimal flowspan [56]. It may be noted that the worst-case time complexity of this time-tabling algorithm is $\mathcal{O}(n^3N^3)$, where n is the number of flows and N is the maximum number of constituent operations (transmission as well as processing) in any flows. Given our assumption of a bounded network diameter, the maximum number of constituent operations is also bounded. This is because the length of path over a flow may be routed is restricted to 7 hops. Thus, the worst-case time complexity of the used time-tabling algorithm is $\mathcal{O}(n^3)$.

Algorithm 1 Time-tabling Algorithm

```
1: function TIMETABLER(FO)
2:     Schedule ← { }, Span ← 0
3:     for each flow in FO do
4:         FlowStartTime ← Earliest possible start time
5:         Schedule[flow] ← FlowStartTime
6:         Span ← max(Span, FlowStartTime + flow.totalFlowTime)
7:     end for
8:     return (Span, Schedule)
9: end function
```

4.4.2 Sequencing Problem

The sequencing algorithm creates a total ordering of flows in the set of flows to be scheduled, F, to minimize the resulting makespan of the schedule computed by the time-tabling algorithm. The search space for the sequencing problem consists of $n!$ possibilities, where n is the number of flows in the NW-PSP instance. A brute force approach to compute the optimal solution would involve each of the possible sequences to be executed with the presented time-tabling algorithm (complexity of $\mathcal{O}(n^3)$), and would therefore not scale to large problem sizes. Hence, we use Tabu search for a guided exploration of the solution space.

Tabu search is a well defined method for exploration of solution space in optimization problems [57]. Tabu search mainly generates an initial solution (based on a heuristic) and iteratively processes it by selecting the best possible solution in the neighbourhood that does not violate certain criterion (being on the tabu list) for the next iteration. After a pre-defined number of iterations, the best-ever solution encountered over these iterations is selected. We adapt the Tabu search developed for solving a NW-JSP (from [56]) to solve an instance of NW-PSP modelling the scheduling problem in discussion. The existing algorithms for NW-JSP have been primarily designed

Algorithm 2 Sequencing Algorithm

```
1: function SEQUENCER(F)
2:     currentSoln ← GenerateInitialSolution(heuristic(F))
3:     bestOrder ← currentSoln
4:     while Termination criteria not satisifed do
5:         neighbourhood ← GenerateNeighbourhood(currentSoln)
6:         for each soln in neighbourhood do
7:             computeMakeSpan(soln)
8:         end for
9:         selectedSoln ← SolutionSelection()
10:        currentSoln ← selectedSoln
11:        if selectedSoln better than bestOrder then
12:            bestOrder ← selectedSoln
13:        end if
14:    end while
15:    return bestOrder
16: end function
```

and evaluated for problem with 30–50 jobs. For NW-PSP, we aim for a solution that scales up to 1000+ flows. In the following, we describe the steps involved in our Tabu search method for the sequencing problem, summarized in Algorithm 2.

Initial Solutions

In Tabu search, initial solutions are typically generated using heuristics. Of the many popular heuristics available for NW-JSP, we chose two for generating initial solutions for NW-PSP, viz., the sum of processing times for all the constituent operations of a flow and the processing time of the longest operation of a flow. The flows may be ordered using these heuristics in ascending or descending order to create an initial solution. As suggested in [56], we also use random ordering for obtaining an initial solution. Overall, we execute five runs of the Tabu search algorithm, each starting with a different initial solution—four generated from the two heuristics in ascending and descending order, one based on a random ordering—and choose the best solution with respect to the flowspan from all these runs.

Neighbourhood Generation

After the generation of the initial solution, we iteratively process them to reduce the resulting flowspan. During each iteration the neighbourhood of the current solution is generated based on its "critical flow". Critical flow of an ordering is the flow which finishes last as per the schedule generated by the time-tabling algorithm, and is thus, responsible for the current flowspan. In presence of several critical flows, one of them is

randomly selected. The principle of our neighbourhood function is to transform critical flows of the current solution into non-critical flows, and thus, result in reduction of flowspan.

We mainly use two unary operators—*Insertion* and *Swapping*—that operate on a flow for generating the neighbourhood of the current solution with respect to the flow. The *Insertion* operator removes the critical flow from the ordering and inserts it before the operand flow. The *Swapping* operator swaps the critical flow with the operand flow while maintaining the order of all the other flows. We define the neighbourhood of the current solution as the set of all possible orderings obtained by execution of the Insertion and Swapping operators on all flows preceding the critical flow in the current ordering. Thus, the neighbourhood contains, at the most, $2(n-1)$ possible solutions, where n is the number of flows.

Solution Selection

After the neighbourhood generation, we select a new solution for the next iteration. To this end, all orderings from the neighbourhood are evaluated using the time-tabling algorithm to determine their flowspan. The ordering with the lowest possible flowspan in the neighbourhood which does not violate the tabu list, or satisfies the aspiration criterion is selected for subsequent iterations. The tabu list contains a list of flows that were identified as critical flows in previous x iterations. Solutions in the neighbourhood whose critical flow(s) lies in the tabu list are rejected even if they have a low makespan. This is primarily done to avoid the solutions at the local optimum. The aspiration criterion, however, overrules the tabu list, i.e., we ignore the tabu list and accept an ordering, if the flowspan of the solution is lower than that of the best ordering encountered so far.

We terminate our execution runs when previous y iterations of the algorithm do not yield an improvement on the best flowspan encountered till the moment. The performance of our algorithm can be tuned with different values of x and y. While higher values for x and y may yield better schedules, it would also result in increased execution times.

4.4.3 Schedule compression

This procedure of schedule compression is specifically aimed at reducing the number of gate opening events in the computed transmission schedules. Fewer gate opening events also imply a smaller schedule, and thus, lower memory requirements. Addressing memory limitations in network elements, for instance, the size of TCAM tables used to store routing information, has always been a major research goal [58].

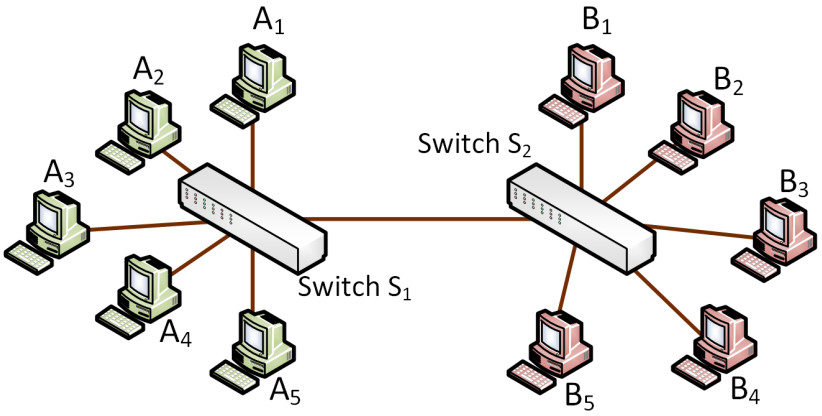


Figure 4.2: Benchmark topology with 10 hosts (A_1–A_5, B_1–B_5) connected to 2 switches (S_1 and S_2) with 10 Gbps links and 5 time-triggered flows ($F_i : A_i \rightarrow B_i$; $i \in [1 \ldots 5]$)

With respect to NW-PSP, reducing the number of gate opening events in IEEE 802.1Qbv networks is equivalent to ensuring that machines responsible for transmission are scheduled to work in fewer but longer sprints instead of several shorter bursts, i.e., a gate-open event results in transmission of several packets of scheduled traffic. Our idea of schedule compression is based on the principle that start times for certain operations may be delayed such that they end just before the succeeding operations begin on the corresponding machines. In terms of the schedules, it means that the transmission of a scheduled packet may be delayed to a time such that it is finished just before the next scheduled packet for transmission on the same port is available. Overall, this relaxes the no-wait constraint but our algorithm ensures that the flowspan from the original schedule remains unaffected. We explain our approach with a simplified example on a small topology.

Consider the benchmark topology shown in Figure 4.2. In this scenario, five time-triggered flows ($F_i : A_i \rightarrow B_i$) traverse over the link $S_1 - S_2$. The transmission of the flows over this link is a conflicting operation and cannot proceed in parallel. Figure 4.3 represents a potential schedule (not necessarily with minimal flowspan) for the time-triggered flows in the scenario. We mainly focus on the transmission operations of the flows over the link $S_1 - S_2$. The schedule shows the start times for all the operations of the flows: transmissions (denoted with subscript t) and processing (denoted with subscript p) on the switch ports and NICs. The schedule of the machine responsible for transmission on link $S_1 - S_2$ is of interest with respect to the compression algorithm. As can be seen in Figure 4.4, there are two gate opening events on the machine responsible for transmission on link $S_1 - S_2$—the first handles flow F_1 while the second handles all other flows. As shown in Figure 4.3 and Figure 4.4, the schedule can be modified to delay the transmission of packet belonging to flow F_1 such that a single gate open event can service packets from all flows in a quick succession.

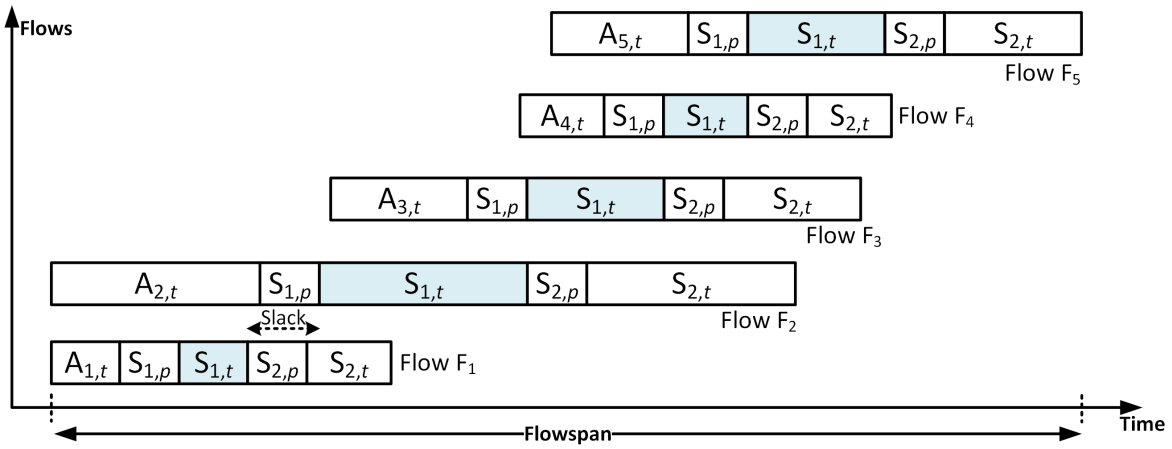


Figure 4.3: Sample schedule for the 5 flows on the benchmark topology in Figure 4.2. Here, t and p represent the time spent for transmission and processing operations on the corresponding switches or NICs, respectively. E.g., $A_{1,t}$ is the transmission delay on the NIC of source host A_1. Slack represents the possibility to delay a particular operation in order to compress the schedule.

This method may not always result in a single gate open event per link like in the presented example. Delaying the transmission of a packet of a flow impacts the starting times of the subsequent operations for the flow. It may so happen that they cannot be delayed, for instance due to impact on the schedule flowspan. Further, it must be ensured that the order in which the switch processes/transmits packets is not be altered while delaying transmission of any packet belonging to a flow. For instance, in Figure 4.4, the transmission of packet of flow F_1 on link $S_1 - S_2$ cannot be delayed such that packet belonging to flow F_2 is transmitted before it. This would violate the FIFO semantics of the queues in switches. Given these constraints, compression of schedules is not a trivial problem.

In the following, we present an efficient algorithm for compressing the schedules to reduce the number of gate opening events in the entire schedule. This algorithm can also be used for compressing schedules that were computed using methods other than the presented Tabu search algorithm, e.g., the ILP formulation introduced in Section 4.3.3. Basically, the scope for schedule compression stems from the existence of "slack" in the start times of the operations, as shown in Figure 4.4. Slack for a given operation is the amount of time by which its start may be delayed on a machine such that it is finished before the next operation is due to start on the machine. For instance, consider a machine that processes operations O_1, O_2, O_3 (belonging to different flows), each needing three time units, between times 1–3, 6–8, 9–11, respectively. Slack for the operations O_1 and O_2 is 2 and 0, respectively, i.e., the start of operation O_1 can be delayed by up to 2 time-units and must start at time $t = 3$ to be finished before O_2 is due to start, while start of O_2 cannot be postponed.

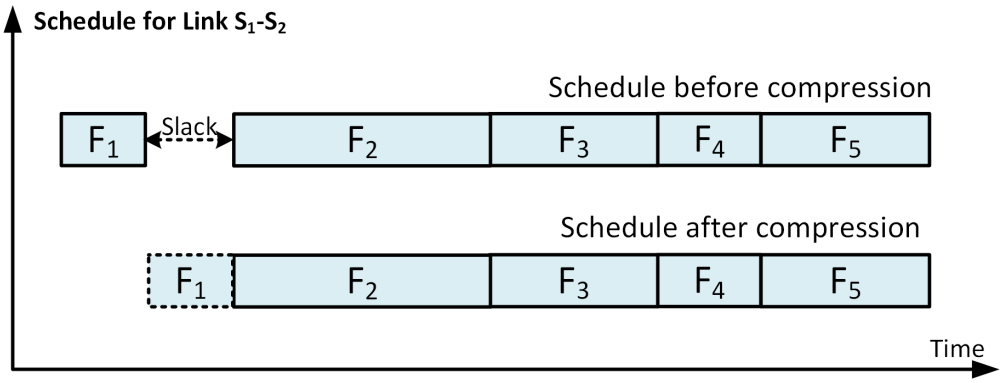


Figure 4.4: Schedule (before and after compression) for transmission on link $S_1 - S_2$. The transmission of Flow F_1 is delayed to finish just before the transmission of Flow F_2 is scheduled to start.

Delaying an operation also implies a delay for the subsequent operations of the flow. Physically, this means that a packet cannot be processed on the switch next hop before it is completely transmitted from the current switch. Thus, the amount of time by which each operation can be delayed clearly depends on the slack of all the subsequent operations of the job. Based on the slack for each operation, the amount of time by which they can be delayed is computable. For a flow with N operations, $\{O_1, O_2, \ldots, O_N\}$, and corresponding slack times $\{S_1, S_2, \ldots S_N\}$, an operation O_i is delayed by time t_i, where $t_i = min(S_i, S_{i+1}, \ldots, S_N)$. Our algorithm, summarized in Algorithm 3, computes slack for all operations of all jobs (Line 7) and based on the slack computes the times by which each of the operation may be potentially delayed (Line 12). Finally, the algorithm delays the start times of the operations based on the computed delays (Line 14). It does so iteratively till no further operations can be delayed. The modified schedules have reduced number of gate opening events compared to the original schedule prior to the compression.

The time required for each iteration of the compression algorithm is directly proportional to the number of flows (n) and the maximum forwarding operations per flow (N). During each iteration of the algorithm, the start times (after including delays from previous iterations) of at least one forwarding operation is finalized. Thus, in the worst case, the algorithm may need up to $N \times n$ iterations. Overall, the worst-case time complexity of our algorithm is $\mathcal{O}(n^2N^2)$. Given that N is bounded in our system model, the time complexity reduces to $\mathcal{O}(n^2)$. Thus, our schedule compression approach has a polynomial time complexity.

It must be noted that on account of relaxation of the no-wait constraints for schedule compression, the end-to-end latency of the time-triggered flows may increase from the minimum, i.e., the cumulative processing, propagation and transmission delays of the flows. In the worst case, a flow may suffer latency equalling the length of the scheduling cycle, t_{bp}. However, our algorithm can be easily extended to take into account the

Algorithm 3 Compression Algorithm

```
1: function COMPRESSOR(schedule)
2:     compress ← True
3:     while compress do
4:         compress ← False
5:         for each flow do
6:             for each operation do
7:                 slack ← computeSlack(schedule, flow, operation)
8:             end for
9:         end for
10:        for each flow do
11:            for each operation do
12:                delay ← computeDelays(slack)
13:                if delaying possible then
14:                    applyDelays(schedule, flow, operation, delay)
15:                    compress ← True
16:                end if
17:            end for
18:        end for
19:    end while
20:    return sched
21: end function
```

maximum end-to-end delay that a flow is allowed to incur while computing the slack of its corresponding operations.

4.5 Schedule Adherence for End systems

The programmable gating mechanism specified in the IEEE 802.1Qbv standard enables enforcement of the transmission schedules for time-triggered traffic on the switches. Interestingly, the TSN standards do not specify special mechanisms for the end systems to adhere with the schedules. Furthermore, the aforementioned gating mechanism is a time-aware mechanism which is oblivious to the traffic in the queues, i.e., the gates cannot be operated based on the packets which are in the queues of its ports. Hence, it is imperative that the source hosts of time-triggered traffic stick to the transmission schedules precisely for adhering with the programmed schedules at the switches. Deviations at the source host will render the entire schedule useless. Given that we use only one queue for handling scheduled traffic, transmission of a packet too early from its source host may block the switch ports for traffic scheduled to traverse the port at that point of time. Similarly, too late a transmission may also result in the closure of the gates at some switch port leading to blocking of the packet and all the scheduled

traffic to follow.

Algorithm 4 Source - Userspace DPDK application

```
1: function SRC(basePeriod(bp), transmissionTime(tt))
2:     init NIC and sending queues
3:     intervalAlarm ← flowPeriod
4:     firstAlarm ← now() + (bp − now() % bp) + tt
5:     firstAlarm ← firstAlarm − pktCreationTime
6:     timer_settime(firstAlarm, intervalAlarm)
7:     while True do
8:         if alarm is triggered then
9:             Create payload by executing required tasks
10:            pkt ← dpdk.createPacket()
11:            dpdk.sendPacket(pkt)
12:        end if
13:    end while
14: end function
```

We evaluated the socket API's in Linux (CentOS, kernel version 3.10) to determine if they are suitable for usage in source hosts of time-triggered traffic. For our evaluations, we deployed two userspace applications which act as source and destination of time-triggered traffic on nodes A_i and B_i, respectively, of our benchmark topology (cf. Figure 4.2). This topology was created using five commodity machines (Intel Xeon E5-1650), each equipped with an Intel XL710 quad 10 GbE network interface, and an Edge-Core cut-through bare-metal switch (AS5712-54X) running PicOS (ver 2.6.1). The switch was partitioned into virtual switches to create the topology, while each machine hosted two end systems, for instance, Host A_1 and B_1 were placed on the same machine but used different network interfaces. This enabled us to measure end-to-end latencies (between the source and the destination application) experienced by the packets of these flows without synchronizing clocks across the five hosts. It may be noted that the switch we used in our evaluations for schedule adherence did not support the gating mechanism specified in the IEEE 802.1Qbv standard. However, as these evaluations were only aimed at determining the precision with which the source hosts can adhere with the schedule, the lack of scheduling capabilities in the switches played no significant role in these evaluations.

We measured the end-to-end latency for 10,000 packets (each of size 1500 bytes), one packet sent every 10 ms. Given that processing delay of the switches is about 0.7 μs, one may expect a network latency of about 4–5 μs in absence of any queuing or cross

traffic. However, the results (cf. Figure 4.5) show the latency varying between 37–117 μs with an average latency of 63.58 μs and a standard deviation of 4.88 μs with the usage of socket APIs. Such high jitter is attributed to the variable delays (10–100 μs) that packets incur while traversing the network stack of the operating system [59], i.e., invoking *send()* on a socket does not place the packet on the network interface with deterministic delay, nor does *receive()* return with bounded delay after the network interface receives a packet. These non-deterministic delays incurred in the network stack of end systems make it impossible to use socket APIs to adhere to the transmission schedules with high precision.

High throughput packet processing frameworks, like Intel's Data Plane Development Kit (DPDK) [33] or netmap [34], bypass the network stack by using custom device drivers and hand the complete control of communications to userspace applications. These may be used to get around the problem of variable delays in the network stack of the end systems. To evaluate the feasibility of using these frameworks, we developed two DPDK applications, one as the source and the other as the destination of time-triggered traffic and measured the end-to-end latency between them, similar to our evaluation of socket applications. The destination application simply receives the packet from the network interface bypassing the network stack and parses the packet to decode the information sent by the source. DPDK provides high performance packet processing API's for this purpose. The source application (pseudo-code in Algorithm 4) plays an important role with respect to the transmission scheduling. It is responsible for configuring timers suitably to trigger packet transmissions. For this we used Linux interval timers (*timer_settime()*) that generate an alarm at fixed intervals based on the base-period (Lines 3–5). The source host can use the generated alarm for transmitting the time-triggered packet prepared beforehand, or use it as a trigger to also create the packet (generate the payload by executing the sensing or the control tasks of a cyber-physical system). We use the latter approach (Lines 9–10) and hence advance the interval timer by *pktCreationTime* (profiled beforehand) to compensate for the time required to generate the payload and create the corresponding packet.

With DPDK API's, the latency incurred by the packets varied in a narrow band between 7–10 μs with an average latency of 7.94 μs and a standard deviation of 0.4 μs (cf. Figure 4.5). Overall, the performance of the aforementioned packet processing frameworks is up to an order of magnitude better in comparison to the socket APIs. The low end-to-end latency between the source and destination applications indicate that packets are placed on the network interface with minimal delay after the corresponding API is invoked. Furthermore, though the packets are slightly delayed in the DPDK stack, the delays are almost constant making it suitable for usage in the source hosts of time-triggered flows.

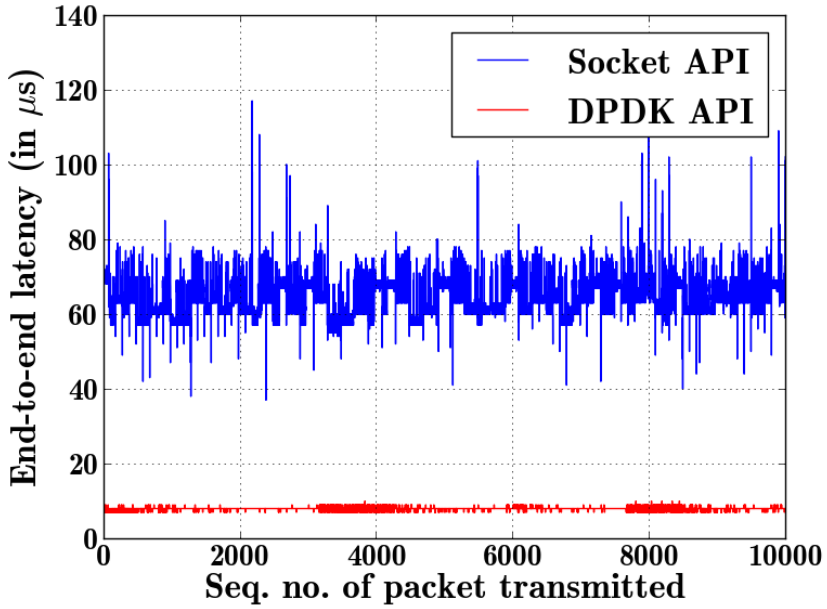


Figure 4.5: Intel's DPDK versus Sockets

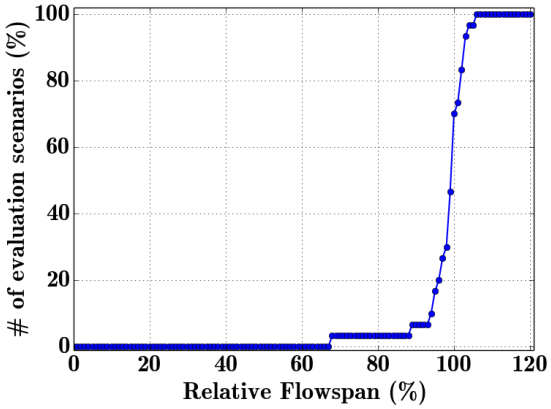
4.6 Evaluations

We present the evaluation results of our scheduling approach for networks compliant with the IEEE 802.1Qbv standards in this section. We evaluated our solutions in various scenarios (randomized topologies with a set of time-triggered flows randomly generated) to determine the quality of flow schedules they compute (in terms of flowspan), their scalability and the number of gate-opening events they result in. It must be noted that, as described in Section 4.4, the performance and the worst-case execution times for our approaches do not depend on the type or size of the topologies but rather on the number of flows being scheduled. Nonetheless, we used various models of randomized graphs (Erdős-Rényi (ER) model [60], random regular graphs (RRG) model, and the Barabási-Albert (BA) model [61]) created using NetworkX [62], a Python based library for handling complex networks, to ascertain this.

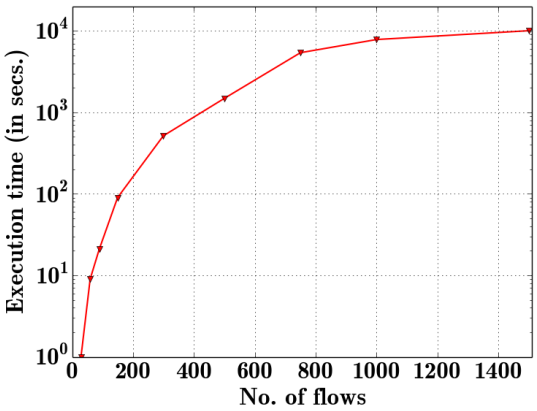
4.6.1 Qualitative Evaluations

In our evaluations, we determine how closely do the schedules computed using the presented approach approximate the optimal schedule in terms of the flowspan. For this we compared the schedules computed using our approach to the ones generated by an ILP solver which solves the ILP formulation of the corresponding problem (cf. Section 4.3.3).

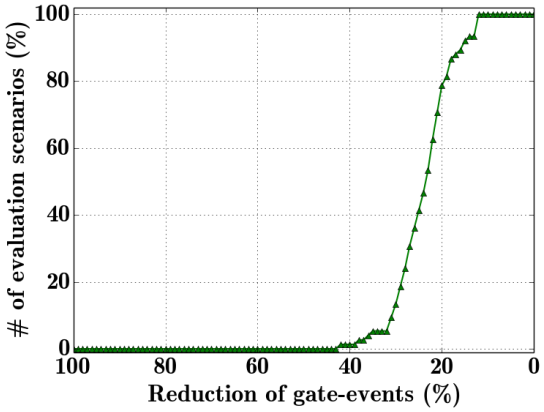
The NW-PSP, an NP-hard problem, however cannot be optimally solved in a reasonable time frame by an ILP solver if the problem instance is not small enough. Our



(a) Tabu search v/s ILP solution



(b) Execution time



(c) Impact of schedule compression

Figure 4.6: Evaluations Results

initial attempts to solve small instances with up to 50 flows using CPLEX [63], a state-of-the-art commercial ILP solver from IBM, required over three days to compute optimal solution. However, we observed that the solver quickly generates a feasible solution and improves it subsequently. In a few cases, the best solution computed by CPLEX after a few minutes turned out to be the optimal solution, though it took CPLEX a few hours to confirm it. Hence, CPLEX provides primitives to terminate the optimization problem early and obtain the best solution that has been computed till that moment along with the optimality bounds for the solutions, i.e., how does this solution potentially compare with the optimal solution. For our evaluations, we set time-bounds on CPLEX to terminate after a reasonable amount of time and provide the best solution obtained till the moment. We compared this solution with the schedule computed using our Tabu-search approach, in terms of the flowspan.

We executed our evaluations in 30 scenarios on topologies of various sizes (24–100 hosts, 5–20 switches generated using ER, RRG and BA models) with varying number

of flows (30–1500). We configured CPLEX with an upper time-bound of 300 min. for solving the ILP formulation for each instance of NW-PSP. Correspondingly, we also computed schedules using our approach based on Tabu search for the same instance. Our evaluations show that restricting the length of the Tabu list, x, to 10 % of the number of flows being scheduled in the scenario and terminating the execution runs if the previous 10 iterations did not yield an improvement in the solution, i.e., setting $y = 10$, result in solutions with a flowspan reasonably close to the flowspan of the solutions computed using ILP solvers with a time restriction.

For each scenario, we calculated the relative flowspan, i.e, the ratio of flowspan of the schedule calculated by our Tabu search heuristic to the flowspan of the schedule computed by the ILP formulation. Figure 4.6a shows a cumulative distribution of the computed relative flowspans. Overall, in over 70 % of our scenarios, the Tabu search computes schedules with flowspan equal to or lower than the ones computed by a state-of-the-art ILP solver (restricted to run for 300 minutes or less). In about a third of the scenarios, the solutions computed by the Tabu search had a flowspan slightly higher than the ones computed using the ILP. However, in these cases, the difference in the flowspan of these schedules was less than 5 %. Overall, the average relative flowspan for the set of scenarios we evaluated was $\approx$ 97 %. Thus, the Tabu search computes solution which, on an average, have lower flowspan than the ones computed by an ILP solver with a restriction on execution time.

4.6.2 Scalability Evaluations

To evaluate the scalability of our approach, we measured the execution times of our algorithm (Tabu search along with schedule compression) while computing schedules for varying number (10–1500) of flows. We executed our evaluations on a multi-processor machine (Intel(R) Xeon(R) CPU E3-1245 V2 @ 3.40GHz) with 2×4 cores and 16 GB of memory.

These evaluations are summarized in Figure 4.6b. Up to 50 flows, the scheduling using the Tabu search approach takes less than 10 secs., while beyond that the execution times increase polynomially, as expected, with the number of flows. Overall, the Tabu search can compute schedules for about 1500 flows in about 3.2 hrs. Moreover, our evaluations suggest that the execution times for the Tabu search are mainly a polynomial function of the number of flows being scheduled. They do not depend on the size of the underlying topology or the model on which the topology is based.

4.6.3 Impact of Schedule Compression

Compression of flow schedules, as we presented them, is an important aspect of scheduling in networks with IEEE 802.1Qbv enhancements. To evaluate the effectiveness of

our approach, we measured the percentage reduction of gate opening events for time-triggered traffic in the schedules after compression.

For this evaluation, we computed flow schedules in over 75 scenarios (varying sizes and models of topologies, varying number of flows etc.). The computed schedules were subjected to our compression algorithm for reducing the number of gate opening events. We not only used the presented Tabu search algorithm but also ILP solvers for computing the initial schedules to show that our schedule compression algorithm is equally effective on schedules computed using different methods. The cumulative distribution function for the reduction in number of gate opening events is shown in Figure 4.6c.

Our evaluations show that the schedule compression algorithm, reduces the number of gate-opening events by at least 12 %. In certain scenarios, the achieved reduction goes up to 42 %. Overall, on an average, we observed a reduction of 24 % for gate-opening events in compressed schedules compared to the original schedules.

4.6.4 Evaluation Summary

In total, our evaluations show:

1. The presented Tabu search algorithm calculates near-optimal solutions for NW-PSP with respect to minimizing the flowspan.
2. The execution times for the presented Tabu search algorithm depends on the number of flows being scheduled. Overall, our approach scales to scheduling over 1500 time-triggered flows in about 3 hours.
3. The presented algorithm for schedule compression results in an average reduction of 24 % for gate-opening events/guard bands.

4.7 Discussion

4.7.1 Optimization Goal - Flowspan Minimization

Our approach to computation of transmission schedules for networks compliant with programmable gating bunches up the scheduled traffic towards the beginning of the scheduling cycle. This increases the chances that packets from different time-triggered flows are scheduled back-to-back resulting in a reduction in the number of gate driver entries required for the computed schedule. Moreover, the proximity of the flows in the schedule makes the schedule compression algorithm, presented in Section 4.4.3, rather effective.

An important effect of our optimization goal is the presence of larger continuous space within a scheduling cycle for best-effort traffic where these packets can be transmitted

without interruption, i.e., these schedules may lead to formation of traffic bursts. The exact effects of such schedules on best-effort traffic, in particular the congestion and flow control mechanisms of TCP, need to be studied as a part of future work.

4.7.2 Incremental Scheduling in Networks with IEEE 802.1Qbv Enhancements

The IEEE 802.1Qbv standard specifies mechanisms for runtime updates of the gating schedules in the switches. In order to make the scheduling changes throughout the network atomic in nature, the switches can be programmed with new schedules along with a future timestamp specifying when they come into force. This way the schedule changes can be synchronized throughout the network. However, the challenge remains as to how to compute incremental schedules, i.e., ensure that a new time-triggered flow to be scheduled in the network does not influence the schedules of the existing flows.

For this, we exploit an important property of the presented time-tabling algorithm. The starting time of flows, as computed by the algorithm, is influenced only by the flows preceding it in the totally ordered set generated using the sequencing algorithm. The ones succeeding it play no role in determining its starting time. Thus, for adding new time-triggered flows in the schedule, the new flows can be appended to the current ordering of flows and the time-tabling algorithm can be re-executed. While the new sequence may not be an optimal one (in terms of the schedule flowspan), the schedules for existing flows will never be altered.

While the process of addition of flows to an existing schedule is limited to re-execution of the time-tabling algorithm, removal of flows throws a bigger challenge. Removing flows and then re-executing the time-tabling algorithm will effect all the flows which appear in the sequence after the removed flow. To avoid affecting other flows, additional gating entries can be added to close and re-open the gate appropriately during the time when the packets of flow are supposed to traverse a switch port. However, this will lead to a gradual fragmentation of schedules on the lines of heap fragmentation in memory management. Additional concepts required to overcome these effects are to be handled in future work.

4.7.3 Extension for Unsynchronized Hosts

Our scheduling approach for networks compliant with IEEE 802.1Qbv standard assumes that the clocks of all network participants are synchronized. However, with minor extensions this scheduling approach can also be applied in scenarios where clocks of one or more end systems are not synchronized with the rest of the network. For such end systems, the switches they are connected to act as a synchronizer taking over the responsibility of synchronizing the transmission of the flow into the network.

This implies that the schedules of time-triggered flows whose source hosts are not synchronized with rest of the network are computed from the switch to which the source hosts are connected to the destination of the flow, i.e., the schedules are computed from the second hop onwards. The lack of scheduling at the first hop means that the jitter incurred at this hop may be as high as the base-period. Furthermore, to synchronize the transmissions of the flow at the second hop, the switch must dedicate one queue at the corresponding output port to store the packets of the flow till the time that they can be transmitted over the second hop. Thus, the capacity of a switch to support unsynchronized end systems acting as source hosts for time-triggered flows is limited by the number of queues available for usage at the output ports of the switch. However, with respect to the scheduling algorithm itself, the fact that the end system is not synchronized does not constitute a significant limitation.

4.8 Related Work

Computing schedules for time-triggered traffic in real-time networks, like TT-Ethernet, ProfiNET, etc., is reasonably addressed in the literature. Given the complexity of the problem, most approaches including the one that we have presented in this chapter separate the scheduling problem from the routing aspect.

The pioneering work in this direction comes from Wilfried Steiner who used Satisfiability Modulo Theories (SMT) to model the scheduling problem of time-triggered flows in TT-Ethernet for computing a (any) feasible transmission schedule [54]. This approach has also been extended to compute network transmission schedules along with task schedules for the tasks executing on the end systems [64] [65]. TT-Ethernet is also a layer-2 multi-hop switched Ethernet network architecture that uses an internal buffer to store packets belonging to time-triggered flows which are then put into the queues of the output ports at specified times based on the flows they belong to. Scholer et al. present SMT based scheduling approaches for schedule computation in general time-triggered networks [66]. However, these approaches cannot be used without further adaptations for networks with the programmable gating mechanism which is oblivious to the flows to which the packets in the queues belong [67].

Hanzalek et al. modelled the scheduling problem in ProfiNET as a Resource Constrained Project Scheduling (RCPS) problem to minimize the makespan of the computed schedule [68]. Dvořák et al. also mapped the concept of makespan minimization for TTEthernet [69]. However, these approaches are specifically directed towards ProfiNET, TTEthernet etc., and does not discuss with the problems like guard bands which are prevalent in the gating mechanisms specified under the IEEE 802.1Qbv mechanism.

Recently, a few approaches which combinedly explore the routing and scheduling in time-triggered networks have been published. Schweissguth et al. model the combined

routing and scheduling problem using ILP and then leverage ILP solvers to compute routes and schedules for the time-triggered flow [70]. Falk et al. presented ILP formulations for computing routes and schedules simultaneously and determined the factors which directly influence the runtime of the ILP solver [71]. Mahfuzi et al. have taken this problem a step further and integrated the stability criterion of the cyber-physical systems relying on the time-triggered flows into the problem [72]. However, neither of these approaches scale well for larger problem scenarios, i.e., network topologies with a large number of flows.

We also present related work for computation of schedules considering other classes of traffic in time-triggered networks. Maxim et al. [73] and Zhao et al. [74] investigate the effect of schedules for time-triggered traffic on other traffic classes in the networks with IEEE 802.1Qbv extensions and TT-Ethernet, respectively. In [75], Steiner proposed creation of porous schedules for reserving enough bandwidth of other traffic classes in TT-Ethernet, thus, resulting in networks for supporting systems with mixed criticality. Further, [76] proposes a Tabu search algorithm for adapting schedules of time-triggered traffic such that deadlines for different traffic classes can be satisfied. Specht et al. introduce an asynchronous traffic class in the network and by means of an urgency based scheduler provide low and predictable worst-case delays at high link utilization [77].

Our approach along with [30] are among the first approaches which deal with the computation of transmission schedules specifically for the gating mechanisms introduced in the IEEE 802.1Qbv standard. Recent work by Oliver et al. uses first order array theory encoding for formulating constraints for an SMT model describing the scheduling problem [78]. While most approaches look for finding a feasible solution for deployment in the network, our approach aims to optimize the makespan of the schedule with a goal to reduce bandwidth wastage under guard bands by reducing the number of "gate-open" events for scheduled traffic in the schedule. Further, we also look to improve the scalability of our approach for larger scenarios by means of our Tabu search heuristic.

4.9 Summary

In this chapter, we presented the problem of scheduling time-triggered traffic in networks compliant with the recently standardized IEEE extensions for handling scheduled traffic in Ethernet networks. We modelled this problem as the No-wait Packet Scheduling Problem (NW-PSP) which can be reduced from a No-wait Job-shop Scheduling Problem (NW-JSP), a well-known problem from the field of operations research. Our contributions include an efficient meta-heuristic in the form of a Tabu search algorithm to compute schedules by solving the corresponding NW-PSP instance. To further reduce the wastage of bandwidth due to guard bands stemming from the usage of the gating mechanism, we presented a specialised compression algorithm that compresses the schedules to reduce the instances of guard bands in schedules. We also discuss extensions of our approach to incrementally add time-triggered flows into the network

and the usage of the presented approach in networks where one or more end systems are not synchronized.

CHAPTER 5

ROUTING IN NETWORKS WITH IEEE 802.1QBV EXTENSIONS

5.1 Introduction

On account of the high time complexity for computing per-hop transmission schedules, we separate the scheduling of time-triggered flows from their routing in networks equipped with extensions from IEEE 802.1Qbv standard. Our approach to the computation of the gating schedules for a set of time-triggered flows require that the routes of the flows are provided in advance. However, given that the routing of time-triggered flows directly influences the computed schedules, the question arises as to how to route time-triggered flows such that it offers the scheduling algorithm the best chance to compute a feasible schedule, if one exists.

Time-triggered flows can be routed using the same of-the-shelf algorithms that are also used for routing best-effort traffic, like shortest path routing or equal-cost multipathing (ECMP), which minimize the number of hops over which the traffic is routed. While fewer number of hops may translate to lower end-to-end latencies for time-triggered flows, it does not reflect their schedulability in the network. The possibility of explicitly controlling the routing of traffic using software-defined networking protocols like OpenFlow (cf. Section 2.1) or the amendments specified in the IEEE 802.1Qca standards enable design of routing algorithms which also incorporate the schedulability aspect of time-triggered flows.

In this chapter, we explore the impact of routing algorithms on the schedulability of time-triggered flows which we express in terms of the slackness in the transmission schedule. In particular, we show that routing of time-triggered flows does indeed affect its schedulability and that the algorithms which use number of hops as the only

metric for routing the traffic are not suitable for time-triggered flows. Our scientific contribution is the identification of parameters that a routing algorithm must consider to have an impact on the slack of the subsequently computed schedules for time-triggered flows. Moreover, we use these parameters to design ILP-based algorithms for routing time-triggered flows, and, thus, improve their schedulability. Our evaluations show that these routing algorithms improve the slack in the schedules by up to 60 % and 30 % in comparison to our benchmark algorithms the shortest path routing and equal-cost multipathing, respectively.

This chapter is structured as follows. We present the system model and identify metrics required to be considered with respect to the routing of time-triggered flows in Section 5.2. Section 5.3 presents the ILP-based routing algorithms for time-triggered flows based on the identified metrics. We discuss the evaluations of our algorithms and present the related work in Section 5.4 and 5.5 respectively before concluding.

5.2 System Model & Problem Statement

5.2.1 System Model

As we focus on the routing algorithms for time-triggered flows in networks compliant with the IEEE 802.1Qbv standard, the system model in this chapter is similar to the one presented in Section 4.2. It consists of switches equipped with the programmable gating mechanism (cf. Section 2.2.1), end-systems, and a centralized network controller. The end systems function as the sources and sinks of time-triggered traffic, while the network controller is responsible for computing the routes and schedules for the traffic and programming the underlying switches accordingly. We assume that all the nodes and the switches in the network are precisely synchronized and that the hosts can adhere with the computed schedules reasonably. Time-triggered flows are modelled as periodic flows with constant time-periods and payload size. Additionally, the time-periods of all time-triggered flows to be scheduled are restricted to be integral multiples of the base-period, t_{bp}.

To implement the results of the routing algorithm, we require that the switches provide programmatic interfaces (e.g. OpenFlow [14] or mechanisms from IEEE 802.1Qca [27]) which enable routing of time-triggered flows over arbitrary network paths.

5.2.2 Problem Statement

To concretely specify the routing problem we have at hand with respect to time-triggered flows, we need to first understand the impact of routing on the subsequent computation of schedules for such flows. To study this impact, we need metrics by means of which different transmission schedules for a given set of time-triggered flows can be compared.

Comparison of Transmission Schedules

In Chapter 4, we formulated the No-wait Packet Scheduling Problem (NW-PSP) to compute gating schedules for time-triggered flows in networks compliant with IEEE 802.1Qbv standard and mapped it to the No-wait Job-shop Scheduling (NW-JSP), a problem in operations research dealing with the computation of schedules for manufacturing jobs in a shopfloor. NW-PSP is an optimization problem that deals with minimization of flowspan, T_{fs}, of transmission schedules, where the flowspan of a schedule is the total time required in a scheduling cycle for handling all the time-triggered flows in the network. The schedule itself is of the length equalling the base-period, t_{bp}, and is cyclic, i.e., it is repeated from beginning after it ends.

The flowspan of the schedule for a given set of time-triggered flows in a network is critical with respect to their schedulability. Here, schedulability implies that the computed schedule satisfies the timing constraints of all the time-triggered flows in the network. For this, it is necessary that the flowspan is less than or equal to the length of the schedule, i.e., $T_{fs} \leq T_{bp}$. If the flowspan exceeds the length of the schedule, then the traffic from a given scheduling cycle may interfere with the traffic from the subsequent cycle leading to violations of the timing guarantees. Furthermore, the flowspan of the schedule reflects the capacity of the network to accommodate additional time-triggered flows, i.e., the slack in the schedule. The lower the flowspan, higher is the slack, and higher is the quantum of time-triggered traffic that can be additionally accommodated in the network. Thus, if multiple transmission schedules are available for a given set of time-triggered flows in a network topology, the one with the lowest flowspan is preferable. In the following, we use flowspan as a criterion to determine the "quality" of the transmission schedules for a given set of time-triggered flows in a network.

While the concept of flowspan lends itself for determining the quality of schedules computed by the formulation of an NW-PSP instance, it is not very meaningful for other approaches computing gating schedules. The approaches using Satisfiability Modulo Theories (SMT) compute schedules based on the concept of hypercycles, where the length of schedules are equal to the least common multiple of the time-periods of all the time-triggered flows to be scheduled. These approaches do not constrain the transmission period of the flows in any way, and distribute the scheduled traffic throughout the length of the schedule in compliance with their timing constraints. As these approaches do not strive to tightly bunch the time-triggered traffic at the beginning of a scheduling cycle, it is hard to compare multiple feasible schedules or determine the available slack in the schedule for additional time-triggered traffic to be added. Given the widely varying approaches to compute transmission schedules, it is a hard task to identify a universal metric that is valid for all scheduling approaches to compare schedules. We, nonetheless, choose flowspan as a metric for comparing schedules for this work, and leave the task of identifying a universal metric for schedule comparison as future work.

Impact of Routing on Schedulability of Time-triggered Flows

As production processes dictate the sequence of operations composing the manufacturing jobs in NW-JSP, reordering or modifying the sequence is usually not feasible even if it may lead to better schedules. In contrast, for NW-PSP the constituent forwarding operations of the flows can be modified to a certain extent, especially, if it results in a lower flowspan. In other words, the routes of time-triggered flows may be modified to minimize the flowspan of the resulting transmission schedule. So, we analyse the impact of routes serving as an input parameter to the NW-PSP on the flowspan of the resulting schedules.

The standard algorithms (like shortest path first, equal cost multipathing etc.) optimize the number of hops over which the traffic is routed. In terms of the NW-PSP formulation, the flows to be scheduled are composed of fewest possible forwarding operations. However, flows with fewer forwarding operations alone do not guarantee an optimal flowspan. In NW-PSP, the flowspan is also dependent on the number of flows with conflicting forwarding operations, i.e., the forwarding operations belonging to different flows that must be processed on the same egress ports of the switches, and the maximal duration during which any of the switch ports is kept occupied. Hence, time-triggered flows must be routed to have fewer flows with overlapping paths, while also minimizing the aggregated amount of time-triggered traffic that any switch port should transmit. For this purpose, we introduce a parameter, Maximum Scheduled Traffic Load (MSTL), for capturing the effects of distribution of scheduled traffic over the network. We define Maximum Scheduled Traffic Load (MSTL) as the highest quantum of scheduled traffic that is transmitted by any of the switch port in the network per cycle of the transmission schedule. MSTL is directly influenced by the routing algorithm, for instance, routing all time-triggered traffic over any single link leads to a higher value of MSTL, compared to the case where this traffic is distributed throughout the network.

To determine the efficacy of our metric, we conducted preliminary evaluations to determine the relationship between MSTL stemming from routing of time-triggered flows and the flowspan of the resulting schedule. In our evaluations, we observed that the MSTL on account of routing time-triggered flows and the flowspan of the resulting schedule are directly related. Figure 5.1 shows the variance of MSTL (using shortest path routing (SP) and equal-cost multi-pathing (ECMP)), and the resulting schedule flowspan against a varying number of time-triggered flows (200–1000), each sending packets of varying sizes (300–1500 bytes) per scheduling cycle, in an Erdős-Rényi (ER) topology [60] with 10 switches and 50 hosts. The figure clearly shows similar behaviour for the schedule flowspan and the corresponding MSTL.

We argue that to have schedules with lower flowspan, it is necessary that the preceding routing stage routes time-triggered flows accounting for the corresponding MSTL. In the following, we present ILP-based algorithms that explicitly minimize the MSTL

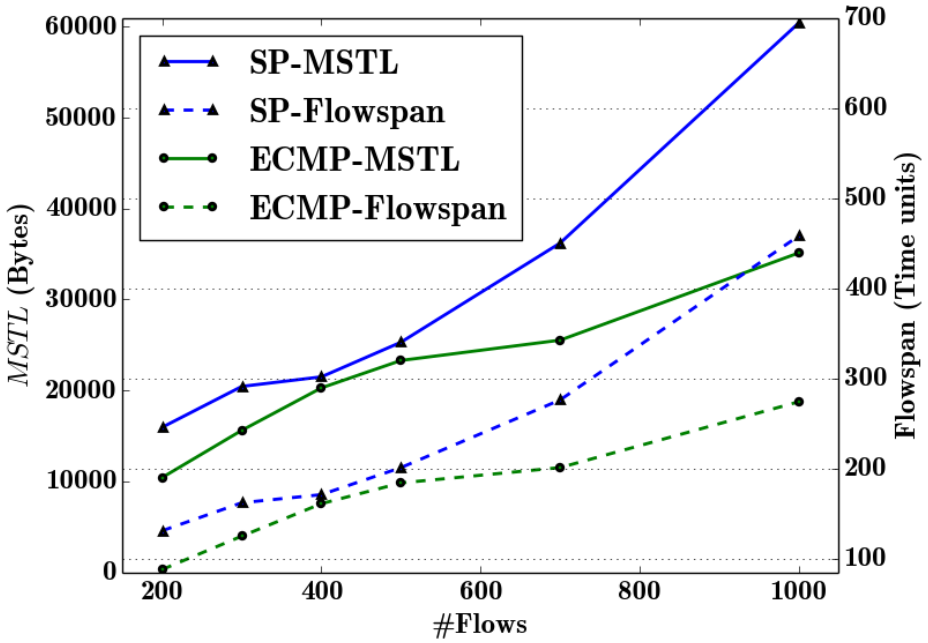


Figure 5.1: MSTL vs Flowspan for Shortest Path Routing (SP) and Equal Cost Multipathing (ECMP)

while routing time-triggered flows. Furthermore, given that there are different methods to compute gating schedules for IEEE 802.1Qbv compliant networks, we also seek to generalize this routing approach to make it compatible with the subsequently used scheduling method.

5.3 ILP Based Routing Algorithms

In this section, we present our ILP-based algorithms for determining routes for time-triggered flows.

5.3.1 Terminologies

We denote the network as a directed graph, $G \equiv (V, E)$, where V is the set of all nodes (hosts and switches) in the network, while $E \subseteq V \times V$ is the set of edges connecting a pair of nodes. Time-triggered flow is denoted as a tuple, $f \equiv (src_f, dst_f, size_f, period_f)$, which implies that the source host, src_f, sends packets with a total aggregated size of $size_f$ bytes to the destination host(s), dst_f, every $period_f$ time units.

The length of the schedule is denoted as T_{cycle} and is based on the scheduling approach to be used. For schedule computation using NW-PSP, $T_{cycle} = T_{bp}$, while for the SMT-based approaches, it is equal to the least common multiple of the time-periods of all the time-triggered flows being scheduled.

5.3.2 Routing Heuristics

We mainly present two ILP-based routing approaches, the first optimizes the routes of the time-triggered traffic based on the resulting MSTL only, while the second one additionally considers the number of hops over which the flows are routed.

MSTL Based Routing

In this heuristic, we solely base the routing decision on the resulting *MSTL*.

The inputs for this ILP are:

(a) Network topology, G,

(b) Set of time-triggered flows to be scheduled, $F \equiv \{f_1, f_2, \ldots f_n\}$.

The variables used for this ILP are:

(a) Route allocation, $Routes \equiv \{r_{i,j}\}\ \forall i \in F, j \in E$.
Here, $r_{i,j} = 1$, if flow i is routed over link j, else 0. The values of these variables, basically, determine the routes for the flows,

(b) Maximum scheduled traffic load, *MSTL*. This variable is used in the objective function. It must be noted that we do not set a value for the *MSTL* upfront, instead we allow the solver to route the flows while minimizing it,

(c) Destination counters, $DC \equiv \{d_{i,j}\},\ \forall i \in F, j \in E$.
Here, $dc_{i,j}$ = number of destinations of flow i reachable over link j, if flow i is routed over link j, else 0. These are auxiliary variables for handling multicast time-triggered flows.

The objective of this ILP is, thus, to minimize *MSTL*, subject to:

(a) The route for each flow starts at its source and ends at its destination(s). i.e., the number of destinations reachable over the outgoing links of the source host is equal to the number of destinations of the flow, while the number of destinations reachable over the incoming links of the destination hosts is 1. For all the other nodes in graph G, the sum of destinations reachable over incoming links is equal to the sum of destinations reachable over outgoing links. The below constraints are applicable for all flows, i.e., $\forall i \in F$.

$$\sum_{j \in \text{in}(src_i)} dc_{i,j} = 0 \qquad \sum_{j \in \text{out}(src_i)} dc_{i,j} = |dst_i| \tag{5.1}$$

$$\sum_{j \in \text{in}(n)} dc_{i,j} = 1 \qquad \sum_{j \in \text{out}(n)} dc_{i,j} = 0 \qquad \forall n \in dst_i \tag{5.2}$$

$$\sum_{j \in \text{in}(n)} dc_{i,j} = \sum_{j \in \text{out}(n)} dc_{i,j} \qquad \forall n \in V \setminus (\{src_i\} \cup dst_i) \tag{5.3}$$

Here, the *in*() and *out*() functions return the incoming edges and outgoing edges of the node passed as parameter, respectively.

(b) The flows must be routed such that no switch port is transmitting more scheduled traffic than stipulated by *MSTL* (which is being minimized in the objective).

$$\sum_{\forall i \in F} r_{i,j} \cdot size_i \cdot \frac{T_{cycle}}{period_i} \leq MSTL \qquad \forall\, j \in E \tag{5.4}$$

Though NW-PSP restricts the periods of the flows to be integral multiples of T_{bp}, in practice, the scheduling algorithm schedules all the flows assuming that their period is equal to T_{bp}. Thus, for NW-PSP this constraint simplifies to consider the size of the flow only, as $T_{cycle} = T_{bp}$ for an NW-PSP instance. Nonetheless, we account for the time-periods of the flows to enable the usage of this ILP formulation for the SMT-based scheduling approaches also.

With minor modifications to the constant terms in the constraint, it can be extended to also account for the differences in the data-rates of different links in the network.

(c) As an auxiliary constraint, it is required that the routing variables are inline with the destination counter. These variables must be related as follows.

$$r_{i,j} \cdot |dst_i| \geq dc_{i,j} \qquad \forall\, i \in F, \forall\, j \in E \tag{5.5}$$

After solving this ILP, the routes for the time-triggered flows can be derived from the values of the ILP variable, *Routes*. It must be noted that the computed routes for the flows may have loops resulting from links that handle scheduled traffic much lower than the *MSTL*. These loops can be removed by means of post processing the routes or adding constraints to the objective function to constrain the solver from routing flows over paths with loops. However, our evaluations show that such modifications to the ILP lead to a significant increase in the execution runtimes. Hence, we chose to post process the ILP solution to remove any loops in the final routes. The post processing of the solution in no way alters the resulting *MSTL*, as it is already minimized by the solver.

MSTL+Hops Based Routing

Routing of time-triggered traffic minimizing the *MSTL* only, may result in some flows being routed over longer paths. In some cases, this may lead to an increase in the

flowspan, as a few flows in the NW-PSP instance would now have an increased number of forwarding operations due to the longer routing of the flows. Hence, we now extend the aforementioned ILP to compute routes for time-triggered traffic, while optimizing the number of hops along with the resulting *MSTL*.

For this, we modify the objective of the aforementioned ILP as follows.

Minimize:

$$\frac{MSTL}{1 + \sum\limits_{\forall i \in F} size_i} + \frac{\sum\limits_{\forall i \in F} \sum\limits_{\forall j \in E} r_{i,j}}{1 + (|F| \cdot |E|)} \tag{5.6}$$

The first term of the objective is, basically, minimizing the *MSTL*, while the second term minimizes the cumulative number of hops over which all the flows are routed. Both these terms are normalized to limit their contribution to the objective function to less than 1, so that the ILP solver does not prioritize one over the other.

Our evaluations show that the presented algorithms do, indeed, reduce the flowspan for the schedules computed using NW-PSP.

5.3.3 Extension for Incremental Scheduling

We have so far considered the problem of routing time-triggered flows when their specifications are known in advance, i.e., the static case. In the dynamic scenario, the routes of the already scheduled flows cannot be altered while computing the routes for the new flows.

The presented ILP-based formulations can be extended for incremental routing of time-triggered flows with a minor modification to the constraint 5.4. The modified constraint considers the load of the scheduled traffic already routed over each of the link. The modification mainly includes addition of a constant term to the constraint, and hence, does not affect the time complexity of the problem.

$$\sum_{\forall i \in F} r_{i,j} \cdot size_i \cdot \frac{T_{cycle}}{period_i} + \text{load}_j \leq MSTL \qquad \forall\ j \in E \tag{5.7}$$

With this modification, the ILP formulations can route multiple flows in a batch while considering the routes of all the previously scheduled flows. However, it must be noted that the resulting MSTL on account of the routing may be sub-optimal, primarily due to the inability to modify the routes of the flows already scheduled.

5.4 Evaluations

In this section, we present the results of the evaluations of the ILP-based routing algorithms with respect to their impact in improving the transmission schedules and their scalability.

5.4.1 Impact on Scheduling

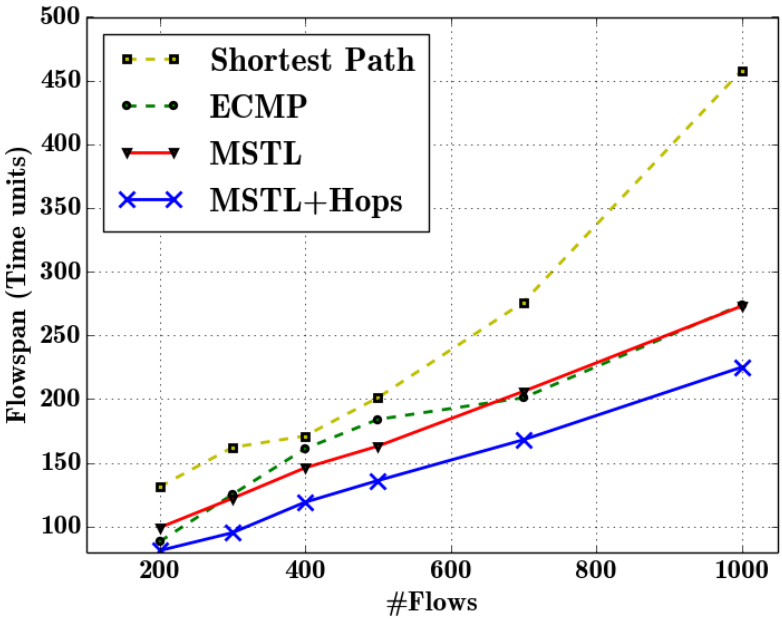
We evaluated the impact of these routing algorithms on the computed schedules with two scheduling approaches, viz., by formulating an NW-PSP instance and by using SMT-based implementation from [30].

Scheduling with NW-PSP

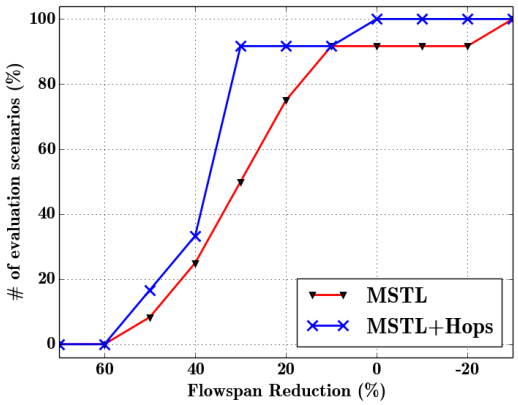
To evaluate the impact of the routing algorithms on the subsequent transmission scheduling, we routed a varying number (200–1000) of time-triggered flows in a random network topology consisting of 50 hosts and 10 switches, generated using the Erdős-Rényi model, using four different routing schemes—Shortest path routing, Equal Cost Multi-Pathing (ECMP), *MSTL* based routing, and *MSTL*+Hops based routing. In the next step, we computed the transmission schedules for the flows using NW-PSP and the routes computed in the previous step with the different algorithms. The results of this evaluation are summarized in Figure 5.2a.

The results show that with an increase in the number of flows the schedule flowspan increases rapidly in the case of shortest path routing. This is because the shortest path routing is agnostic to the load of scheduled traffic while computing routes. ECMP fares much better with the increase in flowspan being gradual, as it tries to randomly distribute the load of scheduled traffic throughout the network. Further, *MSTL*-based routing typically yields better schedules compared to routing using ECMP, but as it may route flows over longer paths, occasionally the flowspan may be higher than that with ECMP. In all the cases, *MSTL*+Hops based routing outperforms all the other routing schemes, and yield schedules that have on an average 38 % and 20 % lower flowspan compared to the shortest path routing and ECMP, respectively.

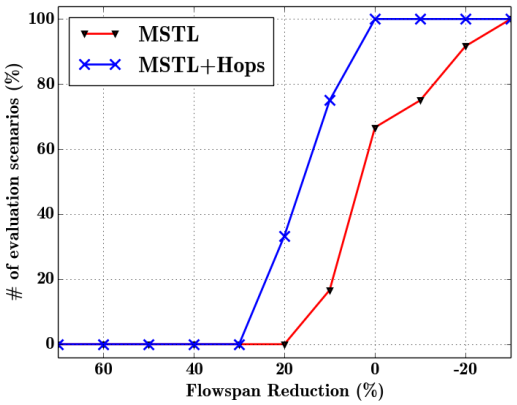
We also executed the presented ILP-based routing algorithms for computing routes for time-triggered flows in 24 different scenarios (varying number of flows on three topologies of differing sizes). The subsequently computed schedules were then compared with the ones resulting from the shortest path routing and ECMP. Figure 5.2b and Figure 5.2c show the cumulative distributions of flowspan reduction compared to the shortest path routing and ECMP, respectively. The results show that while the *MSTL*-based approach, in general, improves quality of schedules, in a few cases, it ends up increasing the flowspan compared to the off-the-shelf algorithms. Overall, such cases were limited to less than 10 % and 30 % with shortest path routing and ECMP as references, respectively. However, the *MSTL*+Hops based approach always yields an



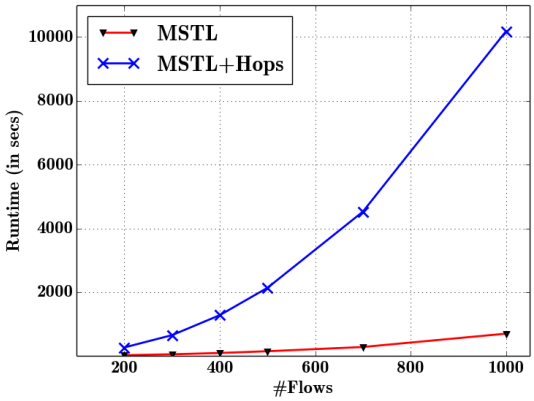
(a) Routing algorithms vs flowspan



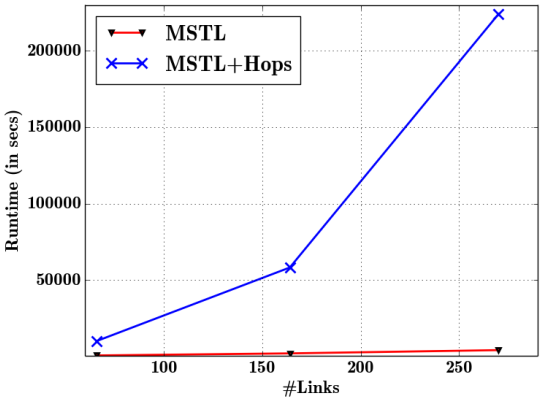
(b) Flowspan reduction compared to SP



(c) Flowspan reduction compared to ECMP



(d) Runtime vs No. of Flows



(e) Runtime vs No. of Links

Figure 5.2: Evaluations Results for the ILP formulations presented in Section 5.3

improvement in flowspans, with schedule flowspans reduced by up to 60 % and 30 % compared to shortest path routing and ECMP, respectively.

Scheduling using SMT-Based Approach

As already mentioned, the SMT-based approaches aim to find a feasible schedule for a set of time-triggered flows given their routes. As they do not strive to bunch the traffic towards the start of the scheduling cycle, evaluating the quality of schedules that these approaches compute using flowspan as a metric is meaningless. Hence, we improvised an evaluation strategy to determine the quantum of time-triggered traffic that can be scheduled into the network [38]. For this, we executed the scheduling algorithm repeatedly increasing the number of time-triggered flows given as input with each iteration, till the scheduling algorithm could no longer compute a feasible schedule, i.e., till the network was saturated with time-triggered flows. However, the problem with this strategy was that as the number of flows increased, the runtime of the scheduling algorithm increased exponentially. To avoid this problem, we modified our approach to determine the saturation point of the network with respect to time-triggered flows by means of incremental scheduling, i.e., we schedule new batches of flows into the network without modifying the schedules of the old flows till we are no more able to add further flows. While this evaluation method does not accurately determine the so-called saturation point of the network with respect to the scheduled traffic, we can still compare the performance of the different routing algorithms with respect to the yielded schedules.

We evaluated the saturation point for an Erdős-Rényi network with 10 switches and 30 hosts by incrementally adding 5 flows in each iteration. Each flow transmits packets of sizes varying between 300–1500 bytes per scheduling cycle. For the sake of simplification, we used the same time-period (100 time units) for all flows. With shortest path routing the scheduling algorithm could not schedule more than 20 flows. This is understandably because the bottleneck link is highly utilized to be able to accommodate further bandwidth for new time-triggered flows without affecting the already scheduled flows. ECMP, on the other hand, could accommodate 70 flows, while the *MSTL*-based approach could manage 100 flows. However, like in the NW-PSP scheduling, *MSTL*+Hops based approach managed to have the highest number of time-triggered flows scheduled into the network with a total of 110 flows.

From these evaluations, we conclude that using our routing approaches will improve the schedulability for SMT-based approaches as well.

5.4.2 Scalability Evaluations

The runtimes for the ILP-based routing algorithms that we presented in this chapter depend on the number of flows which are to be routed and the size of the topology

(in terms of the number of links). To determine the scalability of the algorithms with respect to the number of flows, we executed the algorithms for computing routes for varying number of flows (200–1000) in an Erdős-Rényi network with 50 hosts and 10 switches (66 links). As shown in Figure 5.2d, the runtimes for the *MSTL*+Hops based algorithm increases steeply with the number of flows being scheduled, with approximately 3 hours of runtime for routing 1000 flows. In contrast, the runtimes for the MSTL-based approach increase gradually with the number of flows. The runtime for routing 1000 time-triggered flows using this approach is approximately 11 minutes.

We also evaluated the scalability of the algorithms with respect to the size of the topology. We routed 1000 flows on networks of different sizes. Figure 5.2e summarizes the results of this evaluation. Similar to the earlier evaluations, the runtimes for *MSTL*+Hops based routing algorithm increase rapidly with the number of links. The runtimes increase from about 3 hours to 63 hours when the size of topology is increased from 66 links to 270 links. The *MSTL*-based routing can, however, route 1000 flows in a network topology with 270 links in approximately 1 hour.

5.5 Related Work

Routing in complex networks is a very old and well-researched problem. The initial solutions to the routing problem in computer networks, minimizing the number of hops, were already published in the 1960s [79] [80]. In modern Ethernet networks, routing is achieved by means of a spanning tree of the topology created using the different variants of the spanning tree protocols specified in the IEEE 802.1Q [81] and IEEE 802.1D standards [82]. The fact that routing over shortest path between the source and the destination does not always provide quality-of-service (QoS) guarantees to the flows led to the development of several other approaches to integrate QoS metrics into routing. The emergence of SDN paradigm has further facilitated the implementation of centralized QoS aware routing algorithms [15]. Guck et al. present a detailed survey on the various approaches to integrate QoS metrics into routing algorithms [83].

The Time-sensitive Networking (TSN) Task Group (TG) has explicitly considered routing problem in the extensions published for time-sensitive networks. The IEEE 802.1Qca [27] is a TSN extension which provides explicit control of the routes for time-sensitive data streams, while the IEEE 802.1CB [47] specifies mechanisms for routing the time-sensitive data streams over redundant paths and elimination of duplicate frames which result from such redundant forwarding. However, these standards do not specify any algorithms for computing routes for time-sensitive data streams.

The TSN also proposes extensions for accommodating shaped traffic, e.g., the Audio Video Bridging (AVB) data streams, in addition to mechanisms for scheduled traffic in Ethernet networks. The latency guarantees demanded by these data streams are usually slightly relaxed in comparison to the end-to-end latency and jitter bounds

required for the scheduled traffic. In TSN, shaped traffic can be handled using the credit based shaper specified in the IEEE 802.1Qav [44]. The routing of these data streams must, however, consider the routes and schedules of the time-triggered data streams. On links with high utilization for scheduled traffic, these streams might not get enough bandwidth despite having sufficient credit. Laursen et al. present a greedy approach for routing these data streams such that the latency these streams experience is minimized considering the intervening scheduled traffic [84].

Most approaches for handling scheduled traffic in time-sensitive networks consider only the scheduling aspect, while a few recent approaches have integrated routing along with scheduling (cf. Section 4.8). A noteworthy contribution in this direction is from Gavrilut et al. who present algorithms for computing fault tolerant routes for an asynchronous traffic class scheduled using an urgency based scheduler (cf. [77]) in TSN using redundant links and bridges [85].

For approaches separating the scheduling of time-triggered flows from its routing, the impact the routes of the flows have on the computed schedules is an important aspect. Kentis et al. hold the port congestion measured in terms of number of flows as the single most important factor influencing the schedules [86]. Along with our work, this contribution is among the first ones to discuss the impact of routing on the schedulability in time-sensitive networks. We, however, took this a step ahead and developed routing algorithms based on this effect.

5.6 Summary

Computing appropriate transmission schedules is an important aspect for providing real-time guarantees for the scheduled traffic in the IEEE 802.1Qbv networks. As a consequence of the high time-complexity of the scheduling problem, we separate it from the corresponding routing problem and solve both of them disjointly. However, the routing of time-triggered flows directly affects the computation of their transmission schedules. In this chapter, we discussed the impact that routing of time-triggered traffic has on the quality of the schedules computed and the need for specialized algorithms for routing such traffic. We identified parameters, in addition to the number of hops, which must be considered by routing algorithms while computing routes for time-triggered flows. We also proposed two ILP-based routing algorithms based on our findings for this purpose. Our evaluations show that specialized routing algorithms can improve the quality of schedules by up to 30 % and 60 % compared to the shortest path routing and equal cost multipathing, respectively.

CHAPTER 6

JOINT SCHEDULING AND ROUTING USING SDN

6.1 Introduction

So far we have addressed the scheduling and routing problems for time-triggered flows in networks equipped with the programmable gating mechanisms specified in the IEEE 802.1Qbv standard. While these extensions provide primitives for enforcing transmission schedules in the network, they also increase the complexity, and thus, the costs of the switching hardware. An interesting question to pose here is to what kind of guarantees can be provided to the scheduled traffic in absence of these enhancements. After all, not all applications relying on time-triggered traffic demand zero jitter for communication. Furthermore, it would suffice in some networks, e.g., smaller shop-floors with fewer real-time applications using the time-triggered communication paradigm, if only a relatively smaller quantum of scheduled traffic can be handled in comparison to what is theoretically achievable using additional hardware enhancements like the programmable gating mechanisms.

In this chapter, we look at networks which are not equipped with these specialized extensions, i.e., per-hop transmission schedules cannot be enforced for the time-triggered traffic as the switches merely act as delay elements. It is possible in such networks to schedule the transmissions of time-triggered traffic at their respective source hosts provided their clocks are synchronized. The IEEE 1588 Precision Time Protocol or the IEEE 802.1AS, the standard dealing with clock synchronization within the umbrella of Time-sensitive Networking, which provide precision in the range of nanoseconds could be used for this purpose. However, the algorithms for computation of schedules and routes for time-triggered flows in such networks would vary substantially from the algorithms presented in Chapter 4 and 5. The challenge here is to model the effects of the best-effort traffic in transit on the traversal of the time-triggered flows through

the network. Given the lack of specialized hardware, the temporal isolation between scheduled traffic and other classes of traffic, achieved by means of guard bands (cf. Section 2.2.1), is not feasible.

In this chapter, we present Time-sensitive Software-defined Network (TSSDN), an SDN-based architecture, for handling scheduled traffic in Ethernet networks using network elements which are not equipped with any specialized enhancements for handling scheduled traffic. By means of transmission scheduling at the source hosts and prioritizing the time-triggered traffic within the network while appropriately routing it, TSSDN looks to bound the non-deterministic queuing delays that the packets belonging to time-triggered flows incur. It must be noted that the lack of mechanisms to precisely schedule the transmissions at the switches means that TSSDN cannot completely eliminate the effects of other classes of traffic on the scheduled traffic, nonetheless, it manages to keep the jitter reasonably low. We also introduce the scheduling problem in TSSDN and propose various Integer Linear Programming (ILP) formulations for computing routes and transmission schedules for time-triggered flows. In particular, we present formulations which compute optimal solution along with faster heuristics. In contrast to our solutions for the scheduling and routing of time-triggered flows in networks with IEEE 802.1Qbv extensions, we present integrated solutions which jointly compute the schedules and routes for such flows in TSSDN.

Our evaluations show that the presented ILP formulations can generate transmission schedules for networks of realistic sizes within seconds. Through a proof-of-concept implementation, we show that adherence to the computed schedules using user-space packet processing frameworks does indeed result in bounded network delay and jitter. We observed end-to-end latencies of $\leq 14\,\mu s$ with ultra low jitter ($\leq 7\,\mu s$) on our benchmark topology created using commodity SDN switches.

This chapter is structured as follows. We present the system model of TSSDN and the concrete problem statement with respect to the transmission scheduling and routing of time-triggered flows in TSSDN in Section 6.2. The ILP based solutions to computing transmission schedules and routes for time-triggered flows are presented in Section 6.3. We discuss issues like the network utilization with respect to scheduled traffic in Section 6.4. The evaluations and the related work are presented in Section 6.5 and Section 6.6, respectively.

6.2 System Model & Problem Statement

6.2.1 System Model

In this section, we describe the system model for TSSDN by specifying the changes and the additional assumptions to the unified system model presented in Chapter 3.

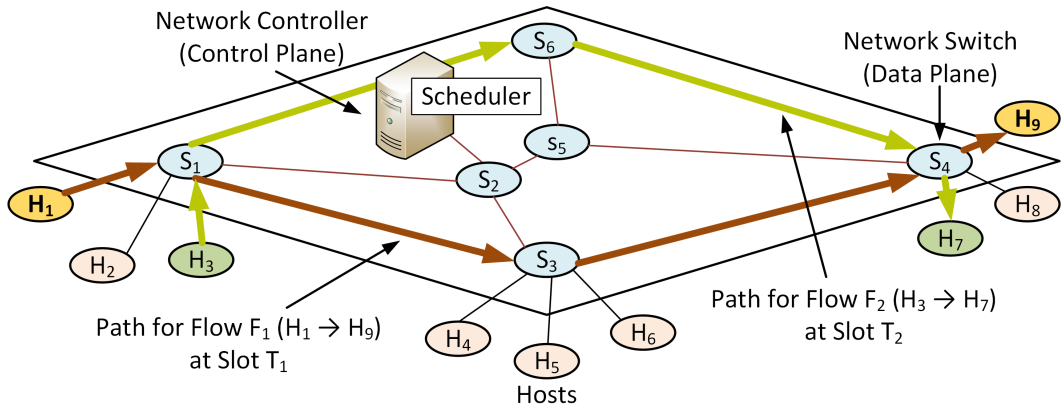


Figure 6.1: Architecture of Time-sensitive Software-defined Network (TSSDN). Network controller routing flows F_1 & F_2 and allocating them slots T_1 & T_2, respectively.

The architecture of TSSDN is shown in Figure 6.1. The data plane in TSSDN consists of commodity SDN switches, cut-through or store-and-forward, and end systems. TSSDN relies on in-network prioritization to separate scheduled traffic from the best-effort traffic. To this end, the packets belonging to the time-triggered flows are tagged by the source host as high priority traffic using for instance the Priority Code Point (PCP) in VLAN tags or the Differentiated Services Code Point (DSCP) in the IP header. Thus, the scheduled traffic is prioritized over the low priority best-effort traffic with respect to forwarding. It must be noted that the tagging of scheduled traffic in TSSDN is to explicitly prioritize it in the network, in contrast to simply directing the concerned packets to the queues specifically reserved for scheduled traffic (which may be different from the queues with highest priority) in networks with the IEEE 802.1Qbv extensions. Another peculiarity of TSSDN is that only the clocks of the end systems need to be synchronized. The switches themselves may be left unsynchronized as they are not equipped with the gating mechanism which can benefit from synchronized clocks.

The control plane of TSSDN is similar to the one mentioned in the unified system model. The control plane is centralized and is responsible for computing schedules and routes for the time-triggered flows as shown in Figure 6.1. The routes are programmed in the switches using OpenFlow, the SDN south-bound protocol. For communicating the schedule information to the end systems, the same protocol can be extended, or alternatively, a dedicated protocol can be developed.

We add additional restrictions to the time-triggered flows described in the unified system model. We require that the source hosts of time-triggered flows transmit application layer data units encapsulated in a single UDP packet (size limited by the network MTU) based on the computed transmission schedule. Like in Chapter 4, the use of cyclic scheduling approach in TSSDN requires all flows to have time-periods which are integral multiples of the base-period, t_{bp}. The base-period indicates the length of the

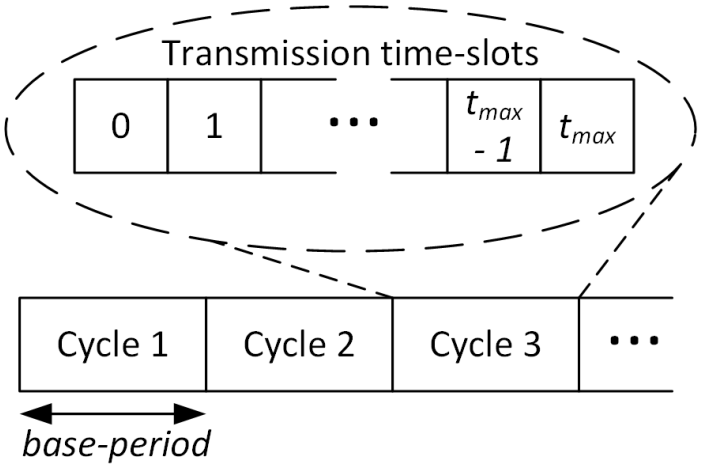


Figure 6.2: Transmission schedules in TSSDN

transmission schedule and the minimum time-period that time-triggered flows in the network may have. For schedule adherence by the source hosts, we again fall back to our solution using Intel's DPDK, presented in Section 4.5.

6.2.2 Problem Statement

The goal of TSSDN is to achieve deterministic network behavior with bounds on network delay and jitter for time-triggered traffic to support real-time communication. Therefore, it is essential for TSSDN to bound the non-deterministic queuing delay for time-triggered traffic. A radical approach is to altogether eliminate the non-deterministic queuing delays in the switches on the path from the source host to the destination host. Queuing occurs in switches when packets from multiple input ports attempt to transmit over the same output port simultaneously. For instance, in the topology shown in Figure 4.2, simultaneous transmissions of packets belonging to flows $F_i : A_i \rightarrow B_i; i \in [1 \ldots 5]$ will result in queuing at the output port of switch S_1. In such cases, the network delay for these packets would depend on the length of queues they encounter, i.e., the flows are affected by jitter. Queuing can be eliminated if no two inputs ports contend for transmitting over the same output port, i.e., the source host initiates transmission only when the entire network path over which the flow traverses is exclusively reserved for it. This implies that whenever packets belonging to time-triggered flows arrive at a switch, the ports on which they are to be forwarded are free and the packets are forwarded with *zero queuing delay*. For this, the scheduling algorithms in TSSDN are formulated to guarantee that no high-priority packets interfere and delay each other.

TSSDN implements a time division multiple access (TDMA) scheme, where every time-triggered flow has well-defined time-slots allocated by the network controller during which its source can transmit. Thus, the transmission schedule is modelled as a cyclic

schedule of duration equalling the *base-period*, t_{bp}, as shown in Figure 6.2. It is divided into smaller time-slots, numbered from 0 to t_{max}, each wide enough for an MTU-sized packet to travel across the longest network path. Note that the slot length is bounded as the longest path in TSSDN is restricted to 7 hops. The network controller can determine t_{max} based on the base-period and slot length, both being system parameters. The scheduler disburses time-slots, $T \equiv \{0, 1 \ldots, t_{max}\}$, to the sources of time-triggered flows while also routing them. To avoid queuing, the scheduler is restrained from allocating the same time-slot to multiple flows that have overlapping paths. For instance, in the topology shown in Figure 4.2, each of the flows, F_i, is allocated a different time-slot to sufficiently skew their transmissions and avoid queuing on the bottleneck link (from switch S_1 to S_2). The sources then compute the exact transmission instants using the base-period, the slot length, its transmission period, and the allocated time-slot. Without a suitable time-slot for a flow, the source cannot send packets as high priority traffic. Overall, by using coarse-grained schedules which determine only the transmission time at the source host (network edge) instead of computing fine-grained link schedules (through the network core) we avoid having to rely on underlying hardware to enforce such schedules.

TSSDN being a converged network supports scheduled traffic along with best-effort traffic. The links carry best-effort traffic while waiting for high priority time-triggered packets to arrive during a time-slot. This implies that the best-effort frames in transit can delay the high priority scheduled traffic. If best-effort frame is being transmitted while a high-priority frame belonging to a time-triggered flow arrives at a switch, either the high-priority frame is delayed till the transmission of the best-effort frame is finished or the best-effort frame is preempted to make way for scheduled traffic. In the former case, the frame may be delayed per hop by the time required to transmit an MTU sized packet in the worst case, while in the latter case the per hop delay ranges between 64–128 byte transmission times only. Given that the network diameter is restricted to 7 hops, the total amount of jitter introduced in TSSDN is also bounded. Zero queuing delay is, thus, a slightly idealized formulation of the goal. However, by almost eliminating the queuing delay for time-triggered flows (other than the one packet under transmission per hop) in the network, TSSDN delivers their corresponding packets to their destinations as early as possible. Furthermore, the slight jitter in TSSDN can be minimized by adding buffers to the destination host. Overall, TSSDN requires switches that have mechanisms to separate time-triggered traffic from best-effort traffic (e.g. using priority queues) and optionally frame preemption mechanisms like in IEEE 8021.Qbu [45] to minimize the jitter resulting from other traffic classes.

This coarse grained scheduling in TSSDN results in fewer time-slots than what is possible with fine-grained link schedules. Hence, the problem we deal with here is *how to compute a transmission schedule that maximizes the number of scheduled time-triggered flows?* By maximizing the time-triggered flows that can be carried over the network, a larger number of real-time applications can be supported. This maximization problem

is NP-complete, reducible from the static light path establishment problem [87] encountered during routing and wavelength assignment in optical networks which deals with allocation of wavelengths to optical data flows and routing them such that no two flows with the same wavelength traverse over the same optical link.

In this chapter, we present scheduling algorithms in the form of ILP formulations that compute transmission schedules for a static set of time-triggered flows known *a priori*. ILPs are a broadly accepted standard method for such optimization problems with readily available, highly optimized solvers. Alternatively, as mentioned in the related work, Satisfiability Modulo Theories (SMT) based approaches could also be used. However, since ILP and SMT showed similar performance [88], the formulations presented in this work are limited to ILPs.

6.3 Scheduling & Routing in TSSDN

The goal of the static scheduling problem is to maximize the number of time-triggered flows which are accommodated in the network, i.e., the flows which are allocated time-slots and routes. The general approach to this problem is to define for each time-triggered flow a set of candidate paths between its source and destinations. The solution to the scheduling problem requires routing flows over one of its candidate paths and allocating them a time-slot, while maximizing the number of flows that can be allocated with a time-slot and a route. This approach is referred to as Scheduling with Pathsets Routing (S/PR).

The choice of candidate paths for the flows influence the number of flows successfully scheduled. In general, the set of candidate paths for each flow can be made from all the shortest paths between its source and its destinations. Selecting all paths (including the non-shortest paths) that exist between the source and the destinations of the flow in the set of candidate paths yields the best solution for the static scheduling problem. Such an approach is named as Scheduling with Unconstrained Routing (S/UR). The set of candidate paths may also be restricted to only one of the randomly chosen shortest paths for computing solutions faster. Such an approach with *a priori* routing is named as Scheduling with Fixed Routing (S/FR). It must be noted that the S/UR and S/FR are specific cases of S/PR and are at the two extremes of the solution space.

In the following, we present the detailed ILP formulations for the three approaches. In interest of comprehension, we restrict these ILP formulations to schedule and route unicast time-triggered flows. Along with each of these formulations, we also mention the rather trivial extensions with which the approaches can be extended to handle multicast flows

6.3.1 Terminologies

For the ILP formulations, we denote the network topology as a directed graph $G \equiv (V, E)$, where V is the set of nodes and $E \equiv \{(i,j)|\ i,j \in V$ and i,j are connected by a network link$\}$ is a set of tuples representing the network links. Further, $V \equiv (S \cup H)$, where S and H are sets of switches and end systems, respectively. The time-triggered flows are denoted as a tuple, $ts_i \equiv (s_i, d_i, p_i)$, where $s_i, d_i \in H$. Here, s_i and d_i is the source and the destination of the flow respectively, while p_i is the period of the flow. The set of time-slots available for disbursement is denoted as $T \equiv \{0, 1 \ldots, t_{max}\}$. Additional functions needed to model the topology and time-triggered flows are listed in Table 6.1.

6.3.2 Determining the Base-period in TSSDN

In TSSDN, the base-period determines the number of time-slots available in the network for accommodating time-triggered flows. While higher values for the base-period result in a larger number of time-slots, the base-period must also be restricted to values lower than or equal to the lowest transmission period that a flow may have in the network. Moreover, the choice of the base-period is important as the periods of time-triggered flows are constrained to be an integral multiple of the base-period. For flows violating this condition, the period can be *reduced to the nearest multiple of base-period.* However, this results in an equivalent increase in the network load (packets belonging to time-triggered flows per unit time) on account of scheduled traffic. For a given set of flows, TS, and a base-period, t_{bp}, the network load due to scheduled flows is:

$$\text{Scheduled Traffic Load} = \sum_{\forall\ ts\ \in\ TS} \frac{1}{t_{bp} \cdot \lfloor \frac{\text{period}(ts)}{t_{bp}} \rfloor}$$

The network load on account of the time-triggered flows in TSSDN varies based on the selected value of the base-period. To minimize load due to scheduled traffic, the base-period is determined by evaluating the load for scheduled traffic across different values of possible base-periods. A reasonable value is chosen such that enough time-slots are available for scheduling while the load on account of scheduled traffic is also

Helper Function	Parameters	Output
in(n)	$n \in V$	$\{(u,v) \in E \| v = n\}$
out(n)		$\{(u,v) \in E \| u = n\}$
src(ts)	Flow ts, $ts \equiv (s,d,p)$	s
dst(ts)		d
period(ts)		p

Table 6.1: Helper functions for modeling network topology and time-triggered flows

minimal. Once chosen the base-period remains fixed unless new flows need to be accommodated in the network, e.g., in dynamic scenarios. It must be noted that modifying the base-period at runtime results in high coordination overhead and is, hence, avoided in TSSDN.

6.3.3 Scheduling with Pathsets Routing (S/PR)

For *Scheduling with Pathsets Routing*, we extend the model of time-triggered flows to additionally include a set of "candidate" paths that it may use. The ILP formulation is restricted to route the flow through one of the paths in this set instead of searching the complete solution space for arbitrary paths. We use the set of all shortest paths between the source and destination of a given flow as its candidate paths.

ILP Inputs

The inputs for the ILP formulation are the set of flows to be scheduled and the paths through which each of the flows may be routed.

- Set of flows to be scheduled, TS.
  $TS \equiv \{ts_i\}$; $i \in [1 \dots NumFlows]$
  Here, $NumFlows$ represent the number of flows to be scheduled.

- Set of possible paths through which the flows may be routed, P.
  $P \equiv \{p_l\}$; $l \in [1 \dots NumPaths]$
  This set contains all the shortest paths from the source to the destination for all flows $ts \in TS$.

- Mapping between flows and paths, SP.
  $SP \equiv \{sp_{i,l}\}$; $\forall i \in TS$, $\forall l \in P$
  $sp_{i,l} = 1$, if flow i can traverse over path l, else 0.
  It must be noted that while a flow has multiple candidate paths over which it may be routed, a given path can be used by only one flow, i.e., the path uniquely identifies the flow.

- Mapping between paths and links, PL.
  $PL \equiv \{pl_{l,j}\}$; $\forall l \in P$, $\forall j \in E$
  $pl_{l,j} = 1$, if path l includes link j, else 0.

ILP Variables

In this formulation, we allocate time-slots to paths instead of flows.
$PT \equiv \{pt_{l,k}\}$; $\forall l \in P$, $\forall k \in T$
$pt_{l,k} = 1$, if path l is allocated time-slot k, else 0.

Objective Function

The objective for this ILP formulation is to maximize the number of paths with assigned time-slots.

$$\text{Maximize} \sum_{\forall k \in T} \sum_{\forall l \in P} pt_{l,k} \tag{6.1}$$

Constraints

The constraints for this ILP formulation are enumerated as below:

- Each path may be allocated at-most one time-slot.

$$\sum_{\forall k \in T} pt_{l,k} \leq 1 \qquad \forall l \in P \tag{6.2}$$

- Each flow can be allocated at-most one time-slot, i.e., for a given flow, only one of its candidate paths can be allocated a time-slot.

$$\sum_{\forall k \in T} \sum_{\forall l \in P} pt_{l,k} \times sp_{i,l} \leq 1 \qquad \forall i \in TS \tag{6.3}$$

- To avoid collisions, no two paths with overlapping links will be assigned the same time-slot.

$$\sum_{\forall l \in P} pt_{l,k} \times pl_{l,j} \leq 1 \qquad \forall k \in T, \forall j \in E \tag{6.4}$$

The ILP solver sets values for PT based on which the network controller can disburse the time-slots for the flows and accordingly route them as well.

This ILP formulation can be easily extended for handling multicast flows by using candidate spanning trees for the corresponding flows. Spanning tree for a multicast flow connects the source host of the flow with all the destinations. Computing multiple spanning trees for a set of nodes is a well researched problem and is now standardized into the Multiple Spanning Tree Protocol specified in the IEEE 802.1Q standard [81].

6.3.4 Scheduling with Unconstrained Routing (S/UR)

In this approach, the scheduler is free to route the time-triggered flows over any available path. Ideally, we could simply compute all paths between the source and destination pairs and reuse the ILP formulation of S/PR. However, computing all paths between a pair of nodes in a network has a time complexity of $\mathcal{O}(n!)$. Hence, we integrate the selection of path for the flows into the ILP formulation.

The network topology and the set of desired time-triggered flows are the inputs. Variables are the time-slots and paths for the flows. The optimization objective is to maximize the number of flows that are allocated a time-slot.

ILP Inputs

The inputs required for the ILP formulation are as follows:

- Network Topology, $G \equiv (V, E)$.
- Set of time-triggered flows to be scheduled, TS.
  $TS \equiv \{ts_i\}$; $i \in [1 \dots NumFlows]$
  Here $NumFlows$ represent the number of flows to be scheduled.

ILP Variables

The decision variables used for formulating the ILP are as follows:

- Mapping of flows to network links, SL.
  $SL \equiv \{f_{i,j}\}\ \forall i \in TS, \forall j \in E$
  $f_{i,j} = 1$, if the flow i traverses over link j, else 0.
- Mapping of flows to time-slots, ST.
  $ST \equiv \{t_{i,k}\}\ \forall i \in TS, \forall k \in T$
  $t_{i,k} = 1$, if the flow i is allocated time-slot k, else 0.
- Helper variables, SLT. These enable the formulation of the scheduling problem as an ILP.
  $SLT \equiv \{m_{i,j,k}\}\ \forall i \in TS, \forall j \in E, \forall k \in T$
  $m_{i,j,k} = 1$, if the flow i traverses over link j and is allocated a time-slot k, else 0.

Objective Function

The objective function is formulated so as to primarily maximize the number of flows that are allocated time-slots. In some situations different solutions might exist with the same number of scheduled flows but where some solutions contain loops in their paths. Obviously, in such cases we would prefer the solutions without loops. Therefore, we define a secondary objective for weeding out solutions that route flows over paths with loops. This term in the objective keeps the path length at a minimum, thus, eliminating paths with loops, and is factored such that its total contribution to the objective is less than one. This ensures that the ILP solver gives priority to maximizing the number of flows that can be scheduled rather than minimizing the length of the individual paths allocated to them. Alternatively, we can process the computed solutions to remove the loops in the paths in the absence of the secondary objective.

Maximize

$$\underbrace{\sum_{\forall i \in TS} \sum_{\forall k \in T} t_{i,k}}_{\text{Primary Objective}} - \underbrace{\frac{1}{(|TS| \times |E|) + 1} \times \sum_{\forall i \in TS} \sum_{\forall j \in E} f_{i,j}}_{\text{Secondary Objective}} \tag{6.5}$$

Constraints

The constraints for the ILP formulation are as follows:

- Every flow shall be allocated at most one time-slot as they carry only one MTU-sized packet during their corresponding period.

$$\sum_{\forall k \in T} t_{i,k} \leq 1 \qquad \forall i \in TS \tag{6.6}$$

- The path for a given flow, i, starts at its source host and ends at its destination host, i.e., the source host has only one outgoing link with no incoming links while the destination host has one incoming link with no outgoing links. For all the other network nodes, the number of incoming links is equal to the number of outgoing links.

$$\begin{aligned} \sum_{\forall j \in in(src(i))} f_{i,j} = 0 \qquad & \sum_{\forall j \in out(src(i))} f_{i,j} = 1 \\ \sum_{\forall j \in in(dst(i))} f_{i,j} = 1 \qquad & \sum_{\forall j \in out(dst(i))} f_{i,j} = 0 \end{aligned} \tag{6.7}$$

$$\sum_{\forall j \in in(n)} f_{i,j} = \sum_{\forall j \in out(n)} f_{i,j} \quad \forall n \in V \setminus \{src(i), dst(i)\} \tag{6.8}$$

  This constraint is valid for all flows, i.e., $\forall i \in TS$.

- Multiple flows cannot be routed over a given link during any of the time-slots. This constraint ensures that the entire path for each flow is reserved for the flow exclusively during its allocated time-slot.

$$\sum_{\forall i \in TS} m_{i,j,k} \leq 1 \qquad \forall j \in E, \forall k \in T \tag{6.9}$$

- Finally, we need additional constraints to ensure that the ILP solver provides consistent values for the variables, i.e., for a flow i, edge j and time-slot k, the variable $m_{i,j,k}$ can be 1, only if variables $f_{i,j}$ and $t_{i,k}$ are both 1. Hence, the following constraint is required:

$$m_{i,j,k} = f_{i,j} \times t_{i,k} \qquad \forall i \in TS, \forall j \in E, \forall k \in T \tag{6.10}$$

Although this constraint is non-linear, it can be transformed into purely linear constraints as follows:

$$\left.\begin{aligned} m_{i,j,k} &\leq f_{i,j} \\ m_{i,j,k} &\leq t_{i,k} \\ m_{i,j,k} &\geq f_{i,j} + t_{i,k} - 1 \end{aligned}\right\} \forall i \in TS, \forall j \in E, \forall k \in T \tag{6.11}$$

The ILP solver sets values for the variables SL and ST corresponding to the computed schedule. The network controller configures the flow-tables in the switches for routing flows as per SL and disburses the time-slots as per ST.

This ILP formulation results in an optimal schedule, i.e., maximum number of time-triggered flows are scheduled from a given set of flows, if the transmission period of all flows is equal to the base-period. Presence of flows with higher periods might result in sub-optimality, the extent of which depends on the number of such flows and the difference between its individual periods and the base-period.

The presented ILP formulation can be extended to handle multicast flows by use of additional auxiliary variables per flow to keep a count of number of destinations of a flow reachable over a link and reformulating the constraints 6.7 and 6.8 on the lines of the routing constraints 5.2, 5.3, and 5.5, presented in Chapter 5.

The runtime for computing transmission schedules with this ILP formulation is impractical (order of days), mainly because of the two degrees of freedom it has, viz., the routes for the flows and the corresponding time-slots. However, with respect to paths it seems reasonable to prefer short paths as it would result in fewer possibilities of slot collisions along paths sharing the same links. Our other approaches—*S/PR* and *S/FR*—restrict the search space to explore only the shortest paths, and thus, reduce the runtime. This may, however, result in a lower number of flows being scheduled in comparison to *S/UR*.

6.3.5 Scheduling with Fixed-path Routing (S/FR)

Another approach further reducing the execution time for computing transmission schedules is the *Scheduling with Fixed-path Routing Approach*. This approach extends the idea of *S/PR*. Here, we take a radical approach by choosing the path for a given flow randomly from the set of all shortest paths between its source and destination similar to Equal Cost Multi Path (ECMP) routing. Then, the ILP formulation only deals with the time-slot allocation. While this approach is faster than *S/PR*, the computed schedule might be of even lower quality relative to *S/UR*.

ILP Inputs

The inputs for the ILP formulation is the set of flows to be scheduled and the path through which each of the flow is routed (selected at random from the set of all shortest paths between the source and the destination of the flow).

- Set of flows to be scheduled, TS.
  $TS \equiv \{ts_i\}$; $i \in [1 \ldots NumFlows]$
- Mapping of flows to links, SL, indicating the links that belong to the path that a flow must traverse.
  $SL \equiv \{f_{i,j}\}$; $\forall i \in TS, \forall j \in E$:
  $f_{i,j} = 1$, if flow i traverses over link j, else 0.

ILP Variables

Decision variables are required only for mapping a flow to time-slots. ST indicates the time-slot that is allocated for a flow.
$ST \equiv \{t_{i,k}\}$; $\forall i \in ST, \forall k \in T$
$t_{i,k} = 1$, if flow i is allocated time-slot k, else 0.

Objective Function

The objective function is formulated so as to maximize the number of flows that are allocated time-slots.

$$\text{Maximize} \sum_{\forall i \in TS} \sum_{\forall k \in T} t_{i,k} \tag{6.12}$$

Constraints

The constraints for this ILP formulation are enumerated as below:

- Each flow may be allocated at most one time-slot.

$$\sum_{\forall k \in T} t_{i,k} \leq 1 \qquad \forall i \in TS \tag{6.13}$$

- To avoid collisions, no two flows can be allocated the same time-slot if they have overlapping paths.

$$\sum_{\forall i \in TS} t_{i,k} \times f_{i,j} \leq 1 \qquad \forall j \in E, \forall k \in T \tag{6.14}$$

The ILP solver allocates time-slots to the flows through T. Based on these values, the slots can be disbursed by the network controller.

6.4 Discussion

6.4.1 Network Utilization in TSSDN

The scheduling model in TSSDN allocates a time-slot to each flow such that the entire path over which the flow traverses is exclusively reserved for it. The time-slots are, hence, required to be long enough for guaranteeing this even for the worst case, i.e., traversal of an MTU-sized packet over the network diameter. Ideally it would be sufficient to reserve time-slots over network links, where the length of time-slots is limited to the time required for transmitting the actual size of the packet plus some guardbands to isolate the scheduled traffic from best-effort traffic, e.g. using the gating mechanisms in IEEE 802.1Qbv [30] [31].

The coarse grained scheduling model of TSSDN results in lower available bandwidth for scheduled traffic in comparison to the fine-grained link scheduling in the ideal case. However, TSSDN, being work-conserving in nature, makes available the remaining bandwidth for best effort traffic. Given that a sufficient amount of best-effort traffic is available in the network, TSSDN can achieve full network utilization. In contrast, with fine-grained link scheduling (non-work-conserving in nature), like in IEEE 802.1Qbv networks, bandwidth is lost due to the presence of guard-bands or when reserved time-slots are not used by time-sensitive traffic, e.g. in event-based communication. During the guard-bands, new packets are not taken up for transmission, even if the corresponding link is idle, to avoid interference with the next time-slot for the scheduled traffic. Usually, the guard bands are configured to be long enough for a MTU-sized packet to be transferred over the corresponding link, unless the switching hardware supports the frame pre-emption mechanisms from IEEE 802.1Qbu. The number of guard bands required and the resulting drop in network utilization is dependent on the topology of the network and the used scheduling approach.

In absence of best-effort traffic, the network utilization would drop significantly for TSSDN. The worst case for TSSDN would be a line topology with 6 switches handling only scheduled traffic. Remember that TSSDN restricts the network diameter to 7 hops. In this case, the network utilization may potentially drop to approximately one-seventh of what would be possible with fine-grained link scheduling. Furthermore, the reservation for time-triggered traffic in TSSDN does affect the bandwidth available for best-effort traffic. However, transport protocols like TCP will adapt to the available bandwidth automatically.

6.4.2 Accounting for Time-periods in TSSDN

One of the shortcomings of the presented ILP formulations for computing schedules and routes in TSSDN is that it does not consider the period of the flows for the computation. Thus, the already scarce time-slots for the time-triggered flows is distributed assuming that all flows have periods equalling the base-period. Accounting for the individual transmission periods of the flows increases the complexity of the scheduling problem manifold along with the execution times of the ILP solver. Most of the related work [68] and state-of-the art field-bus architectures schedule time-triggered flows in industrial automation systems assuming that they transmit either with base-period or periods very close to it. Accordingly, TSSDN also ignores the actual transmission periods for the flows for scheduling.

In Chapter 7, we present additional extensions to TSSDN to share the time-slots across multiple flows with periods higher than the base-period in dynamic scenarios.

6.5 Evaluations

We evaluated TSSDN on two fronts. Firstly, we measured the end-to-end latency for time-triggered traffic on the data plane of TSSDN under various scenarios to determine if it provides the promised real-time guarantees. Secondly, we evaluated the ILP formulations, executing on the control plane, to compute transmission schedules for randomized graphs created using various models to exhibit its correctness and scalability.

6.5.1 Data Plane Evaluations for TSSDN

To evaluate the real-time properties provided by TSSDN on the data plane, we implemented the benchmark topology, shown in Figure 4.2, using five commodity machines (Intel Xeon E5-1650) each equipped with an Intel XL710 quad 10 GbE network interface and an Edge-Core cut-through bare-metal switch (AS5712-54X) running PicOS (ver 2.6.1). The switch was partitioned into virtual switches to create the topology, while each machine hosted two end systems, for instance, Host A_1 and B_1 were placed on the same machine but used different network interfaces. We used the Precision Time Protocol (PTP) for synchronizing clocks on all machines. To this end, we used a separate network infrastructure using a third network interface on each machine (two interfaces are used by the end systems hosted on the machine) dedicated to PTP synchronization. This was basically necessary because of two reasons: First, our switch did not support PTP. Thus, high priority time-triggered packets could potentially impact the accuracy of PTP latency measurements. With a switch which can measure the port-to-port residence time of PTP packets, the precision of clock synchronization would not be affected. Secondly, DPDK exclusively allocates a network interface to a

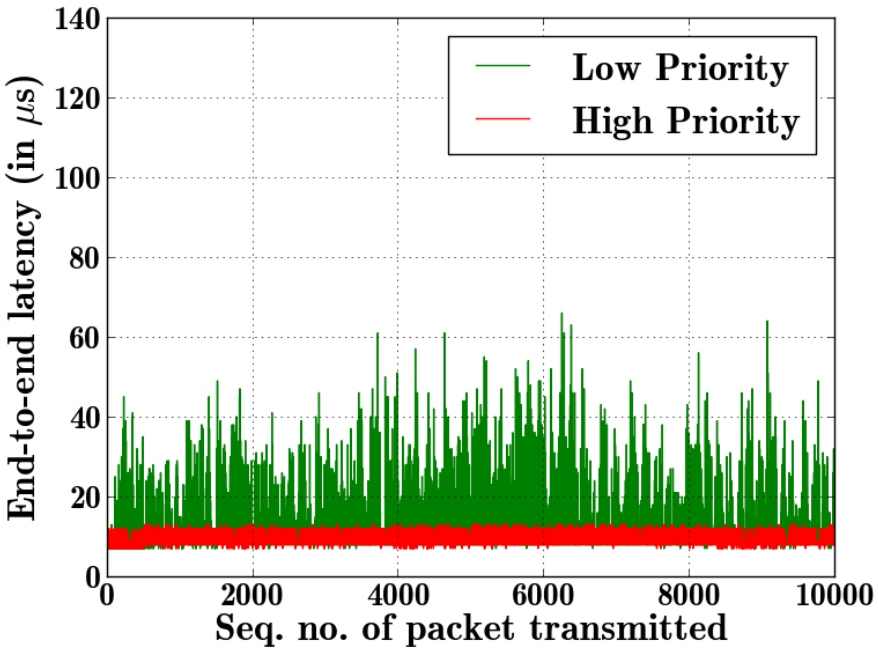


Figure 6.3: Impact of packet prioritization

process, so we cannot easily run a PTP daemon over the same port. Sharing a port between different processes is a common problem of current userspace packet processing frameworks and an orthogonal research problem.

In-network Prioritization

The end systems in TSSDN use userspace packet processing frameworks to adhere with the computed schedules. However, this alone is insufficient as TSSDN is also meant to additionally transport best-effort traffic. In this section, we experimentally show the importance of tagging time-triggered packets as priority traffic, while also motivating the need for transmission scheduling in TSSDN.

To determine the impact of best-effort traffic, we loaded the bottleneck link (link from switch S_1 to S_2) of our benchmark topology with random traffic (random packet sizes and variable bitrate) initiated by end systems A_2–A_5. The link was loaded to around 80 % of its total capacity. With this cross traffic, we measured the end-to-end latencies between the source and destination DPDK applications for 10,000 packets sent from $A_1 \rightarrow B_1$ with a period of 10 ms. As shown in Figure 6.3, the end-to-end latency fluctuates drastically between 7–66 μs, if the packets are not marked as priority packets by the source despite the spare capacity in the bottleneck link. End systems may tag time-triggered packets as high priority packets so that its delivery would be expedited by the data plane. We used the Priority Code Point (PCP) field of the IEEE 802.1Q VLAN tag and marked time-triggered packets with highest possible

# Flows	Avg.	Std. Dev.	Min	Max
1 Flow	7.99	0.62	7	13
2 Flows	8.09	0.57	7	14
3 Flows	8.04	0.49	7	14
4 Flows	8.07	0.48	7	14
5 Flows	8.06	0.54	7	14

Table 6.2: Latencies (in μs) for time-triggered flows when scheduled in adjacent time-slots

priority class (priority 7). With prioritization of time-triggered traffic, the end-to-end latency with cross traffic varies in a narrower band of 7–13 μs. However, the standard deviation of end-to-end latency has increased from sub-microsecond range (in absence of any interference from best-effort traffic) to 1.68 μs. This is because our switch does not support frame preemption (IEEE 802.1Qbu [45]), and hence time-triggered packets, though higher in priority, must queue till the current packet is transmitted. With support for frame preemption, higher priority time-triggered packets will not be affected by the lower priority cross-traffic.

The impact of prioritizing time-triggered packets is, however, nullified, if time-triggered flows are not temporally or spatially isolated. In absence of scheduling, no guarantees can be provided with respect to the bounds on end-to-end delays and jitter, even if the time-triggered packets are tagged as high priority packets.

Impact of Scheduling

To show the impact of scheduling, we deployed a varying number of time-triggered flows on our benchmark topology. We used a slot-length of 15 μs, considering the end-to-end delay in traversing the network diameter of our benchmark topology. We assume a base-period of 1 ms and that all flows use their slots completely, i.e., transmit one packet every 1 ms. The flows are allocated adjacent slots to demonstrate that schedules can be adhered precisely by the end systems. It may be noted that we evaluate our system in the toughest scenario with adjacent slots occupied on a 10 Gbps link as this would amplify the consequence of any non-adherence to schedules. We measured end-to-end latencies for 10^5 packets per flow and summarized the results in Table 6.2. The end-to-end delays for the time-triggered flows vary in a narrow band of $\leq 7\,\mu$s, irrespective of the number of flows in the network. Further, the standard deviation for the latencies experienced by the time-triggered flows is also in sub-microsecond range indicating minimal communication jitter. In networks with lower bandwidth links, the performance would be equally good or even better. Thus, we show that suitable transmission schedules impart real-time communication properties over the data plane of TSSDN.

# Flows	Avg.	Std. Dev.	Min	Max
2 Flows	8.63	0.86	7	14
3 Flows	9.19	1.14	7	14
4 Flows	9.75	1.42	7	15
5 Flows	10.2	1.71	7	17

Table 6.3: Latencies (in μs) for time-triggered flows when scheduled in the same time-slot

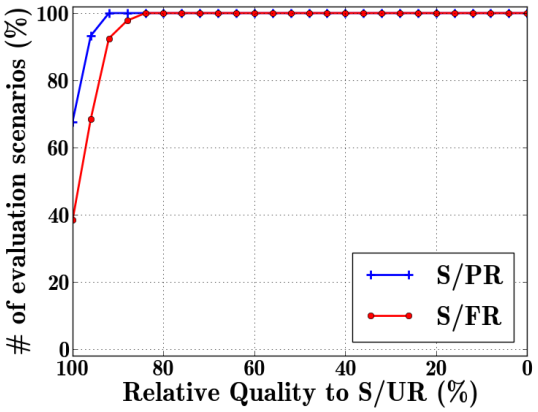
Further, to emphasize the importance of transmission scheduling, we measure the end-to-end latencies for a varying number of time-triggered flows when they are assigned the same transmission slot. Our ILP formulations would never allow these flows to interfere, however, in absence of scheduling such a scenario cannot be ruled out. Hence, we repeated the above experiment but allotted the same slot to the flows instead of adjacent ones. The results summarized in Table 6.3 show that end-to-end latency of time-triggered flows are affected if more than one flow is assigned the same slot. The average end-to-end delay and the standard deviation steadily increases with the number of time-triggered flows sharing the time-slot. Moreover, the jitter increases beyond 7 μs when more than 3 time-triggered flows contend for traversing a network link. This scenario also shows that in absence of scheduling, the time-triggered traffic could end up impeding each other in the network.

We observed that the jitter depends on the transmission frequency of the DPDK application and size of the packets being transmitted. For instance, jitter of $\leq 3\,\mu$s was observed at a frequency of 100 Hz for 64-byte sized packets, while it increased to $\leq 7\,\mu$s at a frequency of 10 kHz for 1500-byte sized packet. We infer that a part of this jitter (1–2 μs) originates from the interval timers in Linux, while the rest is a result of process preemptions or delayed availability of computing slice for the userspace applications (despite executing them with highest priority, i.e., nice level -20 in Linux) at source and destination hosts. One idea to reduce this residual jitter further is to use real time kernel patches on the source hosts.

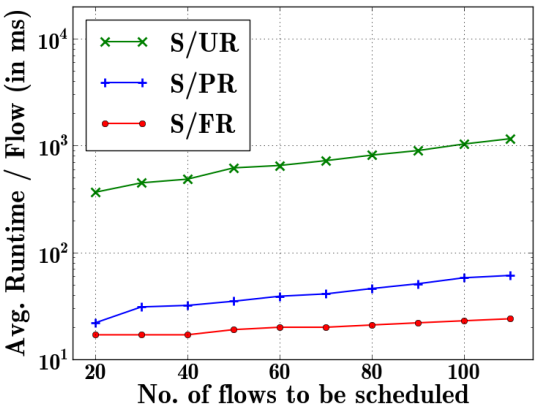
6.5.2 Control Plane Evaluations for TSSDN

In this section, we evaluate the various ILP formulations, presented in Section 6.3, with respect to the quality of schedules they compute and their scalability.

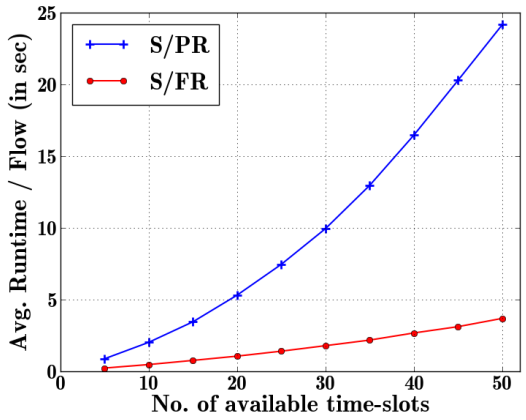
We use the commercial ILP solver CPLEX from IBM to solve our ILP formulations which are specified using PuLP [89], a Python-based tool-kit to specify ILPs. Moreover, we created different network topologies (different sizes and different network models) using NetworkX [62], a Python library for creating complex networks. In detail, we used the Erdős-Rényi (ER) model [60] (random graphs where nodes have *similar* degree), random regular graphs (RRG) (random graphs where nodes have same degree), the



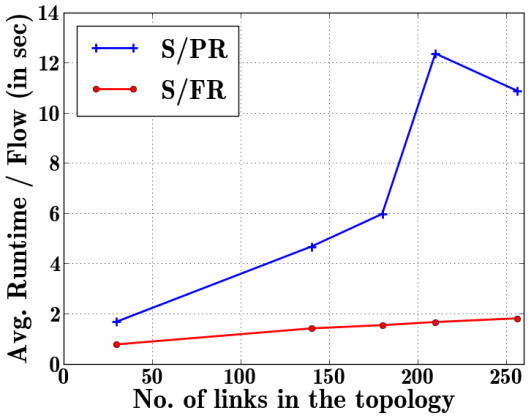
(a) Quality comparison of the ILPs



(b) Scheduling runtime vs No. of flows



(c) Scheduling runtime vs No. of slots



(d) Scheduling runtime vs Topo. size

Figure 6.4: Control Plane Evaluations Results

Barabási-Albert (BA) model [61] (scale-free networks where few nodes have high degree and many have small degree), and the Waxman model [90] (geographic model favoring short-distance links over long links). Together, these models for randomized graphs comprehensively test the limits of our ILP formulations. The sizes of these topologies and the number of time-slots and flows used as input are specified with the concrete evaluations.

We used two machines for evaluating our ILPs. The first is a high performance multi-processor machine with 2×8 cores (Intel Xeon E5-2650) and 128 GB RAM, while the second is a commodity machine with 2 cores and 8 GB RAM.

Qualitative Evaluations

To evaluate the quality of the schedules generated by the ILP formulations S/PR and S/FR with respect to S/UR, we computed the transmission schedules in 160 evaluation

scenarios using 8 different topologies (3 RRG, 2 ER, and 3 BA), each with 24 hosts and 6 switches. Note that we had to choose a smaller topology to be able to compute the schedule using S/UR as reference since it has an impractical runtime. Limiting the number of components in the topology also limited the number of topologies we could examine. Each scenario consisted of 20–110 flows with random source and destination hosts to be scheduled with 3–5 available time-slots in the network. We have deliberately chosen a smaller number of slots to create challenging scenarios for our ILP formulations even for smaller numbers of flows. As a performance metric, we calculate the relative quality of the schedules computed by S/PR and S/FR, i.e., the ratio of the number of flows scheduled by them to the number of flows scheduled by S/UR.

Figure 6.4a shows the cumulative distribution of the relative quality achieved by S/PR and S/FR. This figure shows that the quality of the solutions they generate closely approximate the quality of the ones computed using S/UR. For instance, for S/PR, 80 % of the scenarios have at least a relative quality of 98 % or better. In detail, S/PR and S/FR generated schedules with 100 % relative quality in about 67 % and 38 % of the evaluation scenarios with average relative qualities of 99 % and 97 %, respectively.

Scalability Evaluations

Knowing the quality of the different approaches, we next evaluate their scalability, i.e., the time to calculate solutions for different scenario sizes. Our evaluations show that the runtime for computing the transmission schedule depends mainly on three factors: the number of flows to schedule, the number of available time-slots, and the size of the topology. The model on which the topology is based has no influence on the execution times of the scheduling algorithms.

First, we vary the number of flows for scheduling using the ILP formulations. We use a small scenario, an ER topology consisting of 24 hosts and 6 switches (38 network links) with 5 time-slots for disbursement, to measure the runtime for computing schedules using our various approaches. We measure the runtime for computing the schedules with a varying number (20–110) of flows on our high performance machine. As shown in Figure 6.4b, the runtime for computing the schedules using S/PR and S/FR is at least an order of magnitude lower than that for computing it using S/UR. As per our evaluations, S/PR and S/FR could compute schedules for over 100 flows in approximately 7 sec and 3 sec, respectively, while computing the schedule using S/UR required over 2 min. This translates to an average scheduling time of 1.1 sec, 61 msec, and 24 msec, per flow for S/UR, S/PR, S/FR, respectively. We observed similar or worse results with execution times running into several hours to days for computing schedules using S/UR on other topologies of comparable scale.

Next, we vary the number of available time-slots to evaluate its impact on the runtime of the ILP formulations. For this and subsequent evaluations, we execute the ILP solver on the commodity machine and do not use the S/UR approach as the scenarios

are too large for computing a schedule with it. Here, we use a topology with 200 hosts and 10 switches (256 network links) based on the Waxman model. We scheduled 300 flows on this topology using the ILP formulations and measured the average time to schedule a flow. The number of time-slots were varied between 5–50. As shown in Figure 6.4c, the runtime increases rapidly for the S/PR approach with an increasing number of available time-slots in contrast to the S/FR approach, which scales much better (approximately linearly with number of time-slots). It may be noted that a network with 1 Gbps links and a network diameter of 8 hops provides only about 50 slots (considering MTU as 1500 bytes) for a base-period of 1 ms. Moreover, assuming that a cyber-physical system (CPS) comprises two flows (one from sensor to the CPS controller and other from the CPS controller to the actuator), schedules for supporting up to 150 CPS can be calculated by our ILPs. Thus, we claim that our ILP formulations scale well for realistic scenarios.

Finally, we evaluated the impact of topology size (number of network links) on the runtime of the ILP formulations. For this evaluation, we used different topologies (30–256 network links) and scheduled over 100 flows on them with 50 time-slots for disbursement. Figure 6.4d summarizes the measured runtimes for S/PR and S/FR. We observe that the runtime of S/FR increases linearly with the size of the topology and takes on an average less than 2 sec to schedule a flow in a topology containing 256 links. For the S/PR approach, the runtime is not directly related to the topology size. It, rather, depends on the number of shortest paths between the sources and the destinations of the flows, i.e., the path diversity of the network. Nonetheless, the worst case average time to schedule a flow on a topology with 200 links was just over 12 sec for this ILP formulation.

It may also be noted that in all of our experiments to evaluate runtimes for the algorithms, repeated runs yielded similar runtimes with very low variances.

Computation of Base-period

We also evaluated the runtime for computation of base-period for a set of flows. The evaluations showed that the computation of the base-period for the flows to be scheduled in the network increases linearly with the number of flows. As shown in Fig. 6.5, the base-period for up to 10^5 flows in TSSDN can be determined in 150 sec on commodity machines. Thus, it is reasonable to compute the base-period for the flows upfront.

6.5.3 Evaluation Summary

In summary, our evaluations showed:

1. TSSDN provides virtually constant end-to-end latency (std. dev. $< 1\,\mu$s) with worst case jitter $\leq 7\,\mu$s for the time-triggered traffic on our benchmark topology.

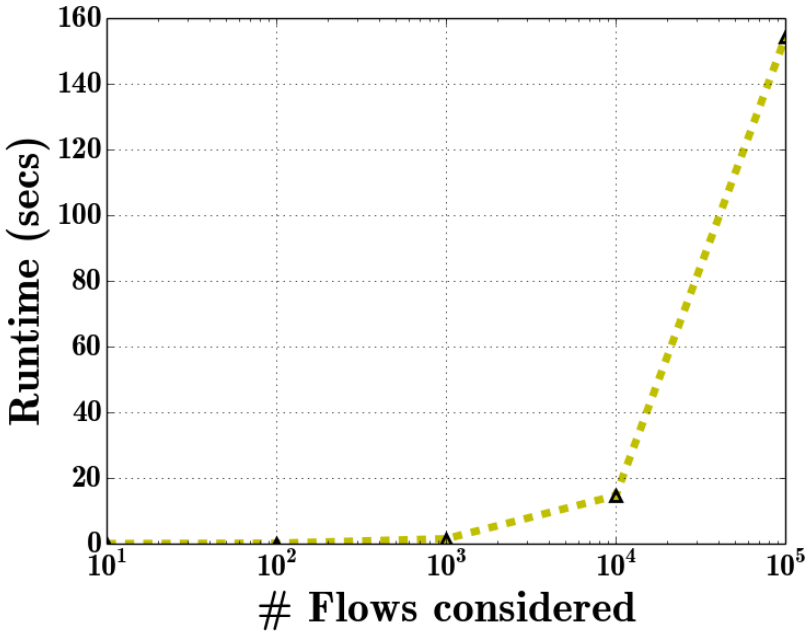


Figure 6.5: Base-period computation

2. The S/PR and S/FR approaches for computing transmission schedules closely approximate the solution computed by S/UR (which provides optimal solutions in most practical cases), by constraining routing, and thus, resulting in lower runtimes.

3. Our ILP formulations, S/PR and S/FR, scale well to compute schedules for over 300 flows in networks with more than 200 network links with a data-rate of 1 Gbps ($\approx$ 50 time-slots assuming a base-period of 1 ms).

6.6 Related Work

As described in the previous chapters, there is a strong trend to make the widely adopted IP-based networks and IEEE 802 networks ready for real-time traffic. These developments are driven, in particular, by the IEEE 802.1 Time-Sensitive Networking Task Group [16] and the IETF DetNets Working Group [23]. We intentionally base the design of TSSDN on the basic principles conforming with the initial proposals of these standards bodies like synchronized end systems and logically centralized configuration. This directly makes our contributions like the scheduling algorithms—which have not been considered by these groups so far—applicable to upcoming standards.

Software-defined networking, being a rather recent development and primarily foreseen for usage in data centers, does not yet extensively look to provide real-time properties for communication. Nonetheless, there are quite a few noteworthy contributions which we review here. Qian et al. present [91] routing algorithms for real-time communication

flows such that their deadlines are respected and implement a EDF-based scheduler on the network elements using virtual SDN switches to show that the deadlines of the flows are met. Kumar et al. present a framework for synthesizing routes and switch configurations for isolating real-time flows into separate queues per outgoing switch-port using SDN [92]. By means of evaluation on a white-box switch, they show that such an isolation leads to stable delays for the real-time flows. However, queues being a limited resource in networks, the approach to provide each real-time flow with its own queue does not scale. Moreover, their evaluations exhibit significant variance in end-to-end latencies when more than 4 flows are routed over the same switch port.

QJump is yet another approach which exploits the queues in the network to provide different levels of latency guarantees to the traffic [93]. QJump trades off throughput with latency variance and allows the sending applications to choose between the different trade offs. Applications can either send real-time traffic which is strictly rate-limited but is provided with bounded latency or non-real-time traffic with high throughput but at the same time high latency variance.

Scheduling and routing traffic with the objective to eliminate incurred queuing delays and providing low latencies in data center networks is a known approach. Vattikonda et al. presented a system to enforce computed schedules by means of the 802.3x flow control and a centralized link scheduler in network with commodity hardware without relying on clock synchronization [94]. Fastpass extends this idea further by integrating scheduling and routing using a centralized arbiter for data center networks [95]. However, the scheduling in these systems is done for sporadic traffic generated by the hosts in data center networks, rather than for sources of time-triggered traffic. Moreover, the scheduler in Fastpass restricts the topologies that can be deployed to rearrangably non blocking (RNB) networks like the Clos networks for faster computation of schedules and routes [96]. Overall, these approaches are specially tailored for the traffic in data centers which are rather soft real-time in nature [97] [98].

There have also been attempts to incorporate deadline awareness for communication flows in transport and network layer of the stack, e.g., Deadline-aware Data-center TCP [99], Preemptive Distributed Quick (PDQ) flow scheduling [100], D^3 [101], etc. These approaches aim to maximize the number of flows meeting their corresponding deadlines but do not consider their latency variance.

Schweissguth et al. also propose the usage of SDN for handling time-triggered traffic by means of a TDMA scheme [102]. While their solutions for the scheduling and routing problem are iterative in nature, we deal with ILP formulations which provide exact solutions for our optimization problem. Along with our work, this contribution is among the first scientific contributions to exploit the logical centralization of software-defined networking for handling time-triggered communication flows in a production shopfloor.

The related work on handling time-triggered traffic using scheduling and routing to isolate the corresponding flows from other classes of traffic using specialized hardware is already presented in Section 4.8 and 5.5.

6.7 Summary

In this chapter, we presented Time-sensitive Software Defined Networks, which provide real-time guarantees for communication of time-triggered traffic by means of transmission scheduling at the source hosts only. As a first step, we presented a set of ILP formulations that compute transmission schedules for a given set of pre-defined time-triggered flows in a network topology. Our evaluations showed that the ILPs efficiently calculate high quality schedules and the adherence to these schedules result in deterministic network delays. An interesting open question which we also left for future work is, how much jitter can be reduced with hardware support (specialized NIC or NetFPGA) or by integrating real-time operating systems (e.g. RTnet [103]) for bounding the jitter in the source host.

CHAPTER 7

INCREMENTAL SCHEDULING AND ROUTING IN TSSDN

7.1 Introduction

We introduced the Time-sensitive Software-defined Network (TSSDN), an SDN-based architecture, for handling time-triggered traffic along with best-effort traffic in Chapter 6. In TSSDN, the transmissions of time-triggered flows are scheduled on the source hosts only, while its simplistic data plane, consisting of switches equipped with in-network prioritization mechanisms (and optionally frame pre-emption mechanisms), only forwards these packets. However, the presence of best-effort traffic and the lack of additional hardware enhancements in the data plane for isolating the scheduled traffic from other classes of traffic results in a small but bounded jitter for time-triggered flows, i.e., the latency bounds guaranteed by TSSDN are slightly relaxed as compared to the bounds that are achievable using specialized hardware enhancements like the programmable gating mechanisms in the IEEE 802.1Qbv standard for handling scheduled traffic. We have already discussed the integrated scheduling and routing problem for static scenarios in TSSDN along with solutions of varying time complexities.

In this chapter, we discuss the dynamic scheduling problem in TSSDN for incrementally setting up applications in the network, for instance, adding "plug-and-produce" devices in the production lines [104]. The challenge for incremental set-up of time-triggered flows in TSSDN is that disturbances to the existing flows due to schedule modifications must be avoided, as it may lead to a temporary violation of bounds on the delays or the jitter with undesirable consequences [105]. Moreover, to be deployable, the solutions to the dynamic scheduling problem are required to have low execution times (less than a few seconds), and hence, the existing methods for offline/static scheduling cannot be used directly due to their high runtime in the order of hours. The fact that the

transmissions in TSSDN are scheduled on the hosts only rather than on the switches also enables TSSDN to provide lightweight mechanisms to swiftly update schedules for dynamically changing time-triggered traffic. However, the lack of information on flows that would be needed to set-up in the future makes it hard for computing schedules which are as good as the ones resulting from the corresponding static scheduling problem where the schedules are computed exploiting *a priori* information of the time-triggered flows.

Our scientific contributions to overcome the challenges of dynamic scheduling problem in TSSDN are manifold. We use the features of OpenFlow to devise methods for adding/removing time-triggered flows from the network without affecting the already scheduled flows. We also formulate the dynamic scheduling problem in TSSDN using Integer Linear Programs (ILP) and provide heuristics which can be used for approximating the optimal solutions of the corresponding static scheduling problem. In order to bring the runtime of our solution under a second, we propose several optimizations while having minimal effects on the quality of the computed solutions in terms of the number of flows accommodated in the network.

This chapter is organized as follows. We describe the system model of TSSDN with respect to the dynamic scheduling problem and concretely define the objective of our solutions in Section 7.2. The ILP formulations and the relevant heuristics for the incremental scheduling problem is discussed in Section 7.3. We discuss the optimizations for reducing the runtime of the scheduling approaches to under a second in Section 7.4. The evaluations and related work are discussed in Section 7.5 and 7.6, respectively.

7.2 System Model & Problem Statement

7.2.1 System Model

The system model for TSSDN is already presented in Section 6.2. In this section, we present other considerations in TSSDN with respect to the dynamic scheduling of time-triggered flows like the update procedure for the transmission schedules.

TSSDN is an SDN-based architecture in which the control plane computes the transmission schedules for time-triggered flows based on its global view of the data plane consisting of commodity SDN switches. The scheduling model is based on cyclic schedules implementing a time division multiple access (TDMA) scheme. The transmission schedule of length equalling the base-period, t_{bp}, is divided into time-slots, each wide enough for an MTU-sized packet to traverse across the network diameter. The TSSDN scheduler allocates time-slots to time-triggered flows such that flows with overlapping paths are always assigned different time-slots. For static scenarios, the network controller computes offline schedules and updates the flow tables of the switches for routing the time-triggered traffic while also sending the schedule information to the source hosts of the flows. However, there are additional steps required to dynamically schedule

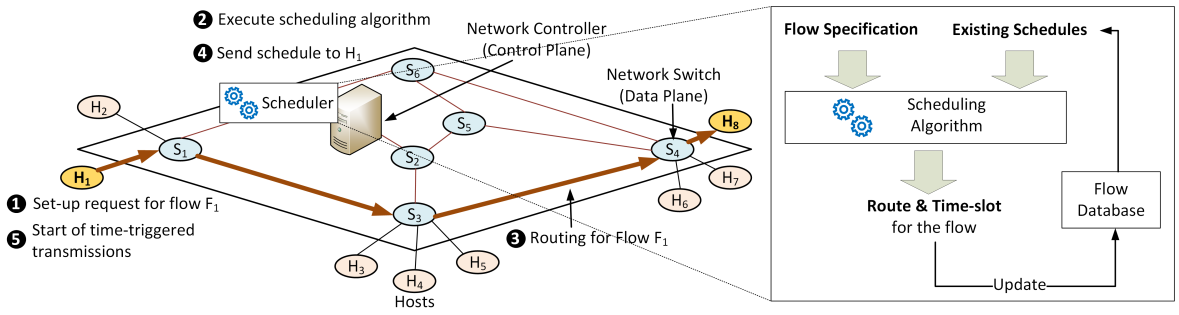


Figure 7.1: Steps involved during incremental scheduling of flow F_1 ($H_1 \rightarrow H_8$) in TSSDN.

time-triggered flows in TSSDN for ensuring that the already scheduled flows are not affected.

The flow set-up procedure for time-triggered flows in TSSDN is shown in Figure 7.1. To schedule an additional flow (say Flow F_1:$H_1 \rightarrow H_8$), the source host (H_1) sends a flow set-up request containing the flow specifications (the source host, the destination host(s), the transmission period, etc.) to the network controller in Step 1. In Step 2, the network controller executes an online scheduling algorithm (focus of this chapter) to determine if the flow can be accommodated in the network. If a suitable schedule (time-slot and route) exists, then the network controller programs the flow table entries in the switches for routing the packets belonging to the flow over the corresponding path (Step 3). In Step 4, the source host (in this case H_1) is informed about the schedule by the network controller. The source host can then commence their time-triggered transmissions in Step 5.

The network controller uses exact match semantics for the flow-table entries, i.e., no flow aggregation. Moreover, the network controller uses a transactional interface (available since OpenFlow v1.4) for this purpose so as to ascertain that the updates are indeed applied before sending the schedules to the source hosts. This way the controller ensures that existing flows are not affected and that the network updates are consistent, i.e., no loops/blackholes are created in the data plane [106]. A source host requiring to send multiple packets per cycle must request for a corresponding number of time-slots in Step 1.

When flows are no longer required, the source host sends a flow removal request to the network controller. Based on the request, the controller updates its flow database and removes the corresponding flow table entries. The freed up time-slots are used for scheduling new incoming flows. SDN protocols like OpenFlow may be extended for the communication between the hosts and the network controller. Overall, the schedule update procedure is limited to inserting flow table entries for new flows and sending schedule information to the source hosts. Since switches play no role in enforcing schedules, they need not be reconfigured, resulting in lightweight schedule updates.

In dynamic scenarios, it may be required to change the base-period of TSSDN, for instance, due to a new time-triggered flow with time-period lower than the base-period, due to exhaustion of the time-slots, or due to sub-optimality of the chosen base-period. However, modifying the base-period has high coordination overhead as the transition to the new schedules must be an atomic operation to avoid schedule inconsistencies between different parts of the network. In particular, reducing the base-period would result in a reduction in the number of available time-slots. Already allocated time-slots may cease to exist, requiring a reallocation for the affected flows. The relative difference between the positions of the new and old time-slots in the overall schedule, for instance, when the time-slot for a flow is moved from the beginning of the schedule to the end, introduces jitter during the transition. Moreover, such transitions may also need re-routing of flows over paths of lengths differing from the initial paths contributing to the induced jitter. Hence, TSSDN does not allow runtime modifications to the base-period, which is calculated offline and remains fixed. For the dynamic scheduling problem, the base-period is computed using a set of flows that are expected to be scheduled in the network using the method described in Section 6.3.2. The computed base-period would be obviously sub-optimal if flows not belonging to this expected set are scheduled.

7.2.2 Problem Statement

In the dynamic or online version of this problem, the focus is on computing schedules faster while approximating the results of the corresponding static scheduling problem.

Our solutions to the dynamic scheduling problem strive to reduce (not bound) set-up times for time-triggered flows to sub-seconds by optimizing the scheduling algorithms only. The minimization of latencies between the hosts, switches, and the network controller is an orthogonal problem, and is out of the scope of this work [107]. Furthermore, the delays in inserting flow-table entries in switches is hardware dependent. State-of-the-art SDN switches insert entries with same priorities in less than 15 ms [108], and hence, do not contribute to the flow set-up delay significantly for TSSDN.

7.3 Dynamic Scheduling in TSSDN

In the dynamic scheduling problem, flows are scheduled incrementally as required in the network. Obviously, scheduling and routing a flow with no prior knowledge of flows that would arrive in the future would yield sub-optimal solutions in comparison to the corresponding static scheduling problem where all the flows are known *a priori*. Hence, heuristics like the number of hops, number of occupied time-slots on each link, etc., are used by the online scheduling algorithms to approximate the solutions generated by the static scheduling algorithm.

In the following, we restrict the ILP formulations to schedule only one flow at a time, i.e., we do not schedule in batches, for two reasons. Firstly, lack of a feasible schedule

Helper Function	Parameters	Output
$\text{in}(n)$	$n \in V$	$\{(u,v) \in E \mid v = n\}$
$\text{out}(n)$		$\{(u,v) \in E \mid u = n\}$
$\text{src}(F)$	Flow F,	s
$\text{dst}(F)$	$F \equiv (s, D, p)$	D

Table 7.1: Helper functions for the ILP formulations

for a single flow will result in the ILP solver not being able to compute schedules for any of the flows in the batch. Secondly, we observed in our preliminary evaluations that the runtime per flow for the scheduling algorithms increase with the number of flows in the batch. The is because the solver must now not only resolve the conflict between the flow being scheduled and the already scheduled flows but also between all the flows belonging to the batch. This implies that the additional runtime for scheduling flows in a batch does not overall amortize.

7.3.1 Terminology

The terminologies used for the presented ILP formulations are slightly different from the ones used in Section 6.3 due to additional parameters that need to be modelled here. Hence, we repeat the terminology and notations used in the ILP formulations of this chapter for the sake of better comprehension.

The network topology is modelled as a directed graph $G \equiv (V, E)$. Here, V is the set of nodes and $E \equiv \{(i,j) |\ i, j \in V$ and i, j are connected by a network link$\}$ is a set of tuples representing the network links. Further, $V \equiv (SW \cup H)$, where SW and H are sets of switches and hosts, respectively. A time-triggered flow is denoted as a tuple, $F \equiv (s, D, p)$, where $s \in H$, $D \subseteq H$, and $p \in \mathbb{Z}^+$. Here, s is the source, D is the set of destinations of the flow, and p denotes the transmission period of the flow. Thus, we model time-triggered flows as multicast flows. The set of time-slots for scheduling is denoted as $T \equiv \{0, 1 \ldots, t_{max}\}$.

Additional functions needed to represent the topology and the flows are listed in Table 7.1.

7.3.2 Shortest Available Path (D/SAP)

The objective of this ILP is to *minimize the number of links over which a flow is routed* while allocating a time-slot for it during which all the constituting links are unoccupied. This results in a larger number of free links during the corresponding time-slot for future flows.

ILP Inputs

The inputs for *D/SAP* are:

- Network topology as *Directed Graph*, $G \equiv (V, E)$.
- *Set of time-slots*, T. $T \equiv \{0, 1, \ldots, t_{max}\}$.
- *Specification of the flow* to be set-up, $F \equiv (s, D, p)$.
- *Existing Schedules*, ES.
  $ES \equiv \{es_{i,j}\}; \forall\, i \in E, \forall\, j \in T$.
  $es_{i,j} = 1$, if any scheduled flow traverses link i at time-slot j, else 0.

ILP Variables

The variables for *D/SAP* are:

- *Link Occupancy*, LO. This indicates the set of links over which the flow F must be routed. $LO \equiv \{lo_i\}; \forall\, i \in E$.
  $lo_i = 1$, if flow F is routed over link i, else 0.
- *Destination Count*, DC. This indicates the number of destinations of flow F reachable over a given link i if the flow is routed over it. $DC \equiv \{dc_i\}; \forall\, i \in E$.
  $dc_i =$ number of flow destinations that may be reached over link i if flow F is routed over i, else 0.
- *Slot Occupancy*, SO. This indicates the time-slot that must be allocated to flow F. $SO \equiv \{so_j\}; \forall\, j \in T$.
  $so_j = 1$, if the flow is allocated slot j, else 0.
- *Auxiliary variables*, A. These variables enable the formulation of the scheduling problem using linear constraints.
  $A \equiv \{a_{i,j}\}; \forall\, i \in E, \forall\, j \in T$.
  $a_{i,j} = 1$, if flow F is routed over link i and allocated slot j, else 0.

Objective Function

The objective function for *D/SAP* minimizes the number of links used for routing the flow.

$$Minimize \sum_{\forall i \in E} lo_i \tag{7.1}$$

Constraints

The constraints for *D/SAP* are as follows:

- *Time-slot constraint*: Flow F shall be allocated exactly one time-slot.

$$\sum_{\forall j \in T} so_j = 1 \tag{7.2}$$

- *Routing constraint*: The route for flow F starts at the source host and ends at the destination host(s), i.e., the number of destinations reachable over the outgoing links of the source host is equal to the number of destinations of the flow, while the number of destinations reachable over the incoming links of the destination hosts is 1. For all the other nodes in graph G, the sum of destinations reachable over incoming links is equal to the sum of destinations reachable over outgoing links.

$$\begin{aligned} \sum_{\forall i \in \text{in}(s)} dc_i = 0 \qquad & \sum_{\forall i \in \text{out}(s)} dc_i = |D| \\ \sum_{\forall i \in \text{in}(n)} dc_i = 1 \qquad & \sum_{\forall i \in \text{out}(n)} dc_i = 0 \qquad \forall\ n\ \in\ D \end{aligned} \tag{7.3}$$

$$\sum_{\forall i \in \text{in}(n)} dc_i = \sum_{\forall i \in \text{out}(n)} dc_i \qquad \forall\ n\ \in\ V \setminus (\{s\} \cup D) \tag{7.4}$$

- *Collision avoidance constraint*: Flow F must be routed and allocated a time-slot such that all network links are occupied by at most one flow at any given time-slot.

$$a_{i,j} + es_{i,j} \leq 1 \qquad \forall\ i \in E, \forall\ j \in T \tag{7.5}$$

- *Auxiliary constraints*: For consistency among the ILP variables, the following constraint is required:

$$a_{i,j} = lo_i \times so_j \qquad \forall\ i \in E, \forall\ j \in T \tag{7.6}$$

This non-linear constraint is modelled by the following set of linear constraints:

$$\left.\begin{aligned} a_{i,j} &\leq lo_i \\ a_{i,j} &\leq so_j \\ a_{i,j} &\geq lo_i + so_j - 1 \end{aligned}\right\} \qquad \forall\ i \in E, \forall\ j \in T \tag{7.7}$$

Additionally, the following constraint is required to model the relation between

DC and LO. For a link i, $dc_i > 0$ implies the link is used for routing, i.e., $lo_i = 1$.

$$lo_i \times |D| \geq dc_i \qquad \forall\, i \in E \tag{7.8}$$

A feasible solution to this ILP provides a suitable schedule for flow F. The values of LO and SO determine the allocated time-slot and route, respectively. In absence of a feasible solution, flow F cannot be accommodated in the network.

7.3.3 Mini-max (D/MM)

Solely minimizing the number of links for routing a flow may result in some scenarios in formation of bottleneck links which are occupied during all time-slots. The second approach for online scheduling, Mini-max (D/MM), aims to distribute spare capacity over all links in the network to improve the prospects of successfully accommodating any future flow at the cost of routing some flows over longer paths. This ILP builds on the formulation of *D/SAP* by modifying its objective and adding a constraint. An additional decision variable, *MaxSlots*, is also added. This variable defines the maximum number of time-slots during which any of the network links may be occupied. Thus, $MaxSlots \in \mathbb{Z}^+$.

Here, the primary objective is the *minimization of the maximum number of time-slots during which any of the network links may be occupied.* Hence the name *mini-max.* The secondary objective minimizes the number of links over which the flow is routed to weed out undesirable solutions that route the flow being scheduled over paths containing loops.

$$Minimize \quad \underbrace{MaxSlots}_{\text{Primary Objective}} + \underbrace{\frac{1}{|E|+1} \times \sum_{\forall i \in E} l_i}_{\text{Secondary Objective}} \tag{7.9}$$

The additional constraint in *D/MM* compared to *D/SAP* does not allow any network link to be occupied for number of time-slots exceeding *MaxSlots*, which is being minimized.

$$\sum_{\forall j \in T} es_{i,j} + lo_i \leq MaxSlots \qquad \forall\, i \in E \tag{7.10}$$

Similar to *D/SAP*, LO and SO determine the route of the flow and the allocated time-slot, respectively.

7.3.4 Dynamic Scheduling of Flows With Different Periods

The presented approaches assume that the transmission periods for all flows equal the base-period. One time-slot is allocated to each of the flows irrespective of their individual transmission period. It is, however, beneficial to also appropriately handle flows

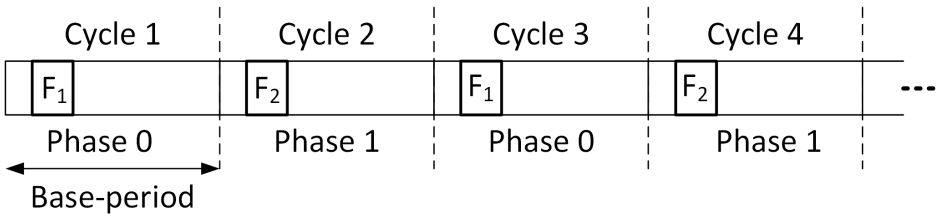


Figure 7.2: Phasing of flows F_1, F_2 with periods $2 \cdot bp$

which have transmission periods higher than the base-period in the dynamic scheduling problem. To avoid over-allocation of capacity, flows with equal transmission periods can be phased, i.e., for every flow being scheduled a *phase* is allocated along with a time-slot, i.e., a $\langle$*time-slot*, *phase*$\rangle$ pair is allocated. *Phase* indicates the validity of the *time-slot* during a given cycle of the schedule. A flow can utilize its allocated time-slot only if the allocated phase matches the corresponding cycle of the schedule. Two flows with equal transmission periods can be allocated the same time-slot despite traversing overlapping paths so long as they are allocated *different phases*, thus, ensuring temporal isolation for the flows. For instance, in Figure 7.2, flows F_1 and F_2 with transmission periods of $2 \cdot bp$ traversing over overlapping paths can use the same time-slot but in alternate cycles; Flow F_1 uses the time-slot during *Phase 0*, while flow F_2 uses it during *Phase 1*.

The total number of phases available for allocation on a link for a given time-slot depends on the first flow that acquires this time-slot and is routed to traverse the link. If a flow with transmission period $n \cdot bp$ is routed to traverse over link l_i at time-slot t_j, then n phases ($[0, \ldots, n-1]$) are available on the (l_i, t_j) pair. A subsequent flow with period $n \cdot bp$ can use time-slot t_j despite traversing link l_i provided that it is allocated a phase different from all the preceding flows.

The presented ILP formulations for the dynamic scheduling problem, *D/SAP* and *D/MM*, schedule flows over the set of time-slots, T, allocating one time-slot for each flow. To use phasing, it must also allocate a phase to each of the flows. For this, an additional input is provided to the ILP formulation, namely the number of available phases, $P \equiv \{0, \ldots, n-1\}$, where $n \cdot bp$ is the period of the flow being scheduled. The ILP formulations must schedule over $(T \times P)$ instead of T and thus allocate a $\langle$time-slot, phase$\rangle$ pair for each flow being scheduled. To this end, all references to T in the ILPs—*D/SAP* and *D/MM*—are replaced with $(T \times P)$.

Additionally, the input *Existing Schedules*, ES, in the ILP formulations is redefined as $ES \equiv \{es_{i,(j,k)}\}$; $\forall i \in E$, $\forall \langle j, k \rangle \in (T \times P)$. Here, $es_{i,(j,k)} = 0$, when no flow is scheduled to traverse link i at time-slot j *or* the transmission period of the flow being scheduled is equal to the periods of flows traversing link i at time-slot j, *and* none of these flows are allocated phase k. In all other cases, $es_{i,(j,k)} = 1$.

With these modifications, *D/SAP* and *D/MM* compute not only a time-slot but also a phase for the flow being scheduled along with the route for the flow, and thus, handle flows with different transmission periods.

7.4 Optimizations for Dynamic Scheduling Approaches

The presented approaches for dynamic scheduling, *D/SAP* and *D/MM*, have computational runtime in the order of several seconds (cf. Section 7.5). The runtime mainly depends on the size of the topology ($|E|, |V|$), the number of available time-slots ($|T|$), and the number of phases considered for scheduling ($|P|$). The following optimizations are introduced to reduce the execution time to sub-second range.

Topology Pruning

With topology pruning, the ILPs consider only a subset of the network topology, $G' \equiv (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, for scheduling. To this end, the topology is pruned to remove network links along with nodes that do not play any role in routing a given flow, $F \equiv (s, D, p)$.

It may be observed that not all edge links (links connecting end systems to the network) play a role in routing flows. Only those edge links are used that connect the source and destination hosts of the flow with the network. The others may be removed without any impact on the computed solutions. Thus, the pruned topology is defined as follows:

- $V' \equiv SW \cup \{s\} \cup D$, where SW is set of switches.
- $E' \equiv \{(n_1, n_2) \mid (n_1, n_2) \in E;$
  $(n_1, n_2 \in SW) \vee (n_1 = s) \vee (n_2 \in D)\}$.

Time-slot Slicing

Another factor that affects the computational runtime is the number of time-slots and phases over which the flow is to be scheduled. To reduce the execution time, only a subset of ⟨time-slot, phase⟩ pairs referred to as time-slot slice, $TP \subseteq (T \times P)$, may be considered.

Use of time-slot slicing may lead to sub-optimal solutions (longer paths in case of *D/SAP* or paths resulting in bottleneck links in case of *D/MM*) leading to fewer flows being accommodated overall. Worse, this may also introduce false-negatives, i.e., the scheduler is unable to allocate a suitable ⟨time-slot, phase⟩ pair and corresponding route for the flow as all the possible solutions lie in $(T \times P) \setminus TP$. It is thus imperative that TP is constructed such that the resulting sub-optimality and number of false-negatives are minimized. For this, the number of unoccupied core links (links between

two switches) during a given ⟨time-slot, phase⟩ pair is used as a heuristic to generate TP from $(T \times P)$. A higher number of unoccupied core links during a ⟨time-slot, phase⟩ pair improves the prospects of yielding a solution to the scheduling problem. For instance, if all core links are unoccupied during a ⟨time-slot, phase⟩ pair, it is certain that the optimal solution will be available, provided that the necessary edge links (connecting source and destinations of the flows to the network) are also unoccupied during the corresponding ⟨time-slot, phase⟩ pair.

Algorithm 5 Generate time-slot slice, *TP*

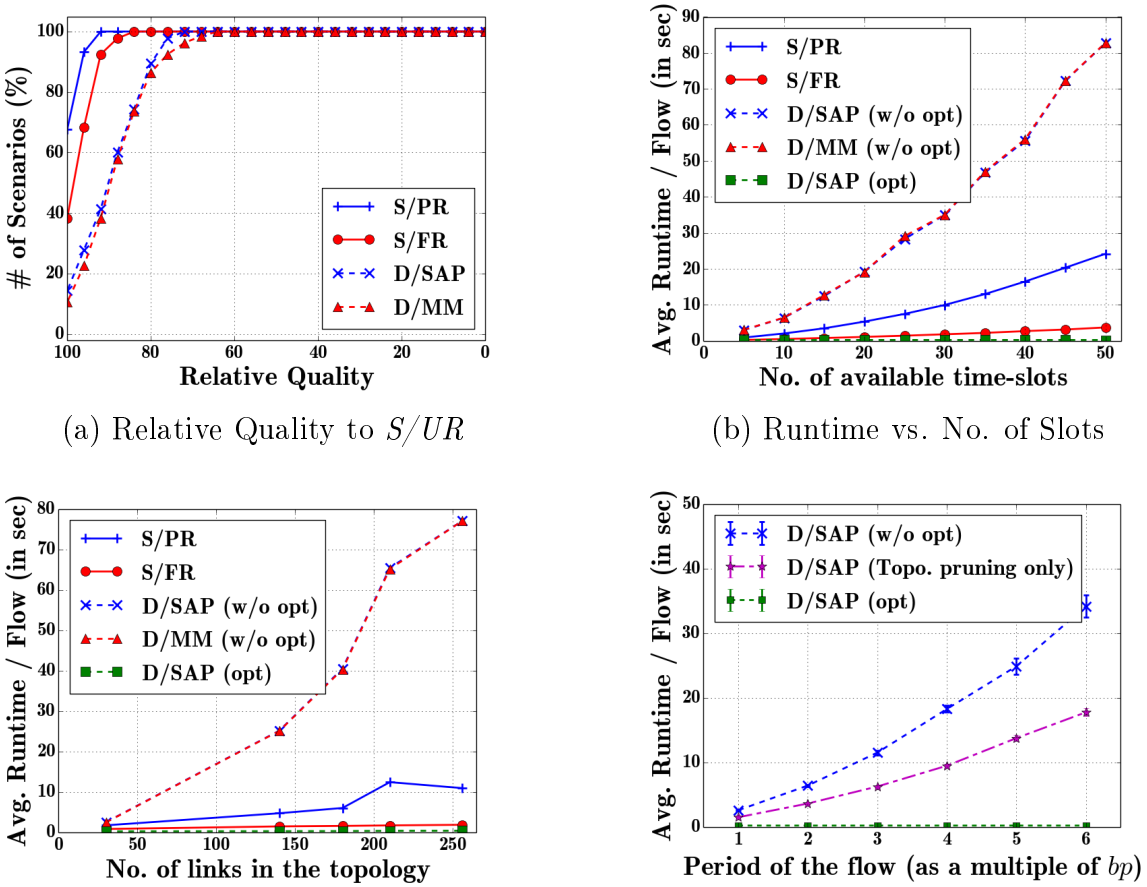
```
 1: function GenerateTP(T, P, G, SliceSize, Schedule)
 2:     TP ← φ                                  ▷ List of time-slot, phase pairs
 3:
 4:     for t in T do
 5:         for ph in P do
 6:             freeCoreLinks(G, Schedule, ⟨t, ph⟩) ▷ Compute free core links at ⟨t, ph⟩
 7:         end for
 8:         TP.append((t, getBestPhase()))
 9:     end for
10:     TP ← sort(TP)          ▷ Sort TP based on number of available core links
11:
12:     return TP[0 : SliceSize]
13: end function
```

Using the aforementioned heuristic, a time-slot slice, TP, of size *SliceSize* can be computed for scheduling as shown in Algorithm 5. The algorithm computes the heuristic for all $\langle t, ph \rangle \in (T \times P)$ (Line 6). For each $t \in T$, a phase, ph, that provides the best chance for the scheduler to compute the optimal solution is short-listed (Line 8). Phase ph provides the maximum number of unoccupied core links during the corresponding time-slot, t. Further, during $\langle t, ph \rangle$, all the edge links required for routing the particular flow, i.e., the links connecting the source and destination hosts to the network, must also be unoccupied. From all the short-listed $\langle t, ph \rangle$ pairs, the best *SliceSize* number of pairs in terms of the computed heuristic form the time-slot slice (Lines 10–12).

Here, the parameter *SliceSize* determines the execution time for scheduling along with the introduced sub-optimality and false-negatives. Low values of *SliceSize* result in lower execution time, but at the cost of an increased number of sub-optimally routed flows and a higher number of false-negatives. To reduce false-negatives, scheduling can be reattempted with an increased value of *SliceSize*. In such cases, those ⟨time-slot, phase⟩ pairs that did not yield any solution in the previous attempts must be excluded.

Together with topology pruning, time-slot slicing realises sub-second runtime for the dynamic scheduling algorithms in TSSDN.

7.5 Evaluations



(a) Relative Quality to *S/UR*

(b) Runtime vs. No. of Slots

(c) Runtime vs. No. of Links

(d) Runtime vs. Period of flow

Figure 7.3: Evaluations Results for the ILP formulations presented in Section 7.3

This section presents the results of the evaluations for the dynamic scheduling approaches in TSSDN. The evaluations are mainly focussed on the computational time and quality of the dynamic scheduling approaches in comparison to the static scheduling approaches in TSSDN. Hence, we extend the evaluations from Chapter 6 with graphs for dynamic approaches. For evaluations on the interactions between the data plane and the control plane in SDN and the latencies incurred in insertion and deletion of flow-table entries in switches, we refer the reader to existing literature [108] [109] [110].

The evaluations use CPLEX(v 12.5) [63], a commercial ILP solver from IBM, for computing the schedules and routes by solving the presented ILPs. The ILPs were specified using PuLP [89], a Python-based tool-kit to specify ILPs. The scheduling approaches were evaluated against a range of network topologies (various sizes and network models) created using NetworkX, a Python library for creating complex networks. In detail, topologies were created using the Erdős-Rényi (ER) model [60], random regular graphs (RRG), the Barabási-Albert (BA) model [61], and the Waxman model [90]. Altogether, these graph models comprehensively test the scheduling approaches. The sizes of the topologies, the number of time-slots, and the flows used as input are specified with the concrete evaluations.

7.5.1 Qualitative Comparison

For a given set of flows with transmission period equal to the base-period, *S/UR* yields the best schedule in terms of the number of scheduled flows. Hence, *S/UR* is used as benchmark for comparing the scheduling approaches.

For comparison, evaluations were carried out in 160 scenarios created using 8 different topologies (3 RRG, 2 ER and 3 BA), each consisting of 20–110 unicast flows whose transmission periods were equal to the base-period. These flows were scheduled over 3–5 time-slots. The high runtime of *S/UR* restricted the evaluations to small scenarios only. The flow schedules were calculated using all the presented approaches in each of the scenarios. For the static scheduling approaches, all flows were collectively considered, while for the dynamic scheduling approaches, the flows were considered *one at a time* in a random order. As evaluation metric, the relative quality of the approach was determined with respect to *S/UR*, i.e., the ratio of number of flows scheduled by the approach to the number of flows scheduled using *S/UR*.

The cumulative distribution of the relative quality for the evaluated scenarios is presented in Figure 7.3a. As expected, the static scheduling approaches yield better relative quality compared to the dynamic scheduling approaches. Overall, the solutions generated by *S/FR* and *S/PR* have average relative qualities of 99 % and 97 % respectively, as against 89 % and 88 % for D/SAP and D/MM respectively. Further, *D/SAP* and *D/MM* result in solutions with relative qualities of at least 72 % and 64 % or better, respectively. On an average, the dynamic scheduling approaches produce solutions with a relative quality of 88 % compared to the *S/UR*. It is thus imperative to ensure that the optimizations to reduce the runtime of the dynamic scheduling approaches do not degrade their quality further.

It is also worth to note that the evaluations show *D/SAP* to be slightly better than *D/MM*. Of the 160 evaluation scenarios, *D/MM* outperformed *D/SAP* in 30 scenarios with respect to the number of scheduled flows, *D/SAP* was better than *D/MM* in 49 scenarios, while in 81 scenarios both of them scheduled an equal number of flows. The

is because *D/MM* occasionally routes flows over longer paths resulting in consumption of time-slots over a higher number of links.

7.5.2 Execution Time

The execution time of the scheduling approaches is mainly affected by the topology size and the number of available time-slots for scheduling. The evaluation results detailing the impacts of these parameters on the execution time, based on evaluations on a machine with Intel Xeon E5-1650 processor (2 × 6 cores) and 16 GB RAM, is presented in Figure 7.3.

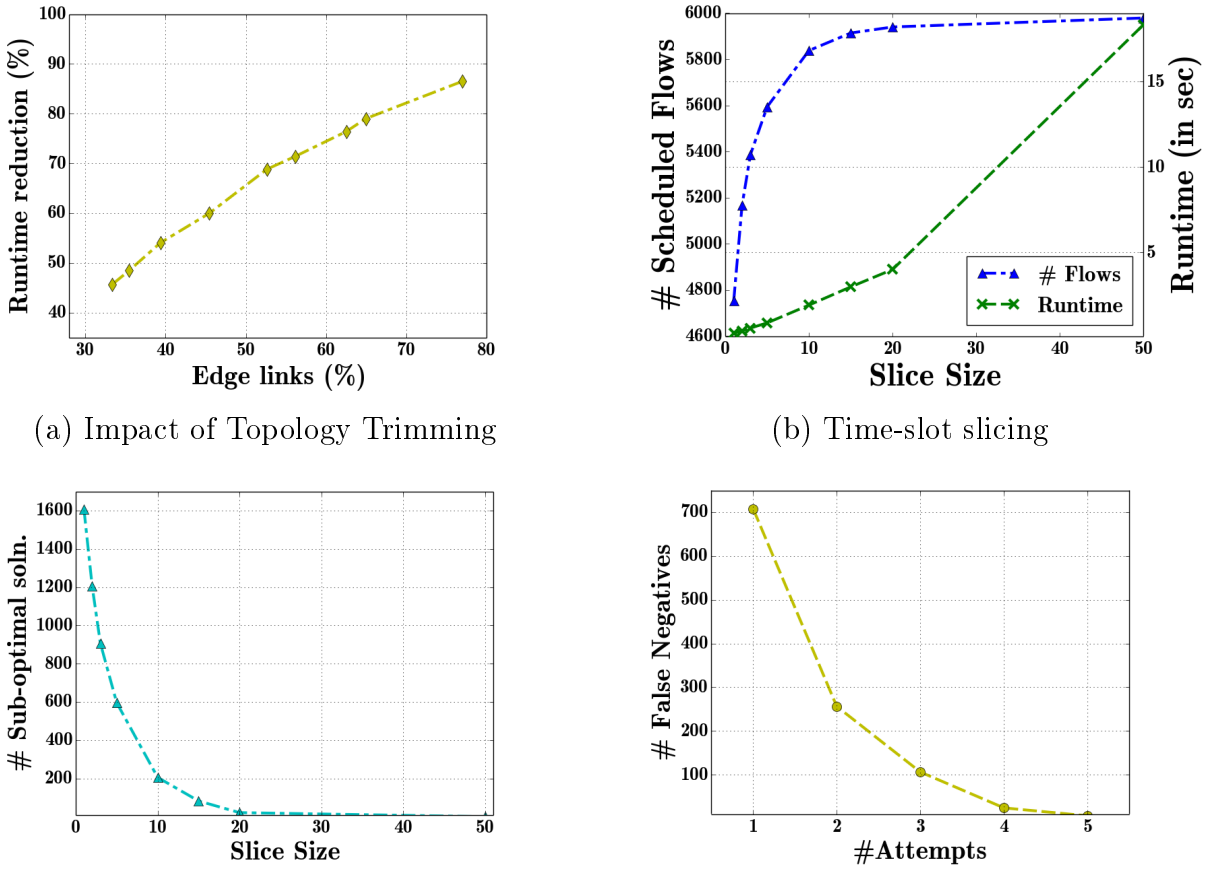
The impact of the number of available time-slots on the execution time (amortized over one flow) for all the scheduling approaches, dynamic as well as static, is shown in Figure 7.3b. The figure shows the time to schedule 300 flows with periods equal to the base-period on a Waxman topology (256 network links) for a varying number of time-slots (5–50). The results show that the runtime of *S/PR* increase rapidly with the number of available time-slots, requiring about 25 sec per flow with 50 time-slots. *S/FR* requires less than 4 sec per flow for scheduling with 50 time-slots. In comparison, the dynamic scheduling approaches without the proposed optimizations require over 80 sec to schedule a flow. However, with optimizations the approaches could determine the schedule in less than a second.

To evaluate the effect of the network size, we measured the runtime for scheduling over 100 flows on topologies of different sizes (30–256 network links). The results (Figure 7.3c) show a trend similar to the results with varying time-slots. The static scheduling approaches fare better than the dynamic scheduling approaches (without the proposed optimizations) with an amortized cost of 12 sec for *S/PR* to schedule a flow. The evaluations also show that the runtime of *S/PR* is dependent on the number of paths available for routing, i.e., the path diversity in the network. On the other hand, the runtime of dynamic scheduling approaches without optimization increases steeply with the size of the topology, with about 75 sec runtime to schedule one flow for a topology with 256 links. Clearly, the raw ILPs are not practical for online scheduling without further optimizations. The evaluations show that the execution times of *D/SAP* and *D/MM* are similar. Further, the execution time is independent of the type of the flow(s) being scheduled, unicast or multicast. In case of multicast flows, the number of destinations also do not have an impact on the execution time.

The dynamic scheduling approaches are also affected by the transmission period of the flow being scheduled. Due to use of phasing, flows with larger transmission periods have more options to be accommodated in the network schedule. Figure 7.3d shows the impact that this has on the runtime (along with their standard deviations) to schedule 100 flow incrementally on an ER topology (150 links). In absence of time-slot slicing, flows with larger transmission periods resulted in longer execution time with an

increasing standard deviation. With time-slot slicing, the transmission period ceases to have an effect on the execution time which is then achieved in sub-seconds.

7.5.3 Impact of Optimizations on Dynamic Scheduling



(a) Impact of Topology Trimming

(b) Time-slot slicing

(c) Slice size vs. Sub-optimality

(d) No. of attempts vs. False-negatives

Figure 7.4: Evaluations of the impact of the optimization

We presented two optimizations for reducing the execution time of the dynamic scheduling approaches. The first of them is topology pruning to remove unnecessary edge links. To evaluate its impact, the execution time for scheduling flows using *D/SAP* and *D/MM* on several topologies (RRG, ER, BA) was measured with and without topology pruning. As shown in Figure 7.4a, with an increasing proportion of edge links in the network, the reduction of runtime achievable also increases substantially. For instance, in a network with over 75 % edge links, topology pruning results in 86 % reduction of execution time for the scheduler, from 7.4 s to 1 s. However, in realistic production networks, the proportion of edge links is lower as multiple redundant links

are required between switches for improving fault tolerance of the network. For instance, in a fat-tree topology (found in data-centers), only 33 % of the total network links are edge links. Here, topology pruning provides about 45 % reduction in execution time.

While topology pruning is qualitatively non-destructive, time-slot slicing may result in sub-optimal schedules or generates false-negatives. It is, thus, not straightforward to evaluate the impact of time-slot slicing. For this, 10000 time-triggered flows were scheduled on an ER topology (500 hosts and 20 switches with 131 core links) over 50 time-slots. For each of these flows, the schedule was computed with and without time-slot slicing (using *D/SAP* and topology pruning) to evaluate its impact on optimality of the computed schedules and determine false-negatives. Figures 7.4b–7.4d summarize the results of this evaluation. Figure 7.4b shows that the execution time of the scheduler increases linearly with the size of the time-slot slice with runtimes lower than 1 s for slice sizes less than 5. However, the number of flows scheduled in the network do not commensurately increase with it. The evaluations also show that the achieved reduction in the number of generated false-negatives and sub-optimal solutions is not substantial with an increase in the size of time-slot slices beyond a certain degree (cf. Figures 7.4c and 7.4d). Overall, *with a slice size of just 3–5 time-slots, sub-second set-up time is achieved for scheduling flows.* Of the total scheduled flows, only 906 and 595 flows were sub-optimally routed with slice size of 3 and 5, respectively. Further, the number of generated false-negatives also reduced substantially by reattempting to schedule the flow using the next best time-slot slice, in this case, from 709 to 106 by allowing just two additional attempts to schedule the flow in case a suitable schedule could not be computed. After deploying the optimizations (topology pruning and time-slot slicing with a time-slice size of 5 with up to 3 scheduling reattempts), the overall number of flows scheduled in the network decreases by less than 4 % only.

7.5.4 Evaluation Summary

Overall, the evaluations show that

1. *S/UR* yields the best schedules in terms of the number of flows scheduled for TSSDN and serves as a benchmark for other heuristic solutions for static as well as dynamic scheduling problem.

2. The dynamic scheduling approaches, *D/SAP* and *D/MM*, with optimizations can compute online schedules for time-triggered flows in sub-seconds.

7.6 Related Work

The development of software-defined networking was mainly driven by the need of managing dynamic communication networks. While the control plane can modify the

data plane in a live network through OpenFlow, using these capabilities naively may potentially lead to inconsistent network states and problems like blackholes, loops, etc., in the network. Several approaches were proposed to solve the problem of inconsistent network updates. Reitblatt et al. proposed usage of VLAN tags to version flow table rules to provide per-packet or per-flow consistency [43]. Another approach was to route traffic over the control plane during the transition period [111]. Jin et al. designed Dionysus, a framework for dynamically scheduling network updates after resolving conflicts to enable faster completion of the updates [112]. We have used the transactional semantics in OpenFlow for avoiding inconsistent network updates. Interestingly, while the problem of updating routes in the network dynamically is well addressed, dynamic updation of schedules for time-sensitive traffic is still not completely solved.

There are a few SDN-based approaches which do consider the timing/bandwidth requirements while admitting them into the network. Danielis et al. present an SDN-based approach to migrate routes of flows to avoid congestion based on the results of a polynomial time algorithm [113]. Hedera, data center networks with multi rooted tree topologies, also use SDN to modify paths of data center flows based on fast heuristics while delivering bisection bandwidth of the network [114]. Greff et al. present an online flow admission control and routing algorithms based on SDN where the switch ports are based on credit-based weighted round robin scheme [115].

In contrast to these approaches, in TSSDN, we developed online algorithms for time-triggered flows along with mechanisms for dynamically adding/removing flows.

7.7 Summary

This chapter presented mechanisms for inserting and removing time-triggered flows in TSSDN, an SDN-based architecture, without affecting already scheduled flows. We also formulated the dynamic scheduling problem in TSSDN and presented algorithms using integer linear programming for computing schedules and routes for flows incrementally. Overall, TSSDN achieves sub-second set-up times for scheduling arbitrary time-triggered flows dynamically on account of the fast online scheduling algorithms and lightweight mechanisms for updating schedules.

CHAPTER 8

SPATIAL MULTIPLEXING IN UNSYNCHRONIZED NETWORKS

8.1 Introduction

In this chapter, we look at networks where neither the hosts are synchronized nor are the switches equipped with specialized hardware enhancements for scheduling transmissions. Such cases can arise in legacy networks with end systems equipped with network interfaces which are not capable of time-stamping packets for the purpose of clock synchronization. In such networks, scheduling of time-triggered flows cannot be enforced by any network participant, and hence, spatial isolation by routing remains the only means for isolating the time-triggered traffic from other traffic classes.

It may be recollected that to achieve deterministic and bounded end-to-end latency and jitter for time-triggered traffic, we seek to eliminate the in-network queuing delays that the packet belonging to these flows incur. To this end, in the networks where transmission scheduling is not a feasible option, network links can be exclusively allocated to the time-triggered flows by routing them over *edge disjoint paths*. These flows would naturally share the allocated links with best-effort traffic but with no other time-triggered flows. Furthermore, we use in-network prioritization to separate the time-triggered traffic in the network links from best-effort traffic, i.e., packets of the time-triggered flows can be tagged as high priority traffic, while all other packets have lower priorities. This ensures that the forwarding of high-priority time-triggered traffic is always expedited in the network. The best-effort traffic in transit may delay the time-triggered flows, however, like in TSSDN (cf. Chapter 6) this delay is restricted to the time-required to transmit an MTU-sized packet over the link. This delay can be further reduced by means of frame pre-emption mechanisms specified in the IEEE 802.1Qbu standard [45].

While we can use the routing capabilities available using OpenFlow for spatially isolating time-triggered flows, the question remains on how to compute routes for time-triggered flows. Ideally, we would like to maximize the number of time-triggered flows accommodated in the network. Given a set of source destination pairs in the network, the problem of maximizing the number of flows which can be accommodated in the network while traversing over edge disjoint paths is proven to be an NP-hard problem [116]. Heuristics to solve this problem have already been presented in [117]. However, we propose routing of time-triggered flows not with the aim of maximising the number of time-triggered flows in the production network, but rather to maximize the number of cyber-physical systems (CPS) which rely on these time-triggered flows. Generally, CPS require varying numbers of time-triggered flows between its components (sensors, actuators, controller etc.) based on the type of production processes it implements and the number of components it has on the shop floor. For instance, CPS responsible for open-loop production processes typically require lower number of time-triggered flows compared to the ones responsible for closed-loop production processes. The number of time-triggered flows that can be accommodated in the network by routing over edge disjoint paths is very limited. Hence, we strive to maximize the number of cyber-physical systems supported in the network, as it also implies a higher number of manufacturing applications in the shopfloor.

This chapter is structured as follows. We present the system model and problem statement for routing of time-triggered flows in an unsynchronized network in Section 8.2. Our solutions to this routing problem is presented in Section 8.3. We present the evaluations and a brief discussion on our approach in Section 8.4 and 8.5, respectively, before concluding.

8.2 System Model & Problem Statement

8.2.1 System Model

We assume a system model similar to the one of TSSDN, presented in Chapter 6.2, with a centralized control plane configuring the network data plane for the time-triggered flows.

The major difference to TSSDN is that we relax the requirement for having synchronized clocks at the end systems. This also implies that control plane computes only routes and not schedules for the time-triggered flows. Based on the computed routes, the control plane configures the data plane using the OpenFlow protocol. Furthermore, the lack of scheduling also implies that no issues arise with respect to schedule adherence from the point of view of the end systems. However, to avoid jitter in the network stack of the end systems, we still use userspace packet processing framework for deterministic latency in the network stack like in Chapter 6.

8.2.2 Problem Statement

Our routing approach allocates edge disjoint paths to the time-triggered flows while seeking to maximise the number of CPS so realised. Edge-disjoint paths avoid the problem of competing flows along any path, thus, eliminating the problem of network congestion and high queueing delay. Note that we do not strive to allocate several flows to the same link. Although this could further increase throughput and number of supported CPS, it requires more complex solutions including scheduling mechanisms at switches, presented in Chapters 4 and 6.

Merely maximising the number of realised CPS raises fairness issues for those requiring higher number of flows. We can mitigate this issue by allocating weights to each CPS based on their importance and then maximise the sum of weights for the set of realised CPS. While our algorithms can be easily extended for this, it is currently out of the scope for this chapter.

8.3 Routing Time-triggered Communication Flows

In this section, we present our solutions for routing of time-triggered communication flows constituting CPS in a manufacturing shop-floor over unsynchronized networks. First, we introduce a basic algorithm for computing the routes for the time-triggered flows, and then present a pair of heuristics to guide this basic algorithm towards improved solutions.

8.3.1 Terminology

In our system model, we address the underlying network for the CPS along with the time-triggered communication flows on which they rely. The network is modelled as a directed graph, $G \equiv (V, E)$, where V is the set of network nodes and E is the set of edges representing the network links connecting the nodes. Further, $V \equiv S \cup H$, where S is the set of network switches and H is the set of hosts in the network. Also, $E \subseteq V \times V$ such that if $(v_1, v_2) \in E$, then $(v_2, v_1) \in E$. This models the full-duplex nature of switched Ethernet networks.

$CPSSet$ represents the set of target CPS required for implementing the production processes in the shop floor. For $cps \in CPSSet$, $cps = \{(h_i, h_j) \mid h_i, h_j \in H\}$, i.e., a cyber-physical system is described as a set of tuples, each containing a pair of hosts — the source host and the destination host for a time-triggered flow.

We refer to a CPS as "realised" if the used routing algorithm assigns network paths for all its time-triggered flows. We model the solution to this routing problem of time-triggered flows as a tuple — ($CPSSeq$, $FlowSeqMap$). $CPSSeq$ is a sequence of the target CPS, $CPSSeq \equiv [cps_1, cps_2, \ldots cps_n]$. While, $FlowSeqMap$ is a map that gives the sequence of the flows corresponding to a CPS. i.e., $FlowSeqMap[cps_i] \equiv$

$[f_{i,1}, f_{i,2}, \ldots f_{i,j}]$, where $f_{i,j}$ is the j^{th} flow of system cps_i. Further, we define the fitness (quality) of a given solution as the number of CPS that are realised by the basic routing algorithm described below.

8.3.2 Calculating Edge Disjoint Routes for Time-triggered Flows

Given a graph G along with a set $T \equiv \{(s_1, d_1), (s_2, d_2), \ldots (s_n, d_n)\}$ containing pairs of hosts in the networks, the problem of maximising the number of pairs to which edge disjoint paths can be allocated is known as the maximum edge disjoint paths (MEDP) problem. This well-known problem is NP-Hard and requires heuristic approaches for finding solutions that are close to the optimal ones.

Our problem formulation takes the maximum edge disjoint paths problem a step further. Instead of maximising the number of flows for which edge disjoint paths can be allocated, we seek to maximise the number of CPS that are realised as a result. Our problem however reduces to the MEDP problem, if all the CPS in $CPSSet$ consist of only one time-triggered flow.

Algorithm 6 Basic algorithm to route time-triggered flows

Require: Graph G, Solution ($CPSSeq$, $FlowSeqMap$)

```
 1: Routes ← { }
 2: for cps in CPSSeq do
 3:     for flow in FlowSeqMap[cps] do
 4:         flowPath ← Dijkstra(G, flow.src, flow.dst)
 5:         if flowPath != NULL then
 6:             Routes[flow] ← flowPath
 7:             G.edges ← G.edges − flowPath
 8:         else
 9:             deallocate all the assigned flows of cps
10:             add corresponding links back to G
11:         end if
12:     end for
13: end for
14: return Routes
```

Algorithm 6 describes the basic routing algorithm to allocate edge disjoint paths to the time-triggered flows based on an input solution described by the tuple ($CPSSeq$, $FlowSeqMap$).

The algorithm realises the CPS as per the order dictated by the $CPSSeq$ (Line 2). For realising a CPS, the algorithm calculates and allocates routes to all its flows in the order defined by the map $FlowSeqMap$ indexed by the corresponding CPS (Line 3). The routes for the flows are determined by executing Dijkstra's algorithm (described in [79]) on graph G (Line 4). To ensure that a network link is not allocated to multiple flows, we remove the corresponding edge from the graph G when it is allocated to a flow (Line 7). If no route can be found for some flow, then the corresponding CPS cannot be realised. It is then futile to allocate the expensive network resources for its other flows for which perhaps routes were already computed. Hence, we deallocate such flows and return the respective network links to the graph G (Line 9–10).

The fitness of a solution with this basic algorithm depends on the ordering of the CPS in $CPSSeq$ and the corresponding ordering of flows in $FlowSeqMap$. In the following, we improve the basic algorithm to derive a greedy algorithm (heuristic approach) and a genetic algorithm (meta-heuristic approach) that generate better solutions, i.e., increase the number of realised CPS.

Greedy Algorithm

In the greedy algorithm, we generate a few candidate solutions heuristically and then select the best one from them. This algorithm takes four inputs: i) G, the network graph, ii) $CPSSet$, the set of target CPS, iii) $MaxSolns$, the number of candidate solutions to be considered, iv) $FlowSeqMap$, a map that gives the initial ordering of the flows corresponding to a CPS.

In the greedy algorithm, described in Algorithm 7, we use the number of flows required to realise a CPS as the heuristic. We create different sequences of target CPS in which they are sorted in an ascending order of the number of constituent flows, i.e., $CPSSeq_i$, where $i \equiv \{1, 2, \ldots MaxSolns\}$. Each of these sequences is combined with the default ordering of flows given by $FlowSeqMap$ to generate $MaxSolns$ candidate solutions, i.e., ($CPSSeq_i$, $FlowSeqMap$). To allow these candidate solutions to vary significantly, we generated $CPSSeq_i$ by randomly shuffling the set of target CPS, $CPSSet$, and then executing a sorting process (Line 4). Since we used a stable sort, the relative order of the CPS before sort is maintained after sort as well. We select the best of the candidate solutions after evaluating their individual fitness (Line 7–13). Function calculate_fitness() (Lines 8) (essentially similar to the Algorithm 6) tracks and returns the number of CPS realised by the solution using the basic routing algorithm.

The greedy algorithm uses the default ordering of flows, as given in the $FlowSeqMap$, due to lack of reliable heuristics for ordering the flows of a particular CPS for routing. To explore the solution space effectively, we developed a meta-heuristic approach using a genetic algorithm that also varies the ordering of flows in $FlowSeqMap$.

Algorithm 7 Greedy algorithm for maximising the number of realised CPS

Require: Graph G, Set of CPS $CPSSet$, Flow Sequence Map $FlowSeqMap$, No. of candidate solns $MaxSolns$,

```
 1: CandidateSolutions ← [ ]
 2: BestSolution ← NULL; BestSolnFitness ← 0
 3: for i = 0 to MaxSolns do
 4:     CPSSeq ← stable_sort(shuffle(CPSSet))
 5:     CandidateSolutions.add((CPSSeq, FlowSeqMap))
 6: end for
 7: for Soln in CandidateSolutions do
 8:     SolnFitness ← calculate_fitness(G, soln)
 9:     if SolnFitness > BestSolnFitness then
10:         BestSolnFitness ← SolnFitness
11:         BestSolution ← Soln
12:     end if
13: end for
14: return BestSolution
```

Genetic Algorithm

Typically, the first step in genetic algorithms is to generate a set of candidate solutions, known as the population. The quality (fitness) of the population is then iteratively improved by execution of three genetic operators — Selection, Cross-over and Mutation. The genetic algorithm described in Algorithm 8 needs an additional input as compared to the greedy algorithm: $MaxIterations$, the number of iterations/generations for which the candidate solutions are allowed to evolve.

We generated the initial population using the method similar to the one used in the Greedy Algorithm, i.e., using stable sort after random shuffling of $CPSSet$ (Line 3). Additionally, we randomly shuffled the ordering of flows in $FlowSeqMap$ corresponding to every CPS (Lines 4–6) for each candidate solution. Further, we improved the calculation of solution fitness to break ties between candidates that realise an equal number of CPS. In such cases, the fitness calculation procedure takes into account the number of flows set up for breaking the tie. The higher the number of flows set up, the better the solution.

With each iteration, the population evolves with improving quality of solutions. In each iteration, three genetic operators (Lines 9–13) are applied on the population:

1. *Selection* is responsible for selecting the candidate solutions from the current generation that contribute towards the candidate solutions for the next generation. We used roulette wheel selection mechanism for this operator [118]. With this

Algorithm 8 Genetic algorithm for maximising the number of realised CPS

Require: Graph G, Set of CPS $CPSSet$, Flow Sequence Map $FlowSeqMap$, No. of candidate solns $MaxSolns$, No. of iterations $MaxIterations$,

```
1: Population ← [ ]
2: for i = 0 to MaxSolns do
3:     CPSSeq ← stable_sort(shuffle(CPSSet))
4:     for cs in CPSSet do
5:         FlowSeqMap[cs] = shuffle(FlowSeqMap[cs])
6:     end for
7:     Population.add((CPSSeq, FlowSeqMap))
8: end for
9: for i = 0 to MaxIterations do
10:    SelectPopulation ← selection(G, Population)
11:    NextGeneration ← crossover(SelectPopulation)
12:    Population ← mutate(NextGenPopulation)
13: end for
14: return fittest(Population)
```

method, candidate solutions with higher fitness have a higher probability of making the cut while solutions with lower fitness head for extinction. This operator, hence, needs graph G as input for calculating the fitness of the population.

2. *Cross-over* combines two "selected" candidate solutions of the current generation to generate two solutions for the next generation. This operation must be carefully designed to ensure that the candidate solutions for the next generation have a high probability of improving over their parents from the current generation. We used a uniform cross-over method for this operation [118]. This implies that two parent solutions are combined to construct two child solutions, such that the children get approximately half of the solution ($CPSSeq$ as well as $FlowSeqMap$) from each parent.

   Further, we also used elitist selection method that allows the fittest solution to walk into the next generation unaltered.

3. *Mutation* alters candidate solutions slightly to maintain diverse solutions for a wider exploration of the solution space. Typically, the mutation operator is applied with a very low probability to avoid randomizing the candidate solutions. While applying this operation, we mutated randomly selected candidate solutions by swapping a randomly selected pair of systems in its $CPSSeq$.

At the end of the specified iterations, we choose the fittest solution present in the population (Line 14).

8.3.3 Extensions for Dynamic Routing

Our approaches can be easily extended to compute routes for time-triggered flows of new CPS added into the network. It may be noted that the routes of the time-triggered flows depends purely on the sequence in which the CPS' are taken up for scheduling, $CPSSeq$, and the sequence of the flows corresponding to a CPS, $FlowSeqMap$. Thus, on adding a new CPS in to the network, we append the CPS to the existing sequence, $CPSSeq$. We then execute the basic routing algorithm, presented in Algorithm 6, for the corresponding CPS only with the unallocated links in the network. Like in the Greedy Algorithm, we use a random ordering of the flows for the new CPS due to lack of heuristics for ordering the flows. In an event of adding several CPS in the network, the Genetic Algorithm can be deployed for the new systems only by including only the allocated links in the network graph. Overall, this ensures that the routes of the existing flows are not altered due to the addition of new CPS.

For removing any CPS from the network, the links allocated to the constituent time-triggered flows can be deallocated and re-added to the topology graph. These links can then be for the routing of the future flows.

8.4 Evaluations

The algorithms presented in this chapter route the time-triggered communication flows over edge disjoint paths. The resulting end-to-end delay is then the sum of propagation delay of the network links and the processing delay of the network switches over which the flow is routed. The queuing delay is eliminated as no two flows contend for access to any network link. The evaluation results in [28] found the processing delay of a state-of-the-art SDN switch to be constant (around 3.8μs). Further, the propagation delay in the corresponding 1 Gbps link was estimated to be constant (around 5–15ns) depending on its length. Thus, using edge disjoint paths for routing results in minimal end-to-end delays (in the microseconds range) for the time-triggered flows.

In the following, we also evaluated our algorithms in terms of the quality of the solution that they generate, i.e., the number of CPS that they realise, and their runtimes. For this, we simulated them to calculate the edge disjoint network paths for time-triggered flows of a varying set of CPS over random network topologies generated using Erdős-Rényi model [60]. We used NetworkX to generate these random topologies for simulating the evaluation scenarios [62].

The algorithms were primarily evaluated in two phases. In the first phase, we compared the quality of solution with the optimal solution for a small topology. Note that due to the high complexity of the problem, such a comparison with the optimum is only possible for smaller scenarios. The goal here was to gauge if the solutions generated by our algorithms are close enough to the optimal solutions. In the second phase, we compared the performance of the algorithms with each other when executed on larger

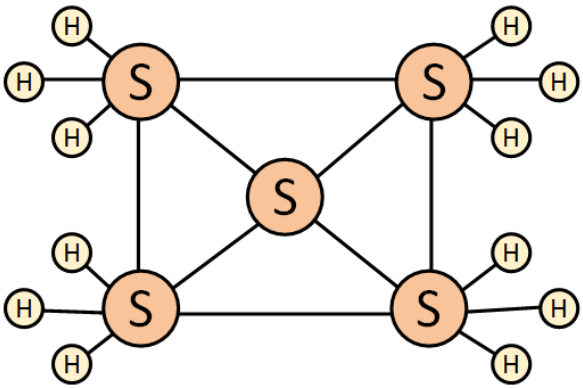


Figure 8.1: Small topology for benchmarking. S indicates the network switches while H indicates the end-hosts.

problem sizes. Here, we evaluated the runtime of the algorithms along with the quality of solutions generated in each of the evaluation scenario.

8.4.1 Performance Comparison with Optimum for Small Problem Sizes

To determine if the solutions generated by our algorithms are close enough to the optimal solution, we created a small benchmark topology with high path diversity consisting of 5 network switches and 12 hosts, as shown in Figure 8.1. Further, we created a set of 20 CPS, each requiring between 1 to 5 time-triggered flows. By exhaustively searching the solution space, we determined that the optimal solution for this problem realises 7 CPS consisting of 12 flows in total.

We executed the greedy and the genetic algorithms on the benchmark topology with inputs to consider 6 candidate solutions. We allowed the genetic algorithm to perform 4 iterations to improve the candidate solutions. The results of 100 execution runs of these algorithms are summarised in Table 8.1. For these executions, the solutions generated by the genetic and the greedy algorithm could, on an average, realise 6.42 and 6.16 CPS respectively compared to the 7 CPS realisable using the optimal solution. Moreover, the genetic algorithm could produce the optimal solution 42 times out of the 100 execution runs compared to the 16 times by the greedy algorithm.

Thus, for smaller topologies, the solutions generated by our algorithms are quite close to the optimal ones. Further, they are able to generate the optimal solutions frequently in case they are executed multiple times.

8.4.2 Comparison of algorithms

In the second phase of evaluations, we compared the performance of our algorithms with each other. For this, we created random graphs using the Erdős-Rényi model,

Algorithms	Category	Mean	Std. Dev.	Optimal Solution
Genetic Algorithm	Systems Realised	6.42	0.49	42 times
	Flows	9.95	1.81	42 times
Greedy Algorithm	Systems Realised	6.16	0.37	16 times
	Flows	8.76	1.61	16 times

Table 8.1: Results of 100 execution runs of greedy and genetic algorithm on the benchmark topology shown in Figure 8.1.

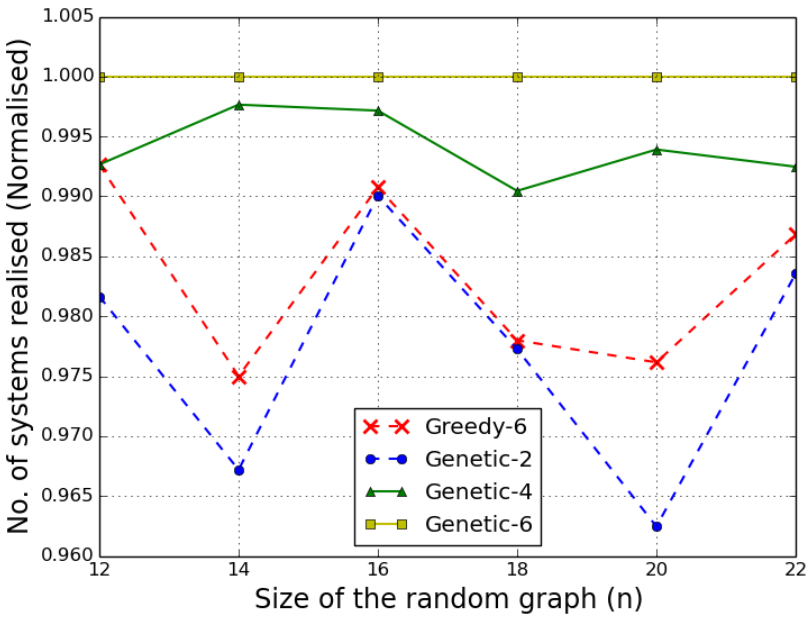


Figure 8.2: Quality of solutions produced (Average of 100 execution runs)

n	Genetic Algorithm			Greedy Algorithm		
	Sys. Realised	Flows	Runtime	Sys. Realised	Flow	Runtime
30	29.72	54.2	105ms	29.04	52.25	18ms
40	39.36	70.45	138ms	38.87	63.48	24.3ms
50	49.43	91.49	188ms	48.8	90.07	34.2ms

Table 8.2: Average results of 100 execution runs of greedy and genetic algorithm on random topologies generated using Erdős-Rényi model ($p = 0.25$ and varying n).

denoted as $G(n, p)$ [60]. These graphs consist of n nodes with p denoting the probability that any two nodes are connected by an edge. We also created a set of target CPS containing $2n$ systems, each consisting of between 1 to 3 time-triggered flows uniformly distributed over the network. We executed our algorithms on these randomly generated graphs and compared their performance with respect to the quality of the solutions they generated and their corresponding runtimes.

To ensure that none of these algorithms gain an undue advantage, we executed both the algorithms with the input to consider only 6 candidate solutions. The genetic algorithm was executed to perform 4 iterations.

The Table 8.2 summarises the results of 100 execution runs of the algorithms on random topologies generated using the Erdős-Rényi model with varying n. These results show that when both the algorithms consider an equal number of candidate solutions, the average solution provided by the genetic algorithm is better than the greedy algorithm although its runtime is an order of magnitude higher than that of the greedy algorithm.

Finally, we also executed the algorithms to evaluate if the genetic algorithm can outperform the greedy algorithm despite considering a lower number of candidate solutions, thereby decreasing the penalty in runtime for the genetic algorithm. For this purpose, we created random topologies, $G(n = 12 \text{ to } 22, p = 0.25)$, again using the Erdős-Rényi model. Similar to the preceding evaluation scenario, we created a random set of target CPS containing $2n$ systems for each of the corresponding topologies. The results of executing the algorithms with different number of candidate solutions (average of 100 execution runs) is summarised in Figures 8.2–8.3. For this evaluation, we executed the greedy algorithm with 6 candidate solutions while the genetic algorithm was executed with 2, 4, and 6 candidate solutions. An obvious result of this experimentation was that the quality of solutions generated (Figure 8.2) and the algorithm runtime (Figure 8.3) increases with an increase in number of candidate solutions considered irrespective of the topology. Figure 8.2 further shows slight fluctuations in the quality of the solutions produced by the algorithms on various topologies. We attribute these fluctuations to the randomness of generating initial candidate solutions in both of these algorithms.

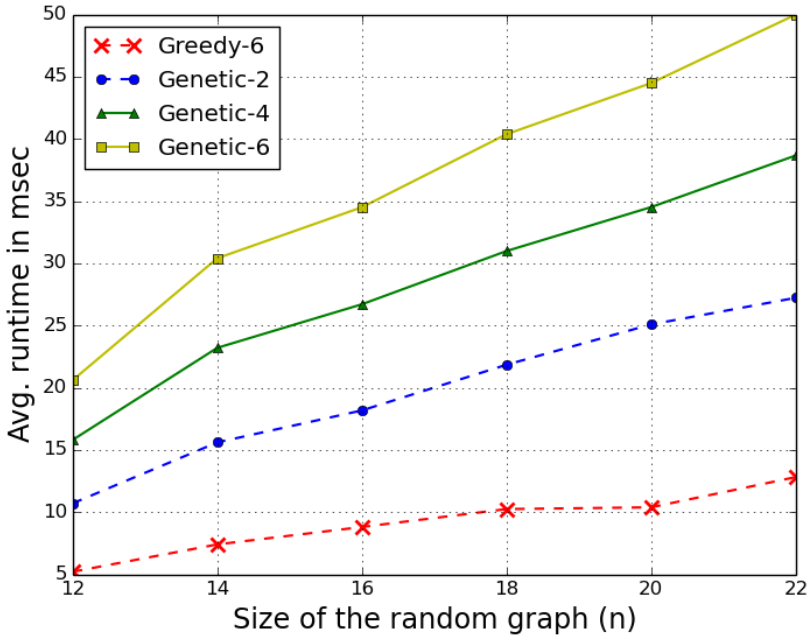


Figure 8.3: Runtime of the algorithms (Average of 100 execution runs)

Despite these fluctuations, we can infer that the genetic algorithms produce better quality results as compared to the greedy algorithm despite using far lower number of candidate solutions for all the topologies that were considered. For instance, the genetic algorithm considering 4 candidate solutions provided better solutions than the greedy algorithm considering 6 candidate solutions.

To summarise, the genetic algorithm can be fine-tuned using its input parameters like the number of candidate solutions it considers and the number of iterations it undergoes. The fine-tuning involves a trade-off between the quality of the solution it generates and the algorithm runtime.

8.5 Related Work & Discussion

The idea of routing communication flows over edge disjoint paths in the network has been used in the networking community for a variety of purposes. Among others, such routing schemes are used for improving the effective bandwidth utilization in the network by routing the flows between the source and destination over multiple disjoint paths [119] [120] [114]. Furthermore, these routing schemes are also used for improving fault tolerance of the networks for the network applications [121]. By routing communication flows over k disjoint paths, the applications relying on the flow are tolerant against up to k link failures, one over each of the path.

Spatial multiplexing, in particular isolating the traffic by means of the queues at the output port of a switch, has been also used as a means to provide real-time guarantees

for communication. Kumar et al. allocate one queue for each real-time flow at all the output ports in a bid to achieve deterministic latencies [92]. On the other hand, Grosvenor et al. present network architecture which provides different trade-offs between deterministic latencies and throughput by separating traffic into different queues each guaranteeing specific bounds on incurred latency [93]. In contrast, our routing algorithms do not strive to exploit the availability of the queues at the output port by allocating a queue for each of the time-triggered flow. This is mainly because the time-triggered flows in the lower priority queues would be affected by the flows in the higher priority queues.

Our routing algorithms, which allocate edge disjoint paths to the time-sensitive communication flows in manufacturing systems, perform well only with network topologies with high path diversity, e.g., in multi-rooted trees. Without high path diversity, it would be difficult to allocate edge disjoint network paths to most time-sensitive flows. Given the concerns relating to the network utilization on account of scheduled traffic, one might argue against allocating network links exclusively to communication flows. While data-center topologies do assign "non-conflicting paths" for some of their flows, they do so only for ensuring that bandwidth constraints on all network links are respected [114]. However, the QoS levels, in particular the communication latency and jitter, desired by time-triggered flows of manufacturing system are a notch higher than the soft real-time communication flows in data-centers. Given that the time-triggered flows are negatively affected when network resources (like switch buffers, network links, etc.) are shared across them, allocating edge disjoint network paths to these time-triggered flows remains the only available option in absence of any level of synchronization in the network.

With our routing approach, the network utilization on account of time-triggered traffic is extremely low in absence of best-effort traffic in the network. Furthermore, we also agree that the approach of assigning network links exclusively (shared with low-priority best-effort traffic) for one time-triggered flow is not scalable to systems consisting of thousands of time-triggered flows. For such large systems, additional primitives are required from the network enabling the time-triggered flows to traverse over the links in a pre-defined time-multiplexed manner, i.e., the flows are assigned time-slots during which they have exclusive access to all the resources in a network path, but overall the network links are used by more than one flow. We have already discussed such network architectures in Chapters 4 and 6. Nonetheless, the algorithms presented in this chapter can be used in unsynchronized partitions of a large network or for tunnelling time-triggered traffic between two synchronized sub-networks over an unsynchronized partition.

8.6 Summary

In this chapter, we discussed the handling of time-triggered flows over unsynchronized networks by spatial multiplexing for providing them with necessary latency guarantees. To this end, we introduced the corresponding routing problem for these flows and proposed a set of algorithms for efficiently solving it by exploiting the logical centralization of the control plane available in SDN. Our solutions compute routes for time-triggered flows while striving to maximize the number of realised cyber-physical systems. While unsynchronized networks are seldom used in production networks due to the issues with the network utilization, the presented algorithms can be used in unsynchronized partitions of a large network or for tunnelling time-triggered traffic between two different partitions of the network, each of which has one or more network participants capable of scheduling time-triggered transmissions.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this chapter, we briefly review the contributions of this thesis and provide an overview on potential future work based on our results.

9.1 Summary of Contributions

Converged communication infrastructure which can provide different levels of service guarantees for the network applications based on their requirements is the need of the hour for the manufacturing networks of smart factories. The information and communication technology (ICT) infrastructures in these smart factories are expected to host wide variety of applications ranging from highly time-sensitive cyber-physical systems to soft real-time complex event processing (CEP) applications. Several standards organisations are hence working towards standardizing real-time extensions for Ethernet, primarily designed for providing best-effort service, to enable its usage as a converged communication network.

The handling of time-triggered traffic stemming from hard real-time cyber-physical systems on the manufacturing shopfloor in Ethernet networks is, in particular, challenging. Typically, such time-triggered traffic demands strict bounds (to the order of a few microseconds) on the end-to-end latency and jitter. To provide such stringent guarantees, the transmissions of time-triggered traffic are scheduled through the network based on a pre-computed schedule inline with the timing constraints of the corresponding data streams. The IEEE Time-sensitive Networking (TSN) Task Group (TG) has even specified a time-aware programmable gating mechanism for enforcing these transmission schedules by effectively isolating the time-triggered traffic from other kinds of traffic. However, the computation of these transmission schedules to effectively utilize the network while also meeting the timing constraints of the time-triggered data

streams is no trivial task. Moreover, the computation of these schedules are heavily dependent on the routes of the corresponding data streams, making routing an important consideration for the scheduling algorithms.

Our scientific contributions in this thesis mainly deal with the computation of transmission schedules and routes for time-triggered traffic in Ethernet networks. We divide the solution space for the scheduling problem of time-triggered traffic in Ethernet networks based on where the computed schedules are enforced, i.e., on the switches as well as hosts, only on the switches, only on the hosts, and neither on the hosts nor switches. Each of these cases provide a trade-off between the network utilization with respect to scheduled traffic, bounds on the latency and jitter, and the costs incurred on account of the need for specialized hardware. In this thesis, we present scheduling and routing algorithms for computing transmission schedules and routes for each of the aforementioned cases. In particular, the results of this thesis are:

1. We formulated the scheduling problem of time-triggered data streams in networks equipped with the programmable gating mechanism (IEEE 802.1Qbv standard) as a No-wait Packet Scheduling Problem (NW-PSP) and mapped it to the No-wait Job-shop Scheduling Problem (NW-JSP). We also introduced the Integer Linear Programming (ILP) formulations for computing an exact solution along with a meta-heuristic approach based on Tabu search for reducing the runtime. The Tabu search approach scales to compute transmission schedules for up to 1500 data streams in less than 3 hours. Furthermore, we presented a schedule compression algorithm which results in up to 42 % reduction in the number of guard bands required for isolating time-triggered traffic from best-effort traffic.

2. To benefit from our scheduling solutions to the NW-PSP, we also introduced the metric Maximum Scheduled Traffic Load (MSTL) to measure the distribution of scheduled traffic in the network. Furthermore, we introduced routing algorithms for time-triggered data streams based on the MSTL to improve the schedulability of time-triggered data streams by evenly distributing the traffic in the network. Our evaluations show that these routing algorithms improve the schedulability of the time-triggered traffic by up to 30 % relative to Equal Cost Multipathing (ECMP) and by up to 60 % relative to shortest path routing.

3. We introduced Time-sensitive Software-defined Network (TSSDN), an SDN based architecture, for handling time-triggered traffic using commodity SDN switches, i.e., the transmission schedules for the time-triggered data streams cannot be enforced on the switches. Furthermore, we presented solutions—S/UR, S/PR, and S/FR—of varying time-complexities for computing schedules and routes for time-triggered data streams in TSSDN for static scenarios. The S/UR computes optimal solutions, and hence, has high runtime, while the S/PR and S/FR approximate the optimal solutions by restricting longer paths for the data streams, thus reducing the runtime.

We also introduced solutions—D/SAP and D/MM—for incrementally scheduling data streams in TSSDN without affecting the already scheduled data streams in the network. With additional optimizations, near-optimal schedules can be incrementally calculated for new time-triggered data streams in under a second.

4. We also introduced the routing problem for time-triggered data streams in networks where neither the hosts nor the switches are synchronized. Here, we maximize the number of cyber-physical systems that can be supported in the network by routing their constituent real-time data streams over disjoint paths, i.e., we spatially isolate real-time data streams. Our solutions—greedy approach and genetic algorithm—closely approximate the optimal solution.

5. Finally, we show by means of a proof-of-concept implementation that the computed schedules can be precisely adhered with using userspace packet processing frameworks like Intel's Data Plane Development Kit (DPDK).

Overall, the contributions of this thesis cover the entire spectrum of the solution space providing scheduling and routing solutions for time-triggered data streams in Ethernet network.

9.2 Future Work

The research results presented in this thesis can be potentially extended in several directions. A short list of potential future work based on our research results is enumerated as follows.

1. In this thesis, we have assumed a rather homogeneous system model, i.e., we assume that all network participants are similar to each other in terms of their scheduling capabilities. An interesting extension of the research presented in this thesis would be to combine the different scheduling and routing approaches as building blocks to develop algorithms for a hybrid system, e.g., a network where not all switches are equipped with IEEE 802.1Qbv extensions and only a few hosts have synchronized clocks.

   One possible approach for this would be to use TSSDN to tunnel time-triggered traffic between switches capable of scheduling transmissions.

2. Another direction to extend the scheduling and routing algorithms is to incorporate redundancy while computing routes to improve the robustness and fault tolerance in the network. The IEEE 802.1CB already provides mechanisms for frame replication in the network in order to forward traffic over redundant paths along with mechanisms for detection and elimination of duplicate packets resulting from such forwarding.

3. We have assumed complete centralization of the network control plane in our system model for simplifying the implementation of the scheduling and routing algorithms. To tackle the resulting scalability issues, one could think of distributing the control plane to multiple controllers which can execute the scheduling and routing algorithms in parallel.

   We could develop hierarchical approaches in which each controller is responsible for scheduling and routing flows in a part of the network. The schedule and routes of flows spanning over multiple network partitions is then composed by combining the results from multiple controllers. Alternatively, controllers can be responsible for a part of the schedule, enabling them to execute the algorithms for multiple flows in parallel without any conflicts. Such ideas have been successfully applied in literature to solve multiple performance issues in various systems [122] [123] [124]. A preliminary approach to compute segmented schedules (based on the periods of the flows) and then combine them to form the global schedule is presented in [125].

4. A logical extension to this work is to evaluate the impact of the scheduled traffic on the latencies/throughput of the other traffic classes in each of the cases we presented. One approach for this could be the use of network calculus for timing analysis like in [74] [126] [127] [128].

5. So far we have focussed on only computing schedules and routes for time-triggered data streams modelled as unicast and multicast flows. Communication paradigms like the publish/subscribe [129] [130] [131] can be implemented on the top of time-sensitive networks. The challenge here would be to map the advertisements and subscriptions of the publish/subscribe middleware to corresponding schedules and routes in the network. OPC UA, a widely used industrial communication middleware to semantically describe data streams, would benefit significantly from such a real-time publish/subscribe communication network.

BIBLIOGRAPHY

[1] F. K. Pil and M. Holweg, "Linking product variety to order-fulfillment strategies," *Interfaces*, vol. 34, no. 5, pp. 394–403, 2004.

[2] G. Volpato and A. Stocchetti, "Managing product life cycle in the auto industry: evaluating carmakers effectiveness," *International Journal of Automotive Technology and Management*, vol. 8, no. 1, pp. 22–41, 2008.

[3] P. Riffelmacher, S. Kluge, R. Kreuzhage, V. Hummel, and E. Westkamper, "Learning factory for the manufacturing industry: Digital learning shell and a physical model factory - ITRAME for production engineering and improvement," pp. 120–131, 01 2007.

[4] D. Zuehlke, "SmartFactory—Towards a factory-of-things," *Annual Reviews in Control*, vol. 34, no. 1, pp. 129–138, 2010.

[5] C. Gröger, F. Niedermann, H. Schwarz, and B. Mitschang, "Supporting manufacturing design by analytics, continuous collaborative process improvement enabled by the advanced manufacturing analytics platform," in *Proceedings of the 16th IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2012*, pp. 793–799, 2012.

[6] C. Gröger, L. Kassner, E. Hoos, J. Königsberger, C. Kiefer, S. Silcher, and B. Mitschang, "The Data-driven Factory - Leveraging Big Industrial Data for Agile, Learning and Human-centric Manufacturing," in *Proceedings of the 18th International Conference on Enterprise Information Systems ICEIS 2016*, pp. 40–52, 2016.

[7] "The Industrial Internet of Things - Volume T3: Analytics Framework." `http://www.iiconsortium.org/pdf/IIC_Industrial_Analytics_Framework_Oct_2017.pdf`.

[8] A. Rojko, "Industry 4.0 Concept: Background and Overview," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 11, no. 5, pp. 77–90, 2017.

[9] "Smart Factory Applications in Discrete Manufacturing - An Industrial Internet Consortium White Paper." `http://www.iiconsortium.org/pdf/Smart_Factory_Applications_in_Discrete_Mfg_white_paper_20170222.pdf`.

[10] "Deterministic Networking - BOF Status." `http://www.ieee802.org/1/files/public/docs2014/tsn-nfinn-Deterministic-Networking-BOF-0914-v1.pdf`.

[11] P. Neumann, "Communication in industrial automation - What is going on?," *Control Engineering Practice*, vol. 15, no. 11, pp. 1332–1347, 2007.

[12] "Industrial Analytics: The Engine Driving the IIoT Revolution." `http://www.iiconsortium.org/pdf/Industrial_Analytics-the_engine_driving_IIoT_revolution_20170321_FINAL.pdf`.

[13] B. Hameed, *RFID-Based Real-Time Production Monitoring in a Variant Production Environment*. PhD thesis, Universität Stuttgart, 2016.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan 2015.

[16] M. Johas Teener, A. Fredette, C. Boiger, P. Klein, C. Gunther, D. Olsen, and K. Stanton, "Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging," *Proceedings of the IEEE*, vol. 101, pp. 2339–2354, Nov 2013.

[17] N. Kim, M. Ryu, S. Hong, M. Saksena, C.-H. Choi, and H. Shin, "Visual assessment of a real-time system design: a case study on a CNC controller," in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 300–310, IEEE, 1996.

[18] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on Real-time Communication via Ethernet in Industrial Automation Environments," in *Proceedings of the IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2014.

[19] J. D. Decotignie, "Ethernet-Based Real-Time and Industrial Communications," *Proceedings of the IEEE*, vol. 93, pp. 1102–1117, June 2005.

[20] E. Schemm, "SERCOS to link with ethernet for its third generation," *Computing Control Engineering Journal*, vol. 15, pp. 30–33, April 2004.

[21] E. Tovar and F. Vasques, "Real-time fieldbus communications using Profibus networks," *IEEE Transactions on Industrial Electronics*, vol. 46, no. 6, pp. 1241–1251, 1999.

[22] J.-f. Wan, D. Li, Y.-q. Tu, and C.-h. Zhang, "Performance analysis model for real-time Ethernet-based computer numerical control system," *Journal of Central South University of Technology*, vol. 18, no. 5, pp. 1545–1553, 2011.

[23] "Deterministic Networking." `https://datatracker.ietf.org/wg/detnet/charter/`.

[24] "Deterministic Networking - Architecture." `https://tools.ietf.org/html/draft-finn-detnet-architecture-01.txt`.

[25] "Avnu's Use of 802.1 TSN Mechanisms for Industrial and Automotive Markets." `http://www.ieee802.org/1/files/public/docs2016/liaison-woods-AVnuResponse-0716-v00.pdf`.

[26] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q— as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q—/Cor 1-2015)*, pp. 1–57, March 2016.

[27] "IEEE Standard for Local and metropolitan area networks– Bridges and Bridged Networks - Amendment 24: Path Control and Reservation," *IEEE Std 802.1Qca-2015 (Amendment to IEEE Std 802.1Q— as amended by IEEE Std 802.1Qcd-2015 and IEEE Std 802.1Q—/Cor 1-2015)*, pp. 1–120, March 2016.

[28] F. Dürr and T. Kohler, "Comparing the Forwarding Latency of OpenFlow Hardware and Software Switches," Technical Report Computer Science 2014/04, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, University of Stuttgart, Institute of Parallel and Distributed Systems, Distributed Systems, July 2014.

[29] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," *IEEE Std 802.1AS-2011*, pp. 1–292, March 2011.

[30] 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, pp. 183–192, Oct 2016.

[31] 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*, pp. 203–212, ACM, 2016.

[32] N. G. Nayak, F. Dürr, and K. Rothermel, "Routing Algorithms for IEEE802. 1Qbv Networks," in *Proceedings of the 15th International Workshop on Real-Time Networks*, ACM, 2017.

[33] "Intel's Data Plane Development Kit." `http://dpdk.org/`.

[34] L. Rizzo, "netmap: A Novel Framework for Fast Packet I/O," in *Proceedings of the 21st USENIX Security Symposium*, pp. 101–112, Aug 2012.

[35] 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, pp. 193–202, Oct 2016.

[36] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks," *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 2066–2075, May 2018.

[37] N. G. Nayak, F. Dürr, and K. Rothermel, "Software-defined environment for reconfigurable manufacturing systems," in *Proceedings of the 5th International Conference on the Internet of Things (IOT)*, pp. 122–129, IEEE, Oct 2015.

[38] S. Singh, "Routing Algorithms for Time Sensitive Networks," Master's thesis, Universität Stuttgart, 2017.

[39] S. Zinkler, "In-network packet priority adaptation for networked control systems," Master's thesis, Universität Stuttgart, 2016.

[40] S. Roy Chowdhury, "Packet Scheduling Algorithms for a Software-Defined Manufacturing Environment," Master's thesis, Universität Stuttgart, 2015.

[41] "Open Networking Foundation - OpenFlow." `https://www.opennetworking.org/sdn-resources/openflow`.

[42] "OpenFlow Specifications - v1.4." `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf`.

[43] 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*, pp. 323–334, ACM, 2012.

[44] "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. C1–72, Jan 2009.

[45] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption," *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pp. 1–52, Aug 2016.

[46] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing," *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pp. 1–65, Sept 2017.

[47] "IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, Oct 2017.

[48] "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)," *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pp. 1–119, Sept 2010.

[49] "IEEE Draft Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements," *IEEE P802.1Qcc/D2.0, October 2017*, pp. 1–207, Jan 2017.

[50] K. J. Åström and B. M. Bernhardsson, "Comparison of riemann and lebesgue sampling for first order stochastic systems," in *Proceedings of the 41st IEEE Conference on Decision and Control (CDC)*, vol. 2, pp. 2011–2016, Dec. 2002.

[51] D. J. Antunes and B. A. Khashooei, "Consistent event-triggered methods for linear quadratic control," in *Proceedings of the 55th IEEE Conference on Decision and Control (CDC)*, pp. 1358–1363, Dec. 2016.

[52] W. P. M. H. Heemels, M. C. F. Donkers, and A. R. Teel, "Periodic event-triggered control for linear systems," *IEEE Transactions on Automatic Control*, vol. 58, pp. 847–861, Apr. 2013.

[53] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *European Journal of Operational Research*, vol. 143, no. 3, pp. 498–517, 2002.

[54] W. Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks," in *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS)*, pp. 375–384, IEEE, Nov 2010.

[55] W. Brinkkötter and P. Brucker, "Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments," *Journal of Scheduling*, vol. 4, no. 1, pp. 53–64, 2001.

[56] R. Macchiaroli, S. Mole, and S. Riemma, "Modelling and optimization of industrial manufacturing processes subject to no-wait constraints," *International Journal of Production Research*, vol. 37, no. 11, pp. 2585–2607, 1999.

[57] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[58] S. Bhowmik, M. A. Tariq, A. Balogh, and K. Rothermel, "Addressing TCAM Limitations of Software-Defined Networks for Content-Based Routing," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS*, pp. 100–111, 2017.

[59] A. Beifuß, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle, "A study of networking software induced latency," in *Proceedings of the International Conference and Workshops on Networked Systems (NetSys)*, pp. 1–8, IEEE, March 2015.

[60] P. Erdős and A. Rényi, "On random graphs I," *Publicationes Mathematicae 6*, pp. 290–297, 1959.

[61] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[62] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15, Aug 2008.

[63] "CPLEX Optimizer." `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[64] S. S. Craciunas and R. S. Oliver, "Combined Task- and Network-level Scheduling for Distributed Time-triggered Systems," *Real-Time Systems*, vol. 52, pp. 161–200, Mar. 2016.

[65] S. S. Craciunas and R. S. Oliver, "SMT-based task-and network-level static schedule generation for time-triggered networked systems," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, pp. 45–54, ACM, 2014.

[66] C. Scholer, R. Krenz-Baath, A. Murshed, and R. Obermaisser, "Computing Optimal Communication Schedules for Time-triggered Networks Using an SMT solver," in *Proceedings of the 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pp. 83–91, May 2016.

[67] S. S. Craciunas and R. Serna Oliver, "An Overview of Scheduling Mechanisms for Time-sensitive Networks." Proceedings of the Real-time summer school LÉcole dÉté Temps Réel (ETR), 2017.

[68] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet IO IRT Message Scheduling With Temporal Constraints," *IEEE Transactions on Industrial Informatics*, vol. 6, pp. 369–380, Aug 2010.

[69] J. Dvořák, M. Heller, and Z. Hanzálek, "Makespan minimization of Time-Triggered traffic on a TTEthernet network," in *Proceedings of the 13th IEEE International Workshop on Factory Communication Systems (WFCS)*, pp. 1–10, May 2017.

[70] E. B. 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 2017, Grenoble, France, October 04 - 06, 2017*, pp. 8–17, 2017.

[71] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp," in *Proceedings of the 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug 2018.

[72] R. Mahfuzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-Aware Integrated Routing and Scheduling for Control Applications in Ethernet Networks," in *Design, Automation and Test in Europe (DATE)*, no. EPFL-CONF-232889, 2018.

[73] D. Maxim and Y.-Q. Song, "Delay Analysis of AVB Traffic in Time-sensitive Networks (TSN)," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*, RTNS '17, pp. 18–27, ACM, 2017.

[74] L. Zhao, P. Pop, Q. Li, J. Chen, and H. Xiong, "Timing analysis of rate-constrained traffic in TTEthernet using network calculus," *Real-Time Systems*, vol. 53, no. 2, pp. 254–287, 2017.

[75] W. Steiner, "Synthesis of Static Communication Schedules for Mixed-Criticality Systems," in *Proceedings of the 14th IEEE International Symposium on Object/Component/Service Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pp. 11–18, IEEE, March 2011.

[76] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of Communication Schedules for TTEthernet-based Mixed-criticality Systems," in *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 473–482, ACM, 2012.

[77] J. Specht and S. Samii, "Urgency-based scheduler for time-sensitive switched ethernet networks," in *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 75–85, IEEE, 2016.

[78] R. Serna Oliver, S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding," in *Proceedings of 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018.

[79] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[80] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[81] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks," *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, 2014.

[82] "IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges," *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pp. 1–277, 2004.

[83] J. W. Guck, A. V. Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2017.

[84] S. M. Laursen, P. Pop, and W. Steiner, "Routing Optimization of AVB Streams in TSN Networks," *SIGBED Rev.*, vol. 13, pp. 43–48, Nov. 2016.

[85] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii, "Fault-tolerant Topology and Routing Synthesis for IEEE Time-sensitive Networking," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, (New York, NY, USA), pp. 267–276, ACM, 2017.

[86] A. M. Kentis, M. S. Berger, and J. Soler, "Effects of port congestion in the gate control list scheduling of time sensitive networks," in *Proceedings of the 8th International Conference on the Network of the Future (NOF)*, pp. 138–140, Nov 2017.

[87] J. Aracil and F. Callegati, *Enabling Optical Internet with Advanced Network Technologies*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[88] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design Optimisation of Cyber-physical Distributed Systems using IEEE Time-sensitive Networks," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[89] S. Mitchell, M. O'Sullivan, and I. Dunning, "PuLP: A Linear Programming Toolkit for Python," 2011.

[90] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988.

[91] T. Qian, F. Mueller, and Y. Xin, "A Linux Real-Time Packet Scheduler for Reliable Static SDN Routing," in *Proceedings of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017)* (M. Bertogna, ed.), vol. 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 25:1–25:22, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[92] R. Kumar, M. Hasan, S. Padhy, K. Evchenko, L. Piramanayagam, S. Mohan, and R. B. Bobba, "End-to-End Network Delay Guarantees for Real-Time Systems using SDN," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Nov 2017.

[93] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues Don'T Matter when You Can JUMP Them!," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, (Berkeley, CA, USA), pp. 1–14, USENIX Association, 2015.

[94] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren, "Practical TDMA for datacenter Ethernet," in *Proceedings of the 7th ACM European conference on Computer Systems*, pp. 225–238, ACM, 2012.

[95] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.

[96] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 63–74, ACM, 2008.

[97] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pp. 267–280, ACM, 2010.

[98] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pp. 65–72, ACM, 2009.

[99] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware Datacenter TCP (D^2TCP)," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115–126, 2012.

[100] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 127–138, ACM, 2012.

[101] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50–61, 2011.

[102] E. Schweissguth, P. Danielis, C. Niemann, and D. Timmermann, "Application-aware Industrial Ethernet based on an SDN-supported TDMA Approach," in *Proceedings of the IEEE World Conference on Factory Communication Systems (WFCS)*, pp. 1–8, May 2016.

[103] J. Kiszka and B. Wagner, "RTnet - a flexible hard real-time networking framework," in *IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, pp. 8 pp.–456, Sept 2005.

[104] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *CIRP Annals-Manufacturing Technology*, vol. 65, no. 2, pp. 621–641, 2016.

[105] R. S. Oliver, S. S. Craciunas, and G. Stöger, "Analysis of Deterministic Ethernet scheduling for the Industrial Internet of Things," in *Proceedings of the 19th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 320–324, Dec 2014.

[106] T. Kohler, F. Dürr, and K. Rothermel, "Consistent Network Management for Software-Defined Networking Based Multicast," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 447–461, Sept 2016.

[107] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN Controller Assignment in Data Center Networks: Stable Matching with Transfers," in *Proceedings of the IEEE INFOCOM 2016*, pp. 1–9, April 2016.

[108] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring Control Plane Latency in SDN-enabled Switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, 2015.

[109] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 43–48, ACM, 2013.

[110] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring Control Plane Latency in SDN-enabled Switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pp. 25:1–25:6, ACM, 2015.

[111] R. McGeer, "A Safe, Efficient Update Protocol for Openflow Networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pp. 61–66, ACM, 2012.

[112] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 539–550, ACM, 2014.

[113] P. Danielis, G. Dan, J. Gross, and A. Berger, "Dynamic Flow Migration for Delay Constrained Traffic in Software-Defined Networks," in *Proceedings of the IEEE Global Communications Conference, GLOBECOM*, pp. 1–7, Dec 2017.

[114] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, 2010.

[115] F. Greff, Y.-Q. Song, L. Ciarletta, and A. Samama, "A dynamic flow allocation method for the design of a software-defined real-time mesh network," in *Proceedings of the 13th International Workshop on Factory Communication Systems (WFCS)*, pp. 1–11, IEEE, 2017.

[116] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, pp. 85–103, Springer, 1972.

[117] M. Blesa and C. Blum, "Ant colony optimization for the maximum edge-disjoint paths problem," in *Workshops on Applications of Evolutionary Computation*, pp. 160–169, Springer, 2004.

[118] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[119] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," in *ACM SIGCOMM Computer Communication Review*, vol. 21, pp. 43–51, ACM, 1991.

[120] M. Handley, O. Bonaventure, C. Raiciu, and A. Ford, "RFC 6824 - TCP extensions for multipath operation with multiple addresses," 2013.

[121] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 27–38, ACM, 2008.

[122] S. Bhowmik, M. A. Tariq, B. Koldehofe, A. Kutzleb, and K. Rothermel, "Distributed Control Plane for Software-defined Networks: A Case Study Using Event-based Middleware," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS '15, pp. 92–103, ACM, 2015.

[123] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the 1st workshop on Hot topics in software defined networks*, pp. 19–24, ACM, 2012.

[124] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *Proceedings of the Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2010.

[125] F. Pozo, G. Rodriguez-Navas, W. Steiner, and H. Hansson, "Period-aware segmented synthesis of schedules for multi-hop time-triggered networks," in *Proceedings of the 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 170–175, Aug 2016.

[126] D. Tamas-Selicean, P. Pop, and W. Steiner, "Timing Analysis of Rate Constrained Traffic for the TTEthernet Communication Protocol," in *Proceedings of the 18th IEEE International Symposium on Real-Time Distributed Computing*, pp. 119–126, April 2015.

[127] J. Diemer, D. Thiele, and R. Ernst, "Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching," in *Proceedings of the 7th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 1–10, IEEE, 2012.

[128] D. Thiele, R. Ernst, and J. Diemer, "Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers," in *Proceedings of the 2015 IEEE Vehicular Networking Conference (VNC)*, pp. 251–258, Dec 2015.

[129] S. Bhowmik, M. A. Tariq, B. Koldehofe, F. Dürr, T. Kohler, and K. Rothermel, "High Performance Publish/Subscribe Middleware in Software-Defined Networks," *IEEE/ACM Transactions on Networking*, vol. 25, pp. 1501–1516, June 2017.

[130] S. Bhowmik, M. A. Tariq, L. Hegazy, and K. Rothermel, "Hybrid Content-Based Routing Using Network and Application Layer Filtering," in *Proceedings of the 36th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 221–231, June 2016.

[131] M. A. Tariq, B. Koldehofe, S. Bhowmik, and K. Rothermel, "PLEROMA: A SDN-based High Performance Publish/Subscribe Middleware," in *Proceedings of the 15th International Middleware Conference*, Middleware '14, pp. 217–228, ACM, 2014.

ERKLÄRUNG

Ich erkläre hiermit, dass ich, abgesehen von den ausdrücklich bezeichneten Hilfsmitteln und den Ratschlägen von jeweils namentlich aufgeführten Personen, die Dissertation selbstständig verfasst habe.

(Naresh Ganesh Nayak)