Technische Universität München
Lehrstuhl für Kommunikationsnetze

Centralized Online Routing for Deterministic Quality of Service in Packet Switched Networks

Jochen Walter Guck, M.Sc.

Vollständiger Abdruck der von der Fakultät Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Andreas Herkersdorf
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Wolfgang Kellerer
	2.	Prof. Martin Reisslein, Ph.D.

Die Dissertation wurde am 11.01.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 30.04.2018 angenommen.

Centralized Online Routing for Deterministic Quality of Service in Packet Switched Networks

Jochen Walter Guck, M.Sc.

18. Juni 2018

Abstract

The main purpose of industrial networks is to transmit critical messages (e.g., the control signals for large automated manufacturing plants). Deterministic real-time Quality of Service (QoS) is a key requirement for many of these critical messages Proprietary industrial communication technologies have been developed to provide this strict QoS. These proprietary technologies are typically costly and lack a common accepted standardized communication interface. This work takes up the original ideas from industrial Ethernet of using commodity hardware to realize industrial real-time communication. To achieve this, modeling techniques have been developed that enable the provision of deterministic real-time communication using commodity hardware. In addition, optimization strategies were designed and evaluated, which maximize the number of served connections.

In this Thesis, the implementation of a centralized deterministic Quality of Service Framework is presented. It was developed for the industrial real-time communication use case. To provide this type of communication, a deterministic end-to-end delay path is needed. To enable fast online routing capabilities in packet-switched networks, a function split has been introduced. This divides the global optimization problem into three independent optimization problems, namely the routing problem, the cost function design, and the resource allocation problem. These three issues work on the basis of a network resource model. In this work, several deterministic network resource models are presented.

To evaluate the centralized deterministic Quality of Service control framework, a stand-alone evaluation of all components was not enough to determine the best system performance. We use a Monte Carlo-based simulation to create a record of system configurations. For these configurations, the maximum traffic intensity and life-cycle time were calculated. The resulting data set was used to identify a high performance configuration for maximum traffic intensity (more than 1000 connections on average) and a suitable computational effort (about one hundred requests per second).

These results show that the implemented centralized deterministic quality of service control framework based on commodity hardware meets industrial requirements.

Zusammenfassung

Der Hauptzweck von industriellen Netzwerken besteht darin, kritische Nachrichten (z.B. die Steuerungssignale für große automatisierte Produktionsanlagen) zu übertragen. Deterministische Echtzeit Quality of Service (QoS) ist eine Schlüsselanforderung für viele dieser kritischen Nachrichten. Proprietäre industrielle Kommunikationstechnologien wurden entwickelt, um diese strikte QoS bereitzustellen. Diese proprietären Technologien sind typisch kostenintensiv und es fehlt ein einheitliches, akzeptiertes und standardisiertes Kommunikationsinterface. Diese Arbeit greift die ursprünglichen Ideen des industriellen Ethernet der Nutzung von Commodity Hardware auf, um industrielle Echtzeitkommunikation zu realisieren. Um dies zu erreichen, wurden Modellierungstechniken entwickelt, die die Bereitstellung von deterministischer Echtzeitkommunikation, unter Verwendung von Commodity Hardware, ermöglicht. Zusätzlich wurden Optimierungsstrategien entworfen und evaluiert, welche die Anzahl der Echtzeitverbindungen maximieren.

Die vollständige Implementierung eines zentralisierten deterministischen Quality of Service Frameworks wird in dieser Arbeit vorgestellt. Es wurde für den industriellen Echtzeitkommunikationsanwendungsfall entwickelt. Um diese Art von Kommunikation bereitzustellen wird ein Pfad mit deterministischer Ende-zu-Ende-Verzögerung benötigt. Um schnelle Online-Routing-Fähigkeiten in paketvermittelten Netzen zu ermöglichen, wurde der Funktionssplit eingeführt. Dieser unterteilt das globale Optimierungsproblem in drei unabhängige Optimierungsprobleme, nämlich das Routing-Problem, das Kostenfunktionsdesign und das Ressourcenzuordnungsproblem. Diese drei Probleme arbeiten auf der Grundlage eines Netzwerkressourcenmodells. In dieser Arbeit wurden mehrere deterministische Netzwerkressourcenmodelle vorgestellt.

Um das zentralisierte deterministische Quality of Service-Kontrollframework zu bewerten, reichte eine eigenständige Evaluierung aller Komponenten nicht aus, um die beste Systemleistung zu ermitteln. Eine Monte-Carlo-basierte Simulation wurde verwendet, um einen Datensatz von Systemkonfigurationen zu erstellen. Für diese Konfigurationen wurden die maximale Verkehrsintensität und die Lebenszyklus-Laufzeit berechnet. Der resultierende Datensatz wurde verwendet, um eine Konfiguration mit einer hohen Leistung hinsichtlich der maximalen Verkehrsintensität (mehr als 1000 Verbindungen im Durchschnitt) und eines geeigneten Berechnungsaufwands (ungefähr hundert Anfragen pro

Sekunde) zu identifizieren.
Die Ergebnisse der Evaluation zeigen, dass die Implementierung eines zentralisierten deterministischen Quality of Service-Kontrollframeworks für auf Basis von Commodity Hardware möglich ist.

Contents

List of Figures

List of Tables

1. Introduction

Industrial networks carry critical messages, e.g., control signals, for large automated production facilities. Many of these critical messages must be delivered with tight deterministic real-time Quality of Service (QoS) [42] requirements. A wide gamut of proprietary industrial communications technologies have emerged to provide this strict QoS [44]. These proprietary technologies are typically costly and lack a uniformly accepted standardized communication framework.

To understand the need of this proprietary solution the networking QoS capabilities have to be studied. State of the art QoS mechanisms try to adapt the allocation of the network resources to the demands of services running over this network. The goal is to provide each service with the required network resources given the possibly limited overall available resources. Unfortunately, these services have different QoS requirements, such as latency, jitter, error-rate, throughput, and redundancy. It is difficult to design a system that addresses all the requirements. Two standardized frameworks attempt to overcome these problems [117] in the Internet. Integrated Services (IntServ) [138] was the first framework to provide network QoS. With IntServ, the network is able to provide end-to-end QoS connections per flow. However, the computational effort for establishing these end-to-end connections is huge. As a result, the scalability of this framework is poor. The next step in the standardization process was Differentiated Services (DiffServ) [139]. This framework is designed for scalability but has low granularity in flow control. Furthermore, the implementation of DiffServ is highly proprietary, leading to difficulties in its management [140]. As a result both standardized QoS frameworks do not provide the features to enable deterministic QoS for large automated production facilities.

Decotingie [15] provided a broad overview of Ethernet-based real-time communication, including deterministic Ethernet standards. The opportunity of cost reduction by using off-the-shelf commodity hardware for real-time communication instead of specialized proprietary communication systems was one trigger for the shift to Ethernet-based technology [44, 119]. However, commodity hardware does not fulfill all requirements of real-time communications, which has led to the development of proprietary Ethernet-based solutions [43]. The proprietary industrial real-time communication solutions that fulfill hard real-time requirements need changes of the network protocol stack or topology

restrictions or both, leading to expensive hardware and software.
This thesis revisits the initial ideas of industrial Ethernet of using commodity hardware to realize industrial real-time communications. Therefore, modeling techniques are introduced which enable the provisioning of deterministic real-time communication by using commodity packet switched hardware. In addition, optimization strategies are designed and evaluated to maximize the number of served connections.

1.1. Problem Statement

There are two big challenges in real-time communication. The first issue is that of simultaneousness, meaning that all devices have to be time-synchronized to fulfill their task. There are several time synchronization protocols specified by the IEEE which could be used for this purpose (NTP,PTP). Therefore, time synchronization is out of scope of this thesis. The second challenge is timeliness; all information that is important for the system has to be transmitted before a particular deadline. A violation of this deadline could cause damage to the factory site. The usual communication partners in industrial scenarios are the Programmable Logic Controllers (PLCs) which run the automation software and the remote-I/O which is connected it via industrial Ethernet. The remote-I/O provides interface to the sensors and actuators connected to it. Each PLC remote-I/O connection requires a specific QoS. To design a proper industrial real-time communication which enhances the state of the art, the following seven problems have to be solved:

Problem 1 (Deterministic end-to-end delay guarantees)**:** *The main challenge of industrial real-time systems is to provide guarantees on the end-to-end delay. This guarantees are needed because industrial real-time connections usually carry control information of a real physical process. If there are violations of the delay requirements the proper operation of the entire industrial facility is in danger.*

Problem 2 (Online flow management)**:** *Since not all industrial scenarios allow a beforehand communication planning the industrial real-time system has to deal with online requests. Those requests are defined as a dynamic adding and removal of real-time flows.*

Problem 3 (Sufficient flow capacity)**:** *The amount of supported flows has to be sufficient for nowadays industrial scenarios. An amount of 100 to 1000 real-time connections per PLC will provide a sustainable performance that also holds for the future.*

Problem 4 (No topology limits)**:** *Since the communication technology should not restrict the installation, the industrial real-time system should be able to deal with all kind of topologies.*

Problem 5 (Commodity Hardware)**:** *The current Ethernet based real-time communication systems have to extend the standard Ethernet to meet real-time requirements. This leads to expensive special purpose solutions. To decrease the cost it is necessary to use commodity hardware only.*

Problem 6 (Cross traffic)**:** *The communication system should support non real-time traffic in parallel. This enables a shared infrastructure which reduces the costs.*

Problem 7 (Large topologies)**:** *Since future industrial real-time systems could cover whole factory sites, the system should handle more than 100 network nodes. But not only switches and routers are networking nodes in a industrial scenario. PLCs and remote-I/Os could also contain an internal switch, which makes them also part of the network.*

Problem 1 is well known in the state of the art. There are several approaches which provide deterministic end-to-end delay bounds. However, providing these bounds only is not enough to meet a broad area of industrial use cases. Therefore, we introduce problems 2-7. These additional problems ensure that our solution to Problem 1 can be applied to more general scenarios.

1.2. Contributions

In this thesis we present a solution for all problems raised in section 1.1. We have designed, implemented and evaluated a centralized QoS framework which meets the requirements of our problems. The main contributions of this work are:

Function Split: the concept of a function split is introduced to break down the computational complexity. Therefore, the network resource management and the routing process itself are split into two separate problems. This split enables a fast online routing process. To increase the overall performance, a background optimization of the resource allocation is proposed.

Deterministic Services (DetServ) network models: a set of network calculus based network resource models are defined in this thesis. These models provide different accuracy levels of their worst case bounds, increasing the maximum number of connections at the cost of a higher model complexity.

Analysis of Constrained Shortest Path (CSP) routing algorithms: since the state of the art of CSP algorithms provide already a broad variety of problem solutions, a performance evaluation of 26 algorithms is presented in this thesis. This analysis is done for industrial real-time scenarios, in particular.

Simple forwarding plane requirements: the only requirements for achieving deterministic real-time communication is that every switch has a priority scheduler for every outgoing link and provides an interface which enables to install queue based forwarding rules. These two assumptions match to a broad range of commodity networking hardware.

Proof of concept implementation: The four main building blocks (Resource Model, Routing, Cost Function and Resource allocation) were implemented to validate the capabilities of a function split based real-time communication system.

System Evaluation: Since all building blocks have cross dependencies with an impact on their performance, a combined evaluation of all implementation parameter is needed. Therefore, a Monte-Carlo method based simulation was used to identify the building blocks and parameter, which have a significant performance impact.

Practical Test: during the VirtuWind [127] project, the work presented in this thesis was used to provide the intra domain real-time communication. As part of this project, there was a field trial at a wind park performed by a colleague.

Patent submissions: the results of this work has led to three patent submissions [7, 8].

Table 1.1 links the contributions of this section to the problems defined in section 1.1. We provide more details about this linking in the following.
The Function Split tackles problem 1,2 and 5. Since providing deterministic end-to-end real time connections on commodity hardware leads to a problem that consists of multiple NP-hard problems, the function split was introduced to reduce the computational complexity. Indeed, the Function Split breaks the overall problem down to a single CSP search which is still NP-hard but well researched in state of the art.
The DetServ models are a result of the Function Split. They model the network resources in a way which enables the provisioning of deterministic end-to-end real time connections by using CSP algorithms for the path computation. By providing this feature, the DetServ models contribute to problem 1,2 and 5. In

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5	Problem 6	Problem 7
Function Split	X	X			X		
DetServ network models	X	X		X	X	X	
Analysis of CSP routing algorithms		X					
Simple forwarding plane requirements					X		
Proof of concept implementation	X	X	X	X	X	X	X
System Evaluation	X	X	X	X	X	X	X
Practical Test	X	X	X	X	X	X	X
Patent submissions	X	X					

Table 1.1.: Overview of contributions mapped to initial problem statements

addition the models are not limited to any topology which is demanded from problem 4. And furthermore they are tolerant to cross traffic, which is required by problem 6.
Since CSP algorithms provide the paths used by the QoS framework, they are the main influence on the runtime. The performance comparison of state of the art algorithms will show which of those are suitable for our online scenario (problem 2).
Since the DetServ models are designed to work on commodity hardware, the requirements on the forwarding plain are quite low. There is plenty of hardware available fulfilling this necessary requirements, so problem 5 is covered.
The complete proof of concept implementation of the QoS framework allows a full evaluation. In addition, a practical test as part of the VirtuWind project was performed by a colleague. This shows that all seven problems are covered by the QoS framework.
The research activity to solve problem 1 and 2 led to three patent submission.

1.3. Thesis Outline

This thesis consists of six chapters, in addition to this introductory section. We will outline the remaining chapters in this section.

Chapters 2 "Industrial QoS Framework": Within this chapter an overview of existing Ethernet based QoS measures, with respect to the industrial use case is provided. Since no existing industrial Ethernet solution could provide deterministic end-to-end real-time QoS without any extension to the Ethernet standard, the structure of our novel industrial QoS framework is

blueprinted. The Function Split presented and its underlying subproblems are introduced.

Chapters 3 "State of the Art Analysis": We focus our state of the art section on two main topics. The first main topic covers the research field of centralized QoS frameworks. Therefore, we classify and outline the most relevant work. The second research field is the existing work of CSP routing. Therefore, we provide an extensive survey of CSP algorithms suitable for a centralized QoS framework.

Chapters 4 "Network Resource Modeling: DetServ": The network resource models are designed to provide deterministic end-to-end delay guarantees and to eliminate any queue based packet loss. Furthermore, it enables a CSP algorithm to find paths which will provide this QoS.

Chapters 5 "Optimization Problems": The Function Split subdivides the overall path embedding problem into three subproblems. First, a state of the art CSP algorithm **(Section 5.1: Routing Problem)** solves the overall problem partly. The solution of the CSP is a route through a network which is optimal in terms of its edge based cost. A cost function has to be designed to maximize the amount of real-time flows in the network **(Section 5.2: Cost Function design Problem)**. The network resource modeling needs a preallocation of resources to achieve a deterministic runtime behavior. This preallocation has to be adapted to the current network state to maximize the performance **(Section 5.3: Resource Allocation Problem)**.

Chapters 6 "Evaluation": This chapter covers three evaluation targets. The first evaluation **(Section 6.1: CSP Algorithm Evaluation)** provides a performance comparison of 26 state of the art CSP algorithms. This evaluation is the most extensive comparison of state of the art CSP algorithms. The second evaluation **(Section 6.2: Function Split Evaluation)** focuses on the performance implications of the function split. Therefore, the Function Split based online solution is compared with a Mixed Integer Program (MIP) based solution. The last evaluation **(Section 6.3: Industrial QoS Framework Evaluation)** targets an overall analysis of the different implementation options of the industrial QoS framework.

Chapters 7 "Conclusion and Outlook": Finally, we provide a conclusion and a future work outlook.

2. Industrial QoS Framework

Industrial Quality of Service (QoS) networking with a centralized controller poses the general problems of path computation (routing) and resource management. The joint solution of the combined routing and resource management problem is NP-hard. This can be derived from the observation that the path computation problem corresponds to the well-known Constrained Shortest Path (CSP) routing problem [93], which is NP-complete. Thus, the combined problem, which adds the resource management problem to the CSP problem, is also NP-hard. The combined routing and resource allocation problem can be formulated (with simplifications of some quadratic equations [1]) as a Mixed Integer Program (MIP). The MIP solves both the routing and resource allocation problems jointly but incurs prohibitive computation times already for small networks (see comparisons in Section 6.2). The monolithic MIP solution approach cannot be employed for run-time decisions in realistic-sized industrial networks.
In order to reduce the computational effort for solving the general routing and resource management problem so as to operate centralized industrial QoS networks at run-time, we split the problem into two functions: an online function for CSP routing [137] and an offline function for resource allocation. We operate the online routing (as explained in Section 2.5) so that no QoS relevant information has to be measured at the forwarding plane.
In Section 2.1 the context of our QoS framework is described. Based on the analysis of the context the basic concept of the industrial QoS framework is presented in Section 2.2. A generic problem formulation (Section 2.3), which covers the objectives of this thesis in a formal way, is presented. Since this basic problem is too complex the Function Split is introduced in section 2.4. In Section 2.5 implementation insights are provided. This chapter is based on our work presented in [1, 2, 3].

2.1. Context: Industrial QoS Framework

This section covers four topics relevant for the industrial QoS framework. First an overview of Ethernet/IP-based QoS measures is provided. In addition, the relevance of these measures is discussed in the context of industrial real-time communication. The second topic is Industrial Ethernet itself. Mainly the shortcomings of the Industrial Ethernet solutions are discussed. Time Sensitive

Networking (TSN) (third topic) is the latest standardization activity tackling these shortcomings. Finlay, Software Defined Networking (SDN) is introduced. Additionally the usage of SDN for QoS applications is discussed.

2.1.1. Ethernet/IP-based QoS

Ethernet/IP-based QoS mechanisms try to adapt the allocation of the network resources to the demands of services running over a network. The goal is to provide each service with the required network resources given the possibly limited overall available resources. Unfortunately, these services have different QoS requirements, such as latency, jitter, error-rate, throughput, and redundancy. It is difficult to design a system that addresses all requirements. Two standardized frameworks (Integrated Services (IntServ) - Section 2.1.1.1, Differentiated Services (DiffServ) - Section 2.1.1.2) attempt to overcome these problems [117] in the Internet. Alongside these frameworks, standardization methods were developed to support QoS at the forwarding plane. There are three main methods which are common sense. First, there is packet based scheduling (Section 2.1.1.3), second there is Active Queue Management (AQM) (Section 2.1.1.4) and third there is traffic shaping (Section 2.1.1.5). Alongside with each section the applicability of each IP-based QoS method will be discussed.

2.1.1.1. Integrated Services (IntServ)

IntServ [138] was the first framework to provide network QoS. With IntServ, the network is able to provide end-to-end QoS connections per flow. Three working modes can be reserved via Resource reSerVation Protocol (RSVP). The first mode is a simple best effort mode. The second mode is a bandwidth reservation mode where all switches along the path provide if possible the needed bandwidth for a flow exclusively. The last mode provides a delay guarantee to the flow. Therefore, the delay requirement and the token bucket characteristics (avg. data rate r; burstiness b) have to be transmitted via RSVP. Parekh and Gallager[151, 150] provided a method to calculate worst case delay bounds under the assumption that all IntServ nodes can provide a per flow Generalized Processor Sharing (GPS). However, the computational effort for establishing these end-to-end connections is huge. As a result, the scalability of this framework is poor [140]. In general, IntServ provides the features needed to instantiate industrial real time communication within a network. Therefore, the following problems (Problem 1: Deterministic end-to-end delay guarantees, Problem 2: Online flow management,Problem 4: No topology limits and Problem 6: Cross traffic) are solved properly. However, there is a main problem which is not resolved. Since

IntServ never reached a high market penetration [140] there is no cheap commodity hardware available. So even if IntServ is fully standardized, Problem 5: Commodity Hardware stays unresolved.

2.1.1.2. Differentiated Services (DiffServ)

The next step in the standardization process was DiffServ [139]. This framework is designed for scalability but has low granularity in flow control. To achieve high scalability, DiffServ provides QoS on a flow class basis. Particular flows will be mapped to a flow class when they enter the network. While traversing the network, all QoS measures (i.e. queuing, traffic shaping) will be applied to the flow class aggregate. This reduces the complexity of a DiffServ network, in contrast to an IntServ network. However, the implementation and configuration of DiffServ forwarding devices is highly proprietary, leading to difficulties in its management [140].
Since DiffServ handles only flow aggregates the Problem 7: Large topologies is handled properly. There is plenty of commodity DiffServ hardware available. So DiffServ will solve Problem 5: Commodity Hardware. However, solving Problem 1: Deterministic end-to-end delay guarantees will require a fine granular per flow management which is hard to achieve.

2.1.1.3. Queue Scheduling

Alongside with development of DiffServ, network hardware vendors developed queue scheduling disciplines to support differentiated service classes in large IP networks. The main task of queue scheduling algorithms in packet switched networks is to decide which packet has to be sent next on the output port. There are many different scheduling disciplines which are a trade-off between **complexity**, **control**, and **fairness** [152]. Unfortunately there is no industrial standard. However, there are some common base disciplines, which are used by the vendors to create more complex scheduling disciplines [152]. In the following the following base disciplines are introduced:

First-in, First-out (FIFO) Scheduling is the most basic scheduler implementation. The packets will be transmitted in the order they arrive. The **complexity** of this scheduling scheme is very low. Because pure FIFO Scheduling does not differentiate flows or service classes there is no **control**. FIFO Scheduling cannot provide **fairness** between busty and non busty traffic since both traffic classes will experience packet loss because of the burstiness [152].

Priority Queuing (PQ) is the basic implementation of a service class based scheduler. The flows which should be transmitted are classified. Based on this classification the packets of these flows will be put into FIFO queues of different priority. The packet on the head of a queue will be only sent next if all higher priority queues are empty. The **complexity** of this scheduling scheme is also very low. PQ provides **control** over the packet handling for different service classes. While high priority flows are not effected by low priority flows, low priority flows could suffer (high delay, jitter, packet loss) from the misbehavior of high priority flows. This could influence the service **fairness** negatively [152].

Weighted Round Robin (WRR) Scheduling provides the same classification mechanisms as PQ. The scheduler traverses the queues always in the same order. Every time a queue is visited, a predefined amount of packets is sent. If no packets are left the scheduler will continue at the next queue. WRR Scheduling provides only a little higher **complexity** than PQ. Therefore, it can also be implemented in high speed networks. It provides **control** (as PQ) over the packet handling for different service classes. WRR assures that every service class receives a minimum amount of bandwidth, which enables a better **fairness** than PQ. Please note that the bandwidth distribution is packet-size dependent [152].

Fair Queuing (FQ) is a Round Robin (RR) scheduler on a per flow basis. Each flow is directed to its own queue which is scheduled by a RR scheduler. This approach provides **fairness** trough an equal distribution of the available bandwidth to each flow. On th other hand this implementation is packet size dependent. Flows with smaller packets will receive a smaller portion of the bandwidth. The implementation of this discipline is rather **complex** and therefore done in software. It provides **control** by definition since every flow will be handled separately [152].

Weighted Fair Queuing (WFQ) is an enhancement of FQ. The main change is the replacement of the RR scheduler with a GPS system. By doing this it becomes possible to allocate a minimum data rate per flow. This also resolves the packet size dependency of FQ. WFQ treats all flows **fair** depending on their data rate configuration. The implementation of WFQ is more **complex** than FQ. The **control** behavior is the same. Please note that there is the Class-based WFQ implementation which does not handle every flow within one queue. Instead flows are based on their classification direct into a small amount of queues. Fore these queues a minimum rate is configured. So the complexity is decreased at the cost of fairness, since now fairness is only valid for service classes [152].

Since PQ, WRR and class based WFQ are implementable with reasonable effort, due to their low **complexity**, they are available in commodity hardware. The usage of these strategies would solve Problem 5: Commodity Hardware. However, since flows are handled as aggregates there is no straightforward way to address Problem 1: Deterministic end-to-end delay guarantees with these simple schedulers.

2.1.1.4. Active Queue Management (AQM)

AQM was introduced to improve the behavior of the network in terms of buffer overflows. In order to wait until the buffer is full and the packets which arrive in this state get dropped, more intelligent schemes are applied [153]. There are two possible actions, an AQM algorithm can perform. Either selected packets will be dropped earlier to force Transmission Control Protocol (TCP) to throttle the transmission speed, which should resolve the congestion. Or the corresponding upstream nodes are notified to throttle their transmission speed. Achieving a deterministic communication service (Problem 1) is, due to the dynamic nature of both actions, hard to reach. Therefore, AQM is out of the scope of this thesis.

2.1.1.5. Traffic Shaping

Beside queue scheduling and AQM, traffic shaping is the third well known technique used for QoS provisioning. It is used to force a flow or an aggregate of flows into a specific traffic pattern. There are two traffic pattern which are widely used:

Leaky Bucket: A leaky bucket shaper could be seen as a queue with a specified service rate R. The maximum sending rate is limited to R for flows traversing this shaper. The flow(s) will experience some delay waiting for transmission, but therefore the burstiness is upper bounded by R after leaving the leaky bucket.

Token Bucket: A token bucket is a more advanced shaping mechanism. There is one queue and one bucket. One queue stores the packets of the incoming flows. The bucket stores tokens which represent a number of bits which are allowed to be transmitted. Only if the amount of bits in the bucket is bigger than the packet size the packet will be send. Tokens will be generated with a data rate R. The maximum amount of tokens is limited to a burstiness B. The flow(s) will experience some delay waiting for transmission, but therefore the burstiness is upper bounded by R on the long run. In addition there is a burst of B bytes allowed. This allows the modeling of more dynamic traffic which is still upper bounded.

The usage of a token or leaky bucket shaper would not contradict Problem 5: Commodity Hardware because they are common in todays networking hardware. They are a corner stone for solving Problem 1: Deterministic end-to-end delay guarantees with commodity hardware. Because only if the arriving traffic is upper bounded it is possible to calculate any worst case bounds [151, 150].

2.1.2. Industrial Ethernet

Initially, proprietary solutions (like Profibus, Interbus or CAN) have been specifically developed for real-time industrial communications [121, 130]. These solutions often come with a complete proprietary communication stack, which requires specialized and expensive hardware.
Later, Ethernet data transfer rates increased and Ethernet became ubiquitous in local area networks (LANs) and the Internet. Therefore, it attracted a lot of attention for industrial deployments. However, because of its non-deterministic medium access control (MAC) scheme, Ethernet was initially not considered as a suitable solution. The usage of full duplex point-to-point links along with Ethernet switches instead of shared buses and hubs allowed to avoid collisions and hence the negative impact of the Ethernet MAC protocol [15]. Nevertheless, this introduces buffering and possibly overflows, which were still considered to be a source of non-determinism [15]. Despite this, using Ethernet in industrial environments has major benefits, including simple and cheap deployment, easy connectivity towards office networks, the Internet or more generally any IP traffic, and usage of off-the-shelf communication hardware. Hence, many industrial control system manufacturers decided to develop proprietary extensions of Ethernet to achieve determinism [112, 121]. A broad overview of Ethernet-based real-time technologies, including deterministic Ethernet standards, was provided by Decotignie [15]. Unfortunately, these solutions require changes within the network protocol stack or impose topology restrictions or both, which leads to more expensive forwarding devices than for standard Ethernet.
So all Industrial Ethernet solutions available which solve Problem 1 (Deterministic end-to-end delay guarantees) will fail in solving Problem 5: Commodity Hardware. In this thesis we will show a methodology which enables solving both problems together. By doing this it is possible to enable the cheap commodity hardware to carry industrial real-time communication which was the initial goal of Industrial Ethernet.

2.1.3. Time Sensitive Networking (TSN)

The IEEE 802.1 working group is currently developing TSN as a set of standards which define mechanisms for time-sensitive transmissions over Ethernet. TSN

Standard	Name	Release
802.1Qbu	Frame Preemption	Published
802.1Qbv	Enhancements for Scheduled Traffic	Published
802.1Qca	Path Control and Reservation	Published
802.1AS-Rev	Timing and Synchronization for Time-Sensitive Applications	Draft
802.1CB	Frame Replication and Elimination for Reliability	Draft
802.1Qcc	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements	Draft
802.1Qch	Cyclic Queuing and Forwarding	Draft
802.1Qci	Per-Stream Filtering and Policing	Draft
802.1Qcj	Automatic Attachment to Provider Backbone Bridging (PBB) services	Draft
802.1CM	Time-Sensitive Networking for Fronthaul	Draft
802.1Qcp	YANG Data Model	Draft
802.1Qcr	Asynchronous Traffic Shaping	Draft

Table 2.1.: Overview of standardization activity inside TSN [160, 159]

is the successor/extension of the Audio Video Bridging (AVB) standard. TSN consists of twelve (planned) standards (see Table 2.1). However, currently there only three of them are published. The published standards are IEEE 802.1Qbu, IEEE 802.1Qbv and IEEE 802.1Qca. IEEE 802.1Qbu and IEEE 802.1Qbv are mainly targeting enhancements of the forwarding plane, while IEEE 802.1Qca tackles the control plane. We provide a short overview and a discussion of the applicability in this work for the three published standards:

802.1Qbu standardizes the way frame preemption could be implemented. Frame preemption would enable to stop the transmission of one Ethernet frame to the favor of an other. Since this feature increases the Generalized Processor Sharing (GPS) character of an Ethernet link, it becomes highly relevant feature to achieve good QoS performance. GPS based QoS measures [151, 150] could benefit from less pessimistic assumptions and provide a better resource usage.

802.1Qbv standardizes a new scheduling mechanism. It is manly based on a accurate time synchronization between all the network nodes done with 802.1AS-Rev. The synchronized time base is used to control gates which could prevent queues to forward their packets. With this function it is possible to build up TDMA like scheduling schemes, which could provide a better resource usage in QoS scenarios.

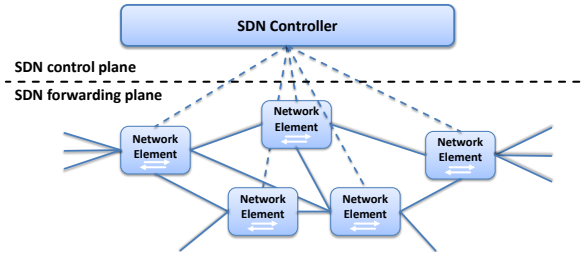


Figure 2.1.: The concept of network function split in Software Defined Networking

802.1Qca standardizes a new control protocol which allows to specify an explicit path trough the network. This path is provided by a Path Computation Element (PCE). By doing this, the standard implements a centralized management interface for TSN.

However, since the TSN standards are not finalized now we decide not to rely on beneficial technology like IEEE 802.1Qbv and IEEE 802.1Qbu. Since there is no hardware available the use of TSN would even prevent the solution of Problem 5: Commodity Hardware.

2.1.4. Software Defined Networking (SDN)

SDN is a new paradigm to increase the flexibility of networks. It introduces a standardized interface to access the forwarding plane of network nodes remotely. The OpenFlow protocol [125] is one candidate standard technology realizing SDN. The principle idea of SDN is to split a router or network switch into a forwarding plane and a control plane (Figure 2.1). The forwarding plane is located inside the network and is able to provide basic forwarding functionality such as classification, marking, metering and scheduling. The control plane is located on one or more servers and manages the forwarding plane remotely. This split changes the way in which network functionality can be designed. In legacy networks, all algorithms have a decentralized character. With SDN, control functions have a centralized view and can be implemented in software in a centralized manner. In summary, the key features of SDN are flexibility, a centralized view and programmability [125, 141, 22].
SDN could be used to realize various QoS scenarios. Policy Cop [19], for example, implements IntServ and DiffServ like functionality using the SDN paradigm. This functionality is managed by a policy-based management system. VSDN [128] and other QoS routing frameworks [142] follow a centralized routing approach to improve video experience in a network. The disadvantage of the current state of the art is that these new SDN applications implement only those QoS management functions that are already available. These projects strongly

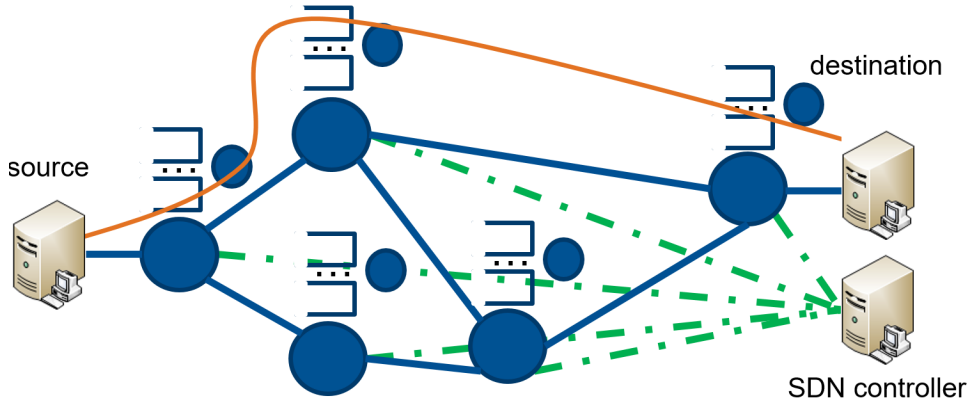


Figure 2.2.: Overview of the industrial QoS framework concept

correspond to the IntServ and DiffServ approach, with the advantage of having a standardized interface to the network hardware. However, all these SDN based QoS schemes do not provide a proper solution for Problem 1: Deterministic end-to-end delay guarantees.
We will show in this thesis that it is possible to implement a deterministic real-time communication system with SDN. This is also a contribution to SDN itself, because it demonstrates the flexibility by addressing a use-case that is not feasible with standard network hardware.

2.2. Putting It All Together

Software Defined Networking (SDN) provides fine-granular access to the forwarding (data) plane through an SDN controller (see Figure 2.2). The SDN controller allows to specify for each flow on each hop which queue should be used. This fine-granular forwarding plane control can serve as the basis for the operation of real-time QoS communication mechanisms, including admission control, scheduling, and resource allocation mechanisms. Specifically, for achieving industrial real-time communication based on SDN it is first necessary to define a model for all forwarding elements in the network. This network model creates an abstraction layer between SDN hardware and SDN applications. This abstraction should allow for the development of QoS-aware SDN applications independently from the network hardware.
We employ deterministic network calculus [118] to bound the maximum (worst-case) delay per queue (see Figure 2.2). The deterministic network calculus model for the worst-case delay allows for relatively easy admission control. In particular, due to the allocation of worst-case delay budgets to the priority queues , the models of the different priority queues at a given link (hop) are

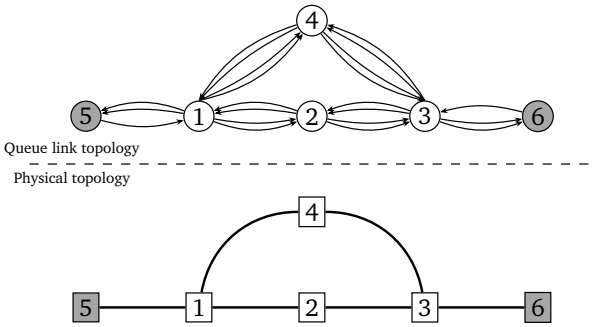


Figure 2.3.: Illustration of the queue link topology concept: A queue link models the outgoing queue for an actual physical link. For instance, the bi-directional physical link between nodes 2 and 3 has two distinct queues in each direction; correspondingly there are two queue links from node 2 to node 3 and two queue links from node 3 to node 2.

independent. Thus, admission decisions of low-priority queues do *not* have to be recalculated every time a flow is added to a high-priority queue.

We employ non-preemptive static priority scheduling, which transmits a packet from a given priority class (queue) only if all higher priority queues are empty, but does not interrupt an ongoing packet transmission. Static priority scheduling is simple as it does not not require any parameter configuration beyond the number of priority queues, yet supports a broad range of delay bounds. We assume for this Thesis that the (computational) processing delay of each forwarding element (hence simple forwarding comes with processing delay around $10^{-6}s$ [156]) is negligible.

Along its path, a packet suffers from different types of delays [39]: processing, queuing, transmission and propagation delays. Since the link characteristics are assumed to be known, the propagation delay for each link is known. Upper bounds on the queuing and transmission delays can, for their part, be computed using the network calculus results presented in Sec. 4.2. The sum of all these components along the route a flow is following makes up the total deterministic end-to-end worst-case delay bound for the flow.

Obviously, the (queuing) delay a packet will experience on its way to its destination does not only depend on the path the packet follows but also on how the packet is scheduled at each output link.

From this, the route selection process for a flow must consider both the physical links the flow will traverse and the queues at which the flow will be buffered at each output link. As a consequence, we [2, 3] introduced a *queue link* network topology. This is illustrated in Fig. 2.3. From the physical network topology, each directed physical link (u, v) is replaced by $|\mathcal{E}^q_{(u,v)}|$ queue links, where u and v are the source and destination nodes of the link and $|\mathcal{E}^q_{(u,v)}|$ is the number of priority queues at the scheduler of the link. The outgoing links of end hosts are

assumed to have only one priority queue and are hence left unchanged. Each link in the queue link network topology hence represents a physical link and a given queue at the ingress of this physical link, i.e., a different QoS level of transmission over this physical link. Route selection on this queue link network thus determines both the path that a flow takes through the physical network as well as the queue in which the flow will be buffered at each physical link.
Performing route selection on the queue link topology allows a flow to be assigned different priorities at each node, thereby increasing flexibility compared to other legacy [115] and SDN [18, 17, 113] approaches which usually assign fixed priorities to flows along their complete path. However, the amount of edges of the graph on which route selection is performed is highly increased, thereby slowing down the routing procedure.
One benefit of using Ethernet for guaranteeing industrial-grade QoS is the interoperability with other IP networks such as a company's office network or the Internet itself. The traffic exchanged with these networks might not have such QoS requirements as the industrial traffic. The lowest priority queue of each link can be used for serving this so-called *best-effort* traffic. In this manner, the real-time traffic, which is only flowing through the higher priority queues, is not influenced by the best-effort traffic which is then only allowed to use resources which are left unused by the real-time flows.
Since best-effort traffic is allocated to a single queue at each link, the corresponding queue link topology is the same as the physical topology and the best-effort traffic can hence be routed using traditional SDN controller modules for routing (e.g., layer-two learning switch).

2.3. Formal Problem Formulation

The main goal of this thesis is to provide a deterministic real time communication service which can handle communication requests online. A deterministic real time communication service provides strict guarantees to the end-to-end delay. A accepted flow will never exceed its delay bound. The terminology of "online" describes that the set of requests is not known in advance and can change over the runtime of the system. Such kind of a system is similar to the traditional telecommunication systems.
The network is represented by $\mathcal{G}$ representing a graph consisting of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$). In addition there is $\mathcal{F}$ which represents a set of flows (f) embedded in the Graph $\mathcal{G}$. Please note that $\mathcal{F}$ varies over time due to the online character of the problem. There should be no restrictions on the topology layout. Nevertheless a suboptimal topology could have influence on the system performance. A real time communication request consisting of an origin (o_f),

a destination (d_f) and a deadline constraint (t_f) of a flow (f). To increase the performance we introduce a routing based on queues. For each flow a path P_f has to be selected which is an element of the set of possible paths $\mathcal{P}_{(o,d)}$.

$$P_f \in \mathcal{P}_{(o_f,d_f)} \quad \forall f \in \mathcal{F}. \tag{2.1}$$

While having an end to end delay of a path P_f ($D(P_f)$) smaller or equal the per flow delay constraint t_f.

$$D(P_f) \leq t_f \quad \forall f \in \mathcal{F}. \tag{2.2}$$

In addition the end to end loss probability of a path P_f ($L(P_f)$) should be zero

$$L(P_f) = 0 \quad \forall f \in \mathcal{F}. \tag{2.3}$$

We only count buffer tail drops as source for packet loss. Other sources of packet loss are neglected in this work. The two end-to-end metrics loss and delay can be calculated with Network Calculus. Please note that delay and loss depend on the routing of the set of flows $\mathcal{F}$. If a solution for this problem exists, all flows will experience a deterministic end-to-end connection.
The optimal solution of this problem will lead to the highest possible utilization of the topology by providing the needed QoS. Unfortunately, this problem is NP-hard and has to be recomputed every time a new flow is added to $\mathcal{F}$. As a result of the complexity increase due to the high number of edges in the graph on which route selection is performed, solving the problem using a MIP formulation leads to intractable runtime. Already hundreds of seconds or more are needed to solve the problem for small networks (see. Section 6).

2.4. Function Split

To solve this problem in reasonable time it is necessary to shrink the problem size. The performance of the optimal solution (Section 2.3) suffers most from the fact that for each new flow an optimization over the set of flows $\mathcal{F}$ is done. A greedy solution that covers only the new flow will break down the complexity:

$$\begin{aligned} P_f &\in \mathcal{P}_{(o_f,d_f)} \\ D(P_f) &\leq t_f \\ L(P_f) &= 0 \end{aligned} \tag{2.4}$$

This simple change of Equations 2.1, 2.2 and 2.3 will lead to an algorithm which takes a path out of the set of paths which fulfill the delay and loss constrains.

However, to approximate optimal formulation nicely the path has to be selected which enables a high amount of future path embeddings. To enable this selection a the minimization of a **cost function** as a objective has to be introduced:

$$\begin{aligned} &\min_{P_f \in \mathcal{P}_{(o,d)}} \text{Cost}(P_f) \\ &D(P_f) \leq t_f \\ &L(P_f) = 0. \end{aligned} \tag{2.5}$$

This cost function mainly impacts the overall performance (maximum number of flows) of the greedy algorithm.
Finding a least cost path, which fulfills an end-to-end delay constraint is a well known problem in the state of the art. It is called DCLC problem or more general CSP problem. CSP **routing** algorithm finds the path (P_f) out of the set of paths which leads from origin to destination ($\mathcal{P}_{(o,d)}$) with the smallest end-to-end costs ($\text{Cost}(P_f)$)

$$\min_{P_f \in \mathcal{P}_{(o,d)}} \text{Cost}(P_f), \tag{2.6}$$

while not violating the deadline (t_f)

$$D(P_f) \leq t_f. \tag{2.7}$$

This algorithm could serve each routing request successively.

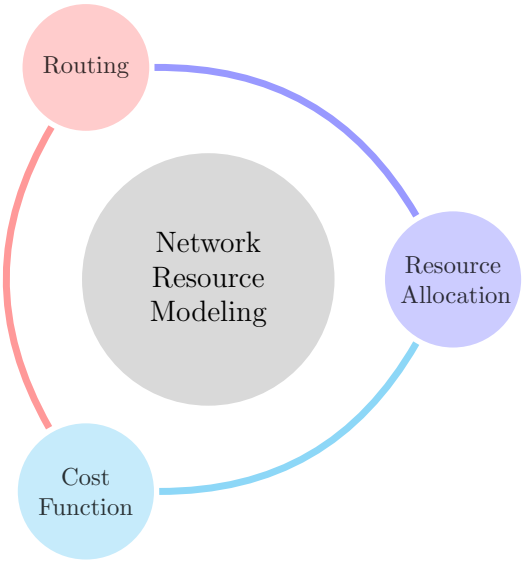


Figure 2.4.: dependencies of Delay-Constrained Least-Cost (DCLC) Routing, Resource Allocation and Cost Function

The greedy routing approach needs a static delay for each edge. If the edge delay would change because of the embedding of a new flow, the other flows might violate their deadline. This will cause a new routing request which tries to find a new valid path. Such a behavior will lead to a potential unstable system. The **network resource model** has to cover this demand. The only way which allows a static delay is to preassign resources to the edges and only allow new flows if these resources are available. By doing this a delay calculation is possible.
Unfortunately, this preassignment is not optimal for each situation in the network. The goal of the **resource allocation** is to adapt the preassignment of the resources to maximize the amount of flows the system can carry.
In summary, a split of the problem defined in Section 2.3 has to be implemented. In total, we divide the overall problem into four parts as shown in figure 2.4. These parts are the routing, the cost function the resource allocation and the network resource modeling. The routing algorithm depends on the cost function. The cost function depends on the resource allocation. And the resource allocation needs the input form the routing algorithm. All three functions need an underling network resource model.

2.5. Building Blocks for Implementing the Function Split

As a result of this online approach, QoS routing is initiated by a query of the data plane (*new flow* arrow in Fig. 2.5). This can be done by contacting the Northbound Interface (NBI) of the SDN controller or by means of a *PACKET_IN* OpenFlow message [126]. The query should at least contain the flow characteristics (e.g., in our case, source, destination(s), burst, rate and maximum packet size) and QoS requirements (e.g., in our case, maximum delay). In case of queries coming from *PACKET_IN* messages, these parameters can be inferred from the packet header (port numbers, transport protocol, etc.). Based on this input and on the current state of the network (*get* arrow in left diagram of Fig. 2.5) routing can then be performed. When this is done, the corresponding OpenFlow rules are pushed to the data plane (*embedding* arrow in left diagram of Fig. 2.5).
When a route is found by the routing algorithm, it updates (right *update* arrow in Fig. 2.5) the state variables through the model in order to reflect the new flow embedding. The guarantees provided to previous flows do not have to be checked because the routing algorithm made sure that they will not be violated by checking the availability of resources at each queue link it followed. The idea is illustrated by the two independent loops in Fig. 2.5. The resource allocation algorithm runs continuously in the background, trying to allocate resources to

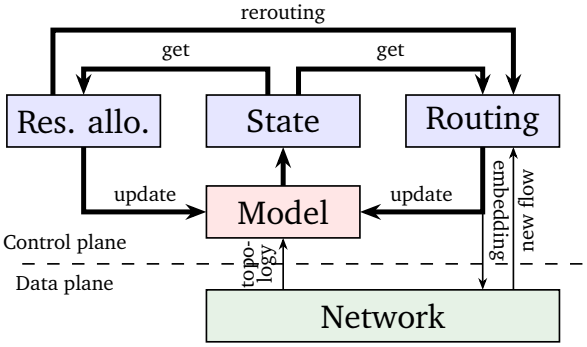


Figure 2.5.: Solving the problem by splitting it into two independent subproblems: resource allocation and routing. The resource allocation algorithm (left loop) continuously optimizes the allocation of resources to each queue and the routing algorithm (right loop) finds a route with enough resources for each incoming flow request. This process requires no interaction with the data plane.

queues (left *update* arrow in Fig. 2.5) in a way that maximizes the number of flows that can still be embedded, while ensuring that the guarantees of already embedded flows can still be satisfied. The routing algorithm, for its part, runs upon the receipt of a flow request (*new flow* arrow in Fig. 2.5) as well as upon an update of the resources allocation by the resource allocation algorithm (*rerouting* arrow in Fig. 2.5), since this might require some flows to be rerouted.
So fare we describe the controller design on a process level. From an information flow perspective the interconnection of the four sub problems gets visible (see Fig. 2.6):

- The *cost function* (Chapter 5.2) transforms the network state, characterized for instance by data rates, buffer consumption, and already embedded flows, into a cost metric for each edge. The cost metric should maximize the number of flows that the network can serve.

- The *resource allocation* (Chapter 5.3) module adapts the allocation of resources (e.g., data rates) to the different queues in order to maximize the total number of flows that can be accepted in the network.

- The *network resource model* (Chapter 4) implements access control and worst-case delay computation based on the resources allocated to the different queues in the network. The network resource model tracks the consumption of resources by the embedded (already accepted) flows. To achieve a deterministic system behavior, the resource model is based on a mathematical traffic model, thus avoiding measurements of the actual network utilization. Since the scheduling delay experienced by a flow at a node depends on the queue at which it is buffered, worst-case delays are computed

per queue. More specifically, the maximum (worst-case) delay per queue is bounded through deterministic network calculus modeling [118, 111]. The resource allocation pre-allocates worst-case delay budgets to the priority queues, which are scheduled according to the non-preemptive static priority policy. This ensures that the models of the different priority queues at a given link (hop) are independent [3]. Thus, admission decisions of low-priority queues do not have to be recalculated every time a flow is added to a high-priority queue.

- The *QoS routing* (Chapter 5.1), e.g., DCLC routing, finds the least-cost path satisfying the end-to-end delay requirement of a connection request. The network resource model in [3] provides delays per queue, thus, routing has to be performed on a so-called *queue-link topology* illustrated in Fig. 2.3, where a given edge (link) of the physical topology is modeled by as many queue links as it has distinct QoS queues. In such a way, routing chooses both the links followed by a flow and the queues at which the flow will be buffered.

The remainder ot this thesis is organized in the flowing way. In Chapter 3 an analysis of the state of the art is provided. Chapter 4 - 5.3 describe each

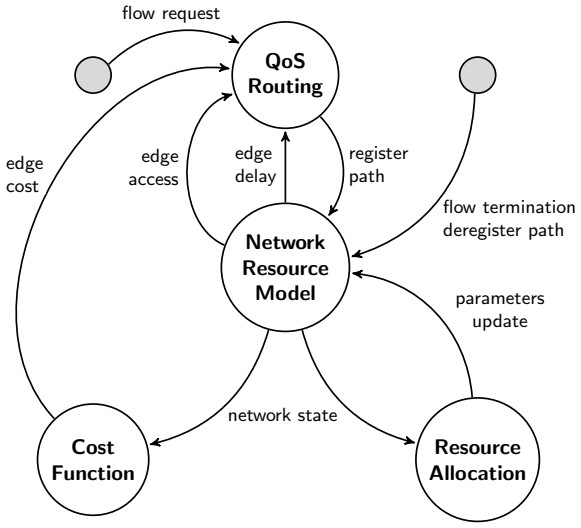


Figure 2.6.: Overview of the different function blocks of QoS networking framework [3]. The *network resource model* provides, through detailed queue modeling, deterministic delay bounds and implements access control that guarantees isolation of flows. The *cost function* indicates which edges should preferentially be used to maximize the probability of future flows to be accepted. The *resource allocation* adapts the allocation of resources to the different queues in order to maximize the probability of future flows to be accepted. *QoS routing* is then responsible for routing incoming requests on a path satisfying the end-to-end delay requirement of the request.

heuristic building block. The *cost function* (Chapter 5.2), the *network resource model* (Chapter 4) and the *resource allocation* (Chapter 5.3) provide as a contribution the implementation of this building blocks. A Evaluation of the interplay of the heuristic building blocks is described in Chapter 6.

3. State of the Art Analysis

This chapter is structured as follows. Section 3.1 explores the existing work of centralized Quality of Service (QoS) frameworks. This state of the art is relevant for the work of this thesis, since providing QoS through a central entity is the abstraction of the goal of this thesis. Therefore, an overview of this topic is provided to point out the novelty of the industrial QoS framework.
The overview of existing work on QoS routing algorithms is provided in Section 3.2. Since the industrial QoS framework enables use of state of the art QoS routing algorithms a survey of those is needed. The goal is to provide a list of algorithms suitable for centralized QoS frameworks. This list could be found at the end of this chapter.

3.1. Centralized QoS Networking Frameworks

The emergence of SDN as a new networking paradigm providing a global view of the network in a centralized control entity provides a new opportunity for smart traffic engineering and QoS provisioning. Hence, a wide range of work has been considering the usage of SDN for QoS networking. In this Section, we present an overview of the state-of-the-art in QoS provisioning using SDN and highlight the contributions of this thesis with respect to the existing literature. We classify the existing approaches in six different categories for which we list a few representative examples.

3.1.1. High-Level Architectural Proposals

Several proposals mainly focus on architectural issues such as interface design, software role modeling, and requirements analysis [131, 16, 15, 128, 114]. These approaches mention that a method for access control and resource reservation is needed but do not tackle this problem. The models we propose in this thesis could be used as part of such frameworks.

3.1.2. OpenFlow Extensions

Other approaches consider the enhancement of the OpenFlow protocol with QoS-related features [123, 40, 128]. Because of the lack of standardization, this

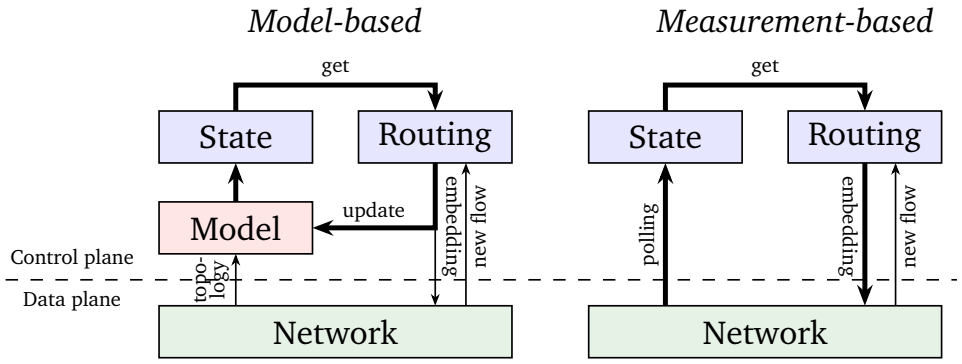


Figure 3.1.: Keeping a model of the network in the control plane allows to avoid the control loop (thick arrows) to go through the data plane, thereby reducing the request processing time.

potentially leads to higher cost and/or effort. In contrast, we propose models which can be used with any SDN protocol providing standard enqueuing and forwarding programming primitives.

3.1.3. TDMA Solutions

Systems making use of time division multiple access (TDMA) on top of Ethernet have also been proposed [133, 134]. These solutions can potentially lead to an optimal utilization of resources. However, because of the need for synchronization, changes in the protocol stack of endpoints might be needed, thereby leading to expensive solutions in terms of cost and effort.

3.1.4. QoS Frameworks based on Data Rate Allocation

Another class of proposals, mainly tailored for Internet QoS, map QoS requirements to equivalent minimum data rates [116, 17, 113, 122]. Such systems typically do not consider the limited capacity of buffers and hence packet loss and queuing delay. These approaches provide the scalability and QoS level needed for wide-area networks but are not sufficient for industrial scenarios.

3.1.5. Measurement-based Frameworks

A wide range of proposals build the network state by retrieving it from the data plane (*polling* arrow in the right diagram of Fig. 3.1) [123, 132, 19, 18, 20, 124, 122]. This step adds a non-negligible delay to the flow request processing. Besides, these approaches suffer from possible measurement errors. Thus, they can only provide soft guarantees. While this is an efficient solution for multimedia traffic, it does not fulfill the requirements of industrial communications.

3.1.6. Model-based Frameworks

The present thesis falls into the category of model-based frameworks where a model of the resources usage is kept in the control plane [116, 113, 120]. The state of the network can then be retrieved from the model itself (see left diagram of Fig. 3.1), avoiding the control loop to go through the data plane, thereby reducing the request processing time. This is illustrated by the thick arrows, representing the QoS control loops for both model-based and measurement-based systems, in Fig. 3.1. The model only has to communicate with the data plane at topology change events (*topology* arrow in left diagram of Fig. 3.1). While stochastic modeling could be used for soft QoS requirements, a deterministic model is needed for providing real-time guarantees. Duan [116] and Tomovic et al. [113] proposed models based on data rate allocation which, as elaborated in Sec. 3.1.4, is not suitable for industrial applications. King et al. [120] detail a deterministic model but which requires a flow embedding procedure that can lead to high request processing time. The Deterministic Services (DetServ) models we propose in this thesis are deterministic models that can be used as part of a model-based QoS framework for fast request processing in industrial scenarios.

3.2. Centralized Routing Algorithms

Given the key importance of routing for communication networks, routing algorithms have been extensively studied for a wide range of network settings and applications. This thesis presents a comprehensive up-to-date survey of unicast QoS routing algorithms within this section. The content of this section is manly based on [4].

3.2.1. Basic Definitions for QoS Routing Algorithms

In this section we first briefly review the main terminology and definitions for unicast QoS routing algorithms and then outline the goals that good QoS routing algorithms should strive for.

3.2.1.1. Definitions and Terminology

Unicast QoS routing refers to the problem of routing a flow from a single source to a single destination so as to fulfill the flow's QoS requirements. Depending on the QoS requirements, different problems can be defined. The most commonly encountered problems are defined as follows and contrasted in Table 3.1.

Table 3.1.: Conceptual comparison of QoS routing problem types.

Problem Type (Acronym)	Number of Optimized Metrics	Number of Constrained Metrics
Shortest Path (SP)	1	0
Constrained Shortest Path (CSP)	1	1
Multi-Constrained Shortest Path (MCSP)	1	M
Multi-Constrained Path (MCP)	0	M

- Shortest Path (SP): The route has to minimize a unique end-to-end QoS metric.
- Constrained Shortest Path (CSP): The route has to minimize an end-to-end QoS metric while keeping another metric below a prescribed bound.
- Multi-Constrained Shortest Path (MCSP): Constrained Shortest Path (CSP) problem with multiple end-to-end metrics that are constrained by individual bounds.
- Multi-Constrained Path (MCP): problem without optimization metric, i.e., the route only has to keep end-to-end QoS metrics below prescribed bounds.

These problems can be extended to k path versions that find k distinct paths. We refer to these extended problems as k Shortest Path (kSP), k Constrained Shortest Path (kCSP), k Multi-Constrained Shortest Path (kMCSP), and k Multi-Constrained Path (kMCP), respectively. It is also possible to define multi-objective problems that optimize more than one metric [37, 63, 35, 38, 36]. These multi-objective problems are beyond the scope of this thesis.
We refer to the metrics that have to be optimized (or minimized) by the routing algorithm as the *costs*. On the other hand, we refer to the metrics that have to be kept below prescribed bounds as the *constraints*. The maximum end-to-end values below which these constraints have to be kept are then referred to as the *constraint bounds*. Note that the term *metric* can refer to either a cost or a constraint. Any metric is always associated to all the individual edges of the network. Depending on how an end-to-end QoS metric is computed from the metric values for individual links, three different categories of end-to-end QoS metrics can be defined: additive, multiplicative, and concave metrics. The

end-to-end values of these three metric categories are, respectively, the sum, the product, and the minimum (or the maximum) of the metric values for the individual links. Delay, packet loss probability, and bandwidth are examples of additive, multiplicative and concave metrics, respectively.
Consider routing to be performed on a network graph $G = \{V, E\}$, whereby V is the set of vertices (network nodes) and E is the set of directed edges (with $|E|$ denoting the number of edges in the network). The vector of costs of the edges is denoted by $\mathsf{c},\ \mathsf{c} \in \mathbb{R}_+^{|E|}$. Let $\mathsf{d},\ \mathsf{d} \in \mathbb{R}_+^{M}$, denote a vector with M elements that represent the bounds for the constrained metrics. Let $\mathsf{D},\ \mathsf{D} \in \mathbb{R}_+^{M\times|E|}$, denote a matrix of the constraint values for the individual edges. Let $P_{sd},\ P_{sd} \subseteq \{0,1\}^{|E|}$, denote the set of paths from source node s to destination node d (whereby a value of 1 for an edge means that the edge belongs to the path). For additive metrics, the Shortest Path (SP), CSP, and Multi-Constrained Shortest Path (MCSP) problems can be mathematically formulated as:

$$z_{\text{opt}} = \min_{\mathsf{x}\in P_{sd}} \ \mathsf{c}^{\mathsf{T}}\mathsf{x} \tag{3.1}$$

$$\text{s.t.} \quad \mathsf{D}\mathsf{x} \leq \mathsf{d}. \tag{3.2}$$

The SP, CSP, and MCSP problems correspond to the cases $M = 0$, $M = 1$, and $M > 1$, respectively.
An *optimal* algorithm is an algorithm that always finds the optimal path with cost z_{opt}. A *heuristic* is an algorithm that finds a possibly sub-optimal path, i.e., a path with cost $z' \geq z_{\text{opt}}$. The Cost Inefficiency (CI) of an algorithm, measured in %, is defined as

$$\text{CI} = \frac{z' - z_{\text{opt}}}{z_{\text{opt}}} \times 100. \tag{3.3}$$

An optimal algorithm therefore always has a CI of 0 %. An algorithm is said to be *complete* if it always finds a feasible solution if one exists. Completeness does not imply optimality.
QoS networking contexts (including the QoS networking framework [3], see Section 2) typically require the QoS routing algorithm to find a least-cost path satisfying an end-to-end delay constraint. This corresponds to a CSP problem with two additive metrics. This subset of CSP problems is also commonly referred to as Delay-Constrained Least-Cost (DCLC) routing problem. For this reason, we will often refer to the optimized QoS metric as *cost* and to the constrained metric as *delay*. The routing algorithm for QoS networking has to be complete. Indeed, if a connection request can actually be accommodated in the network, then the request should not be rejected.
In this thesis, we survey existing unicast CSP routing algorithms for additive metrics. Moreover, since MCSPalgorithms can be used for solving CSP problems, we also present MCSPalgorithms. While Multi-Constrained Path (MCP)

algorithms can find feasible solutions for CSP, MCP algorithms do not optimize the cost metric and we therefore do not consider MCP algorithms in this survey.

3.2.1.2. Goals of QoS Routing

We proceed to summarize the key goals of a good QoS routing algorithm for centralized network control:

- The algorithm should be complete. Indeed, we do not want to reject a connection request if it can actually be accepted.
- Generally, the Delay-Constrained Least-Cost (DCLC) problem is NP complete [137]. Therefore, there is a fundamental trade-off between cost optimality and low runtime. Thus, a QoS routing algorithm should achieve a short runtime as well as a good cost optimality. Indeed, since routing is triggered upon receipt of a connection request and cost minimization leads to a network that can accept more flows, both short runtime and good cost optimality are important for good and fast request handling.
- The algorithm should be able to accommodate real values for the metrics and should not be based on value space reductions. Algorithms that only accommodate integer values may incur quantization errors and are outside the scope of this study.
- The hop count should not be considered for optimization. Indeed, we are primarily interested in low resource usage, which is completely represented by the cost function.
- The algorithm has exact up-to-date knowledge of the state of the network, which can be readily acquired with the centralized network control.
- There is no relationship between cost and delay.
- Cost and delay values may change during the runtime of the real system. Thus, results of computations for prior QoS routing runs, e.g., SP trees, cannot be stored and re-used for future QoS routing runs.
- The constraint must be guaranteed by the algorithm. We strive for strict requirements. Soft constraints that may be violated with a small probability are an interesting direction for future work.
- The connections are unicast connections. Multicast is outside the scope of this survey.

3.2.2. Overview of Shortest Path (SP) Algorithms

DCLC algorithms often make use of underlying SP and kSP algorithm mechanisms; therefore, we briefly review SP and kSP algorithms in this section and in Section 3.2.3. Shortest path algorithms have been studied for a long time and the best algorithms are now well-known [77]. The ***Dijkstra algorithm*** [72] is a centralized algorithm that computes the SP from a single source node to all other nodes (i.e., an SP tree) in a graph with non-negative edge costs.

The Dijkstra algorithm is a priority queue based algorithm. That is, it maintains a queue containing a set of partial paths, i.e., paths starting from the source node and reaching an intermediate destination node which is not the ultimate destination. At each iteration, it takes the least-cost path among the paths in the queue and generates n new paths by extending this partial path with the n outgoing edges of the node at which the given path terminates. Among those paths, only paths with lower cost than the current least-cost path in the queue towards the same destination are added back to the queue. That is, the Dijkstra algorithm *relaxes* based on the cost values. In other words, the Dijkstra algorithm performs a breadth-first search and maintains the current best path found to each destination node. Nodes with least-cost distance from the source node are expanded first, thereby ensuring that any node has to be visited only once.

The **Bellman-Ford (BF) algorithm** [74, 41, 75, 76, 73] is a distributed algorithm that computes an SP tree in a graph, including graphs with negative edge costs. The algorithm maintains the current best path found to each node and runs $|V| - 1$ (where $|V|$ is the number of nodes in the network) iterations updating, for each node, the current best path to all neighbor nodes based on the current best path to the presently considered node. Since the path to any node is at most $|V| - 1$ hops long, all SPs will eventually be found. Note that, in the case of a centralized implementation, if an iteration yields no update, the algorithm can be immediately terminated, as subsequent iterations will not lead to any change. Also, as proposed by Yen [79] for centralized implementations, if the cost of the current best path to a node has not changed since the last iteration, then the outgoing edges of this node can be skipped since they will not lead to any new changes.

Both the Dijkstra and the BF algorithms can be used for finding the SP to a single destination. In such a case, the Dijkstra algorithm can be stopped as soon as the destination node is reached. In contrast, the BF algorithm cannot be stopped earlier than in the SP tree case (when SPs to all network nodes are found). Both algorithms can be adapted to compute the SP from any node to a single destination. These versions are called the *Reverse Dijkstra* and *Reverse Bellman-Ford* algorithms, respectively, and are simply obtained by considering incoming edges rather than outgoing edges when going from one node to the

next node(s).
Hart et al. [78] proposed an improvement to the Dijkstra algorithm, the ***A* algorithm***, for finding a single-destination SP by introducing a so-called *guess function*. At each node, this guess function provides a guess for the cost of the SP from this node to the destination node. Paths out of the priority queue with least *projected cost* (i.e., sum of the current cost to the last node of the path and of the guess value at this node) are expanded first. To ensure the correctness and optimality of the A* algorithm, the guess values have to be lower than the real values. The closer the guess values are to the real values, the faster the A* algorithm will reach the destination. At one extreme, the A* algorithm with an exact guess function will directly traverse the SP to the destination. At the other extreme, the A* algorithm with a guess function of zero corresponds to the original Dijkstra algorithm. The overhead introduced by computing the guess function and the benefit of this guess function constitute the trade-off introduced by the A* algorithm. A straightforward guess function corresponds to the least-hop count multiplied by the cost of the least-cost edge. Such a guess function has to be recomputed upon any topology change. In our evaluations, we do not consider topology changes. Thus, the guess function can be computed offline, ensuring that the A* algorithm is, in any case, at least as fast as the Dijkstra algorithm.
We note that some more complex improvements for centralized implementations of the Bellman-Ford algorithm exist, e.g., [79, 80]. However, the centralized Dijkstra algorithm performs generally better for finding an SP tree than distributed algorithms [108]. Similarly, the A* algorithm performs generally better than distributed algorithms for finding an SP.
Therefore, we only consider the Dijkstra algorithm for finding an SP tree and the A* algorithm for finding an SP as underlying algorithms for the QoS routing algorithms in Table 3.2. A detailed quantitative comparison that includes the BF algorithm, its other improvements, SP heuristics [81], and other A* guess functions [82] are left for future research.

3.2.3. Overview of k Shortest Path (kSP) Algorithms

A very well known kSP algorithm, which is also one of the initial proposals for the kSP problem, is ***Yen's algorithm*** [83]. Yen's algorithm consists of two main parts. First, the SP is found using a traditional SP algorithm. Then, subsequent SPs are found based on the knowledge of this initial path. The $(k+1)$th SP is found by starting at intermediate nodes of previously found paths, blocking the next edge forces the algorithm to find another path, and running an SP algorithm from there. The Least-Cost (LC) path out of all these new paths is the $(k+1)$th SP.

Yen's algorithm does not need to know the value of k when starting. We refer to this type of kSP algorithms as Iterative k Shortest Path (IkSP) algorithms. In contrast, ***Chong's algorithm*** [87] requires k to be known in advance. The algorithm is then identical to the Dijkstra (or A* in our case) algorithm, but keeps, at each node, the current k best paths found. Once the destination(s) has (have) been visited k times, the algorithm can stop. We refer to kSP algorithms which have to know the value of k in advance as Static k Shortest Path (SkSP) algorithms. Note that any IkSP algorithm can also be used as a SkSP algorithm. We will see that for dense topologies with many edges (e.g., queue-link topologies with an edge for each outgoing QoS queue, see Section 2.2), algorithms using an underlying IkSP algorithm have poor performance. Indeed, while they could possibly perform well for sparse topologies, the high number of edges in dense topologies increases the number of paths that have to be traversed to reach the desired optimality. Consequently, we only consider Yen's algorithm as IkSP algorithm. The study of the possible performance increase introduced by the usage of other IkSP algorithms, e.g., [45, 84, 85, 86], is left for future work. We are not aware of SkSP algorithms other than Chong's, and will therefore only consider Chong's algorithm as an SkSP algorithm.

3.2.4. Survey of (Multi-)Constrained Shortest Path (CSP and MCSP) Algorithms

This section provides a comprehensive up-to-date survey of Constrained Shortest Path (CSP) and Multi-Constrained Shortest Path (MCSP) algorithms which can be employed for QoS routing. We categorize these QoS routing algorithms according to the underlying algorithm strategy into five main categories: *1)* elementary algorithms, *2)* algorithms based on a priority queue, *3)* algorithms based on BF, *4)* algorithms making use of the Lagrange relaxation optimization technique, as well as *5)* algorithms making use of the knowledge of the Least–Cost (LC) and Least-Delay (LD) paths in the network. These five main categories of QoS routing algorithms are summarized in Table 3.2.

3.2.4.1. Elementary Algorithms

Joksch [57] provided the initial Integer Linear Programming (ILP) formulation of the CSP problem along with a proposal to solve it optimally using dynamic programming. Unfortunately, dynamic programming techniques can only be used with integers. The algorithm is hence not suitable for real valued costs and delays.
Aneja et al. [56] presented an optimal solution for the MCSP problem. Their

algorithm performs pre-processing and is based on an implicit enumeration of all possible paths and is therefore computationally complex.
An elementary algorithm to find a feasible solution to the DCLC problem is to return the LD path, which can be found with a single SP algorithm run. We will refer to this algorithm as the Least Delay Path (LDP) algorithm. The LDP algorithm does not consider the cost parameter; thus, the cost inefficiency may be high.
A way to take the cost into account while still keeping the algorithm simple has been proposed by Lee et al. [98] as the **Fallback (FB)** algorithm. The FB algorithm first computes the Least-Cost (LC) path using an SP algorithm and checks if it is feasible. If yes, then it can be returned. If not, then the LD path is computed and returned. The algorithm can be extended for solving the MCSP problem by running the SP algorithm successively with the different metrics (first with the cost and then with the different constraints) as cost until a feasible path is found. While the algorithm is complete for the CSP problem, it is not anymore for the MCSP problem. Indeed, for the CSP problem, running an SP algorithm with the constraint as cost will ensure finding a path satisfying the bound of this constraint (if one exists). On the other hand, for the MCSP problem, minimizing one of the constraints does not ensure that the other constraint bounds will be met.
Another simple idea utilizes LC paths as follows. Rather than switching to the LD path if the LC path is not feasible, search for the subsequent LC paths (using an IkSP algorithm) until a feasible path is found. Such an algorithm can also be applied for the MCSP problem and, since it discovers paths in order of increasing cost, is optimal in both cases. We will refer to this algorithm as the k Multi-Constrained Shortest Path (kMCSP) algorithm. Obviously, by continuing its search after finding the first feasible path, this algorithm is also able to solve the kCSPand kMCSPproblems.

3.2.4.2. Algorithms Based on a Priority Queue

A widely considered algorithm for optimally solving the CSP problem is due to Widyono [129], who proposed the Constrained Bellman-Ford (CBF) algorithm. Despite its name, the algorithm is not similar to the original BF algorithm. CBF performs a breadth-first search. While keeping track of the LC path to each visited node, CBF discovers paths in increasing order of delay, stopping once the constraint is violated. As the algorithm is actually an extension of the Dijkstra algorithm, it is also sometimes referred to as the Constrained Dijkstra (CD) algorithm. Indeed, similar to the Dijkstra algorithm, the CD algorithm is based on a priority queue and relaxes based on the cost values. However, paths are retrieved from the priority queue in increasing value of delay, instead of cost.

The discovery process can stop when the delay of the paths to further discover is higher than the deadline, since then no additional feasible paths can be found. Since the relaxation is done based on the cost, the LC path with delay lower than the deadline was found, i.e., the algorithm is optimal.
Liu and Ramakrishnan [89] proposed the ***A*Prune*** algorithm for solving the MCSP problem. As its name suggests, the A*Prune algorithm is in principle similar to the A* algorithm (see Section 3.2.2). The A*Prune algorithm assumes that a guess function is available for each metric (i.e., for the cost and all the constraints), discovers paths (i.e., takes paths out of its priority queue) by increasing value of projected cost (see Section 3.2.2), and prunes (i.e., removes from the set of paths to further extend) those paths for which a projected constraint value exceeds the corresponding end-to-end bound. Once the destination node is reached, the MCSP has been found. Note that, unlike the Dijkstra, A*, and CBF algorithms, the A*Prune algorithm does not keep a single path per node. In other words, its way of reducing the number of paths to further extend is not based on the destination node of these paths but on their projected constraint values. The A*Prune algorithm has the additional feature of being able to solve the kMCSP problem. Indeed, the extension of paths can be continued after the destination has been reached. Once the destination node is reached for the kth time, the optimal kth MCSP has been found. The A*Prune algorithm also solves the CSP and MCSP problems optimally.

3.2.4.3. Algorithms Based on Bellman-Ford (BF)

Jia and Varaiya [88] combined the search strategy of the BF algorithm with the Delay-Constrained Unicast Routing (DCUR) algorithm (which is based on LC and LD paths and will be reviewed in Section 3.2.4.5) to define the Delay-Constrained Bellman-Ford (DCBF) algorithm. DCBF first computes a reverse LD tree. Then, it runs the BF algorithm, but updates the best path at a node only if it has a lower cost and if the sum of *(i)* the delay of the path built so far, *(ii)* the delay of the next edge, and *(iii)* the delay of the LD path from the terminal node of the next edge (i.e., the node reached via the next edge) to the destination is lower than the delay bound. We will refer to this test as the *projected delay test*. Jia and Varaiya also propose an extension to this algorithm, k **Delay-Constrained Bellman-Ford (kDCBF)**, keeping track of the k_d best paths for the reverse LD tree run and keeping the best k_c paths at each node for the forward BF run.
Cheng and Ansari [105] proposed the **dual extended Bellman-Ford (DEB)** algorithm, an algorithm similar to FB (see Section 3.2.4.1). Instead of running an SP algorithm for the LC and LD searches, DEB runs a so-called extended Bellman-Ford (EB) algorithm which is able to find, for every possible hop count h, the optimal h-hop constrained path. For both runs, the path considered is

then the best path among all those found.

3.2.4.4. Algorithms Based on the Lagrange Relaxation

Background on Lagrange Relaxation In mathematical optimization, the Lagrange relaxation technique allows to remove some constraints of the original problem and to introduce them in the optimization objective [137, 34, 109]. For example, the Lagrange relaxation of problem (3.1)–(3.2) is

$$L(\mathsf{u}) = \min_{\mathsf{x} \in P_{sd}} \quad \mathsf{c}^\mathsf{T}\mathsf{x} + \mathsf{u}^\mathsf{T}(\mathsf{D}\mathsf{x} - \mathsf{d}), \tag{3.4}$$

where $\mathsf{u} \in \mathbb{R}_+^M$ is called the *Langrangian multiplier*. The minimized function is called the *Lagrange function* of path x and is also denoted as $L(\mathsf{u}, \mathsf{x})$. It can be shown that, if the original problem is feasible, then there is an optimal solution to

$$z_L = \max_{\mathsf{u} \in \mathbb{R}_+^M} L(\mathsf{u}), \tag{3.5}$$

i.e., a feasible solution of the original problem. Problem (3.5) is referred to as the Lagrangian *dual* of the original problem (3.1)–(3.2), which is then referred to as the *primal* problem. Because solving the dual problem does not necessarily optimally solve the primal problem, we say that there is a *duality gap*.

LARAC without and with Gap Closing (GC) Solving the dual problem requires to solve the relaxed problem (3.4) several times. The interesting aspect of this procedure is that, for the CSP problem, the relaxed problem corresponds to an SP problem with a modified cost function $c_u = c + ud$. This concept is illustrated in Fig. 3.2. Each line in Fig. 3.2 corresponds to the Lagrange function of a path in the network. Lines with null or negative slopes correspond to feasible paths while lines with positive slopes correspond to infeasible paths. The intercept of a line corresponds to the cost of the path. In our example, the optimal path (with cost z_{opt}) is highlighted in red. Since $L(\mathsf{u})$ is a piecewise-linear concave function [110], the u value maximizing $L(\mathsf{u})$ can be found using a binary search and always keeping track of a best feasible and a best infeasible paths, starting with the LC and LD paths (shown in blue in Fig. 3.2). As these two paths have slopes of different signs[1], they intersect at a point u_1. This point is then used as the Lagrange multiplier for the next SP run. This run will find a new path. If the path is primal feasible (resp. infeasible), it replaces the current best feasible (resp. infeasible) path. The new pair of best feasible and infeasible

[1] If this is not the case, either the problem is infeasible (both paths have positive slopes) or LC path is optimal and can be returned (both paths have negative slopes).

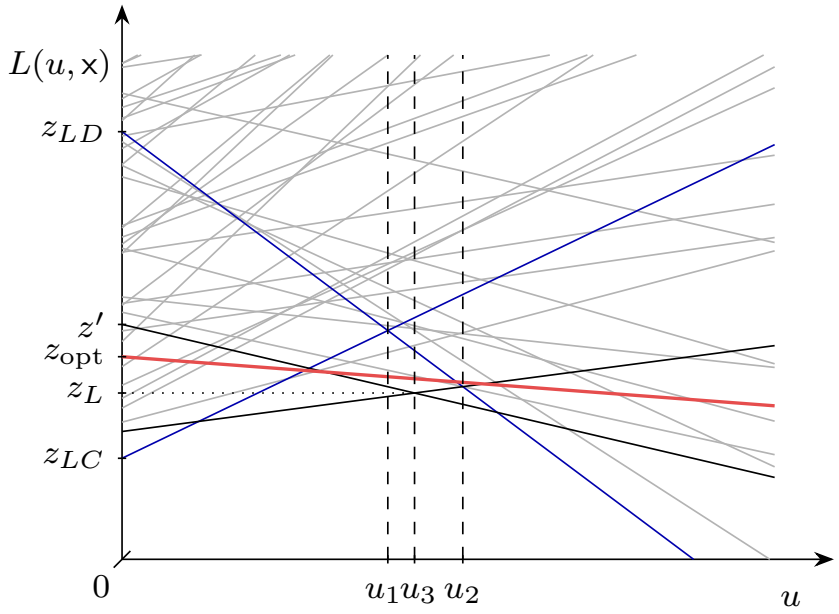


Figure 3.2.: Illustration of Lagrange functions $L(u, \mathrm{x})$ of paths in a network as a function of Lagrange multiplier u. The Lagrange Relaxation based Aggregate Cost (LARAC) algorithm [68, 69, 70, 71] finds the maximum of the lower boundary of this set of curves through a binary search, always keeping track of a best feasible (negative slope) and a best infeasible (positive slope) path. The search starts with the LD and LC paths, found by simple SP searches, and continues with further SP searches with a modified cost function $c_u = c + ud$, where u is obtained as the intersection of the current best feasible and infeasible paths. From mathematical optimization theory, this is an approximation of the optimal solution of the original DCLC problem.

paths defines a new point u_2. The procedure then continues until the Lagrange multiplier does not change. The stored best feasible path at this point is then returned. In the example of Fig. 3.2, we can see that the found path (which corresponds to z' and u_3) is sub-optimal ($z' > z_{\mathrm{opt}}$).

Aneja and Nair [68] initially proposed this algorithm as an optimal algorithm. They did not notice the duality gap. Later, Handler and Zang [69] proposed to close the gap as follows. At the end of the execution of the algorithm [68], the Lagrange value z_L of the found path is a lower bound on the optimal cost z_{opt}. Similarly, the cost z' of this path is an upper bound of the optimal cost z_{opt}. The gap can then be closed by running an IkSP algorithm with the last Lagrange multiplier, i.e., u_3 in our example. Figuratively speaking, the intersections are not relevant for the gap closing; rather, the gap closing traverses the vertical line at u_3 from bottom to top. For each path found by the IkSP algorithm, the upper and lower bounds on the optimal cost are updated. When the lower bound gets greater than the upper bound, i.e., when the Lagrange value of the new path is greater than the cost of the best path found so far, it is ensured that no better path can be found. The best feasible path found so far is then the optimal

solution. In the example of Fig. 3.2, the IkSP algorithm finds the optimal path as the third SP for u_3 and has to find the fifth SP (which is actually the LD path) to notice that no better path exists.
Handler and Zang [69] also introduced a parameter δ to stop the gap closing when the relative distance between the lower and upper bounds is less than δ. As this relative distance is an upper bound on the CI, the δ parameter allows to ensure that the cost inefficiency of the algorithm is always lower than δ. Blokh and Gutin [70] also proposed the algorithm without gap closing. Finally, Jüttner et al. [71] proposed again the same algorithm (without gap closing) and gave it a name: LARAC. They also introduced a *maximal difference* (MD) parameter. The binary search is then stopped when the relative distance between the cost of the current best feasible path and the cost of the current best infeasible path is less than MD. We will refer to the LARAC algorithm with gap closing as **Lagrange Relaxation based Aggregate Cost Gap Closing (LARACGC)**.

LARAC Variations and Extensions Santos et al. [100] proposed an algorithm similar to LARACGC. The difference is that, after having computed the LC and LD paths, Santos et al. directly close the gap without performing the binary search. In particular, Santos et al. use a specific Lagrange multiplier computed based on the knowledge of the delay bound as well as the costs and delays of the LC and LD paths. From the name of its authors, we will refer to this algorithm as **Santos Coutinho-Rodrigues Current (SCRC)**. The algorithm has the same stopping condition as LARACGC and is therefore optimal.
Jia and Varaiya [88] then proposed k **Lagrange Relaxation based Aggregate Cost (**k**LARAC)**, an extension of LARAC that uses a SkSPalgorithm at each iteration, instead of an SP algorithm. The set of k paths found for a given u either contains only feasible paths, only infeasible paths, or a mix of both. As long as only feasible and infeasible sets are found, the new Lagrange multiplier is computed as for LARAC using the LC paths of the two sets. Once a mixed set is found, the LC feasible path of the set is returned. Jia and Varaiya show that the algorithm is always at least as good as LARAC in terms of cost inefficiency. Since k is a parameter of the algorithm, an SkSP algorithm can be used.
The LARAC algorithm can also be visualized in the delay-cost space, see Fig. 3.3, where a point corresponds to a given path in the network. At each iteration, the Lagrange multiplier defines the search direction of the SP run to be perpendicular to the line connecting the current best feasible and infeasible paths. This is shown by the small arrows perpendicular to the solid lines in Fig. 3.3. An SP run in a given direction finds the first point that the corresponding solid line would hit if pushed in this direction starting from point $(0, 0)$.
In the example of Fig. 3.3, LARAC will find the LD and LC paths, then a, then

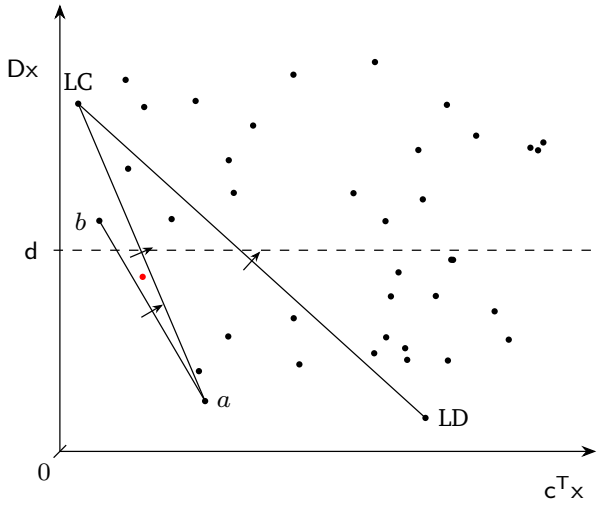
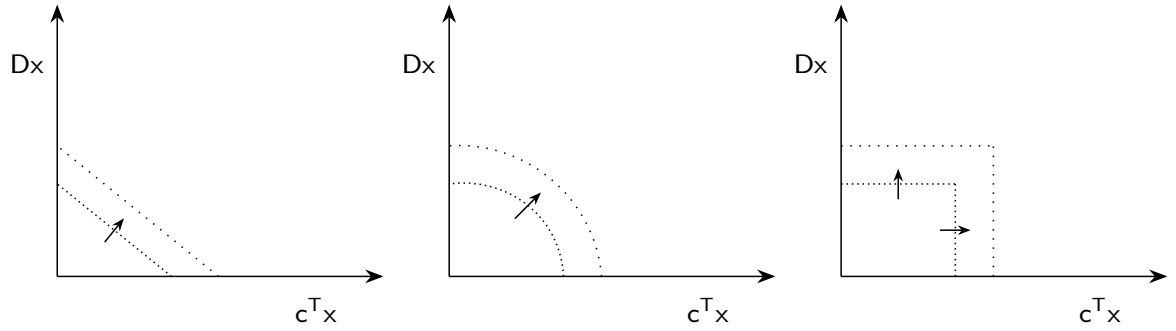


Figure 3.3.: Illustration of operation of LARAC algorithm in the delay (Dx)-cost ($c^T x$) space: At each iteration, LARAC runs an SP search in the direction defined by the Lagrange multiplier. Here, the algorithm will first find LC and LD. Then, based on the Lagrange multiplier computed with these two paths (which corresponds to the normal to the line connecting these two paths), a will be found. Similarly, b will then be found. Then, a or b will again be found, meaning that the Lagrange multiplier will not change. Hence, the algorithm will stop and the best feasible path, i.e., a, will be returned.

b, and the algorithm will then stop and return a. We see that the optimal path, shown in red, is missed. Korkmaz and Krunz [59] argued that this is due to the fact that the search direction is linear in the delay-cost space (Fig. 3.4a). They hence proposed an algorithm called **Heuristic for Multi-Constrained Optimal Path (H_MCOP)** which tries to search simultaneously in the delay and cost directions (Fig. 3.4c). To do so, the algorithm first finds the LD paths from any node to the destination using a reverse SP tree algorithm. Then, it runs an LC forward SP search but updates the best path at a node only when the new path is feasible or has a lower delay than the previously stored best path. The feasibility of the new path is checked using the LD paths stored from the reverse SP tree run. A new path is then considered feasible if it passes the projected delay test. The best path is hence sometimes updated based on the delay and sometimes based on the cost, which is how the algorithm tries to simultaneously follow the search directions shown in Fig. 3.3. The algorithm is nevertheless not optimal because this depends on how fast the delay and cost directions are respectively explored.

As its name suggests, the H_MCOP algorithm is also valid for the MCSP problem. While the explanation of the algorithm for the MCSP problem is more complicated (and is not included because we focus on the CSP problem in this thesis), the algorithm still tries to scan the multi-dimensional cost-constraints



(a) Linear search direction ($\lambda = 1$).

(b) Quadratic search direction ($\lambda = 2$).

(c) Non-linear search direction ($\lambda \to \infty$).

Figure 3.4.: When running an SP (or kSP) algorithm with an aggregated cost, the type of aggregation of the initial metrics influences the search direction of the SP algorithm. In the case of two metrics, linearly combining the metrics leads to the linear search direction shown in Fig. 3.4a. In some cases, including DCLC routing, one may want to explore the delay-cost space simultaneously in both directions. To do so, the initial can be combined in a non-linear fashion. For example, if the aggregated cost is computed by combining each metric to the power of two, a search direction similar to Fig. 3.4b is obtained. By increasing the power used to combine the metrics, one can reach the desired search direction shown in Fig. 3.4c. Unfortunately, once the metrics are not linearly aggregated, the optimal sub-structure property does not hold anymore; thus, classical SP and SkSPalgorithms (e.g., Dijkstra, A* [78], and Yen [79]) are not optimal anymore.

space simultaneously in all directions. However, to do so, an additional parameter λ has to be introduced and is defined as the power value used to combine the original constraints into an aggregated cost. $\lambda = 1$ corresponds to a linear search direction (Fig. 3.4a) and increasing λ towards infinity leads to the search direction shown in Fig. 3.4c. Besides, in the MCSPcase, the algorithm is not complete anymore [59, 107]. Korkmaz and Krunz [59] then also proposed to use Chong's SkSP algorithm for the forward run in order to continue searching in both directions until k paths have been found, thereby possible finding better paths. We will refer to this algorithm as k **Heuristic for Multi-Constrained Optimal Path (**k**H_MCOP)**. Obviously, this does not solve the incompleteness problem of the algorithm in the MCSPcase [59].

H_MCOP can be used to solve the MCP problem by observing the directions of all the constraints simultaneously and returning the first path found [59, 107]. It is then referred to as **Heuristic for Multi-Constrained Path (H_MCP)** and is incomplete [59]. Feng et al. [91] proposed **Nonlinear Relaxation Delay Constraind Least Cost (NR_DCLC)**, an algorithm using H_MCP as underlying MCP algorithm, although NR_DCLC works with any other MCP algorithm. The LC and LD paths are first computed to check for infeasibility or for LC as elementary solution. Then, the cost of the LD path is set as first cost bound and

the delay bound is the one of the original DCLC problem. Then, H_MCP finds an MCP path within these constraints. The cost of the path found is then used as new cost bound. This process is repeated until H_MCP does not find any path. To avoid H_MCP to return the path found at the previous iteration, the cost bound for the next iteration is always set to a value *a little bit* smaller than the actual cost of the found path.
Feng et al. [107] then proposed a variation of NR_DCLC. Instead of running H_MCP with the cost of the LD path as bound, H_MCP is run with the cost of the path found by H_MCOP as bound. As this algorithm improves on the solution found by H_MCOP, the authors refer to it as **Modified Heuristic for Multi-Constrained Optimal Path (MH_MCOP)**. The authors also introduce a parameter to limit the amount of MCP iterations. We will refer to this parameter as H. MH_MCOP can also solve the MCSP problem but, as it is based on H_MCOP, is not complete for it. On the other hand, NR_DCLC cannot solve the MCSP problem. Indeed, it's initial LD search can only deal with one constraint.
Feng et al. [107] additionally proposed an optimal algorithm, **Exact Multi-Constrained Optimal Path (E_MCOP)**, similar to SCRC. E_MCOP first runs *Exact Multi-Constrained Path (E_MCP)*, a complete MCP algorithm. E_MCP first runs an SP search for each constraint. If one of them cannot be met, it terminates. Otherwise, it runs an IkSP algorithm with the sum of the individual constraints, individually divided by the difference between their bound and their least value, as an aggregated cost. E_MCP returns the first feasible path found or stops once a path with an aggregated cost higher than the cost obtained by considering that each constraint reaches its bound is found. If E_MCP found no path, E_MCOP concludes that there is no solution. Otherwise, E_MCOP considers the found path as the current solution, uses its cost to define a cost border and runs an SP search for the cost to find its least value. From these two values, E_MCOP restarts an E_MCP search with the cost added to the aggregated cost (also divided by difference between its bound and its least value). The algorithm returns the current solution once the IkSP algorithm finds no path or when it finds a path with an aggregated cost higher than the cost obtained by considering that each metric reaches its bound. When a path is found by the IkSP algorithm, it replaces the current solution if it is feasible and has a lower cost. This algorithm is optimal for both the CSP and MCSPcases [107].
Guo and Matta [99] elaborated on the idea of using an underlying MCP algorithm in their ***delay-cost-constrained routing*** **(Delay-Cost-Constrained Routing (DCCR))** algorithm. DCCR behaves similarly to NR_DCLC, but runs the MCP algorithm only once. For this to be effective, they use an MCP algorithm that takes the costs of paths into account. This MCP algorithm can hence also be viewed as a DCLC algorithm which needs a cost bound. The algorithm runs an

SP search where the cost of a path is defined as

$$\frac{\text{delay of the path}}{1 - \frac{\text{original cost of the path}}{\text{cost bound}}}, \tag{3.6}$$

which also tries to emulate the simultaneous scan of the delay-cost space in both directions. Nevertheless, with such a function, the cost of a path is not anymore the sum of the cost of its constituting edges and classical SP algorithms cannot solve the problem optimally. Therefore, Chong's algorithm is used to increase the probability of keeping track of good solutions. Instead of using the cost of the LD path as cost bound, Guo and Matta propose to use the cost of the path found by LARAC. This algorithm is then referred to as **Search Space Reduction Delay-Cost-Constrained Routing (SSR+DCCR)**. Since the algorithm improves the solution returned by LARAC, it is closing the duality gap, similarly to LARACGC, but only partially. In order to provide a cost bound, LARAC does not have to run until the end. Therefore, Guo and Matta define a parameter, which we will refer to as L, that limits the number of iterations (i.e., the number of SP runs excluding the LC and LD searches) of the LARAC run.
Agrawal et al. [64] proposed ***E-LARAC***, an extension of LARAC that additionally considers a constraint on the maximum number of hops by using a modified BF subroutine. Because the number of hops is not a constraint in many QoS networking scenarios, we do not consider E-LARAC in our evaluation.
The Lagrange relaxation has also been used by Ribeiro and Minoux [104] for solving the double-sided constrained SP problem, i.e., the problem of finding an SP whose delay (or any other metric) is lower than a upper bound but also greater than a lower bound. We do not consider this double-sided problem in our evaluation.

3.2.4.5. Algorithms Following the LC and LD Paths

Instead of computing a delay-constrained path from SP searches with modified costs, Salama et al. [92, 93] proposed to solve the DCLC problem from the knowledge of the LC and LD trees towards the destination. The algorithm builds the path node by node. At each node, the algorithm chooses between the edge belonging to the LC path towards the destination and the edge belonging to the LD path towards the destination. The LC edge is chosen if it satisfies the projected delay test; otherwise, the LD edge is chosen. It can happen that a loop is created. In such a situation, the algorithm backtracks to a node that chose the LC edge and then chooses the LD edge instead. Salama et al. show that this backtracking ensures the removal of the loop. This algorithm is called DCUR.
Fig. 3.5 shows an example graph with the corresponding LC (red) and LD (yellow) trees towards the destination node d. Starting from the source node s,

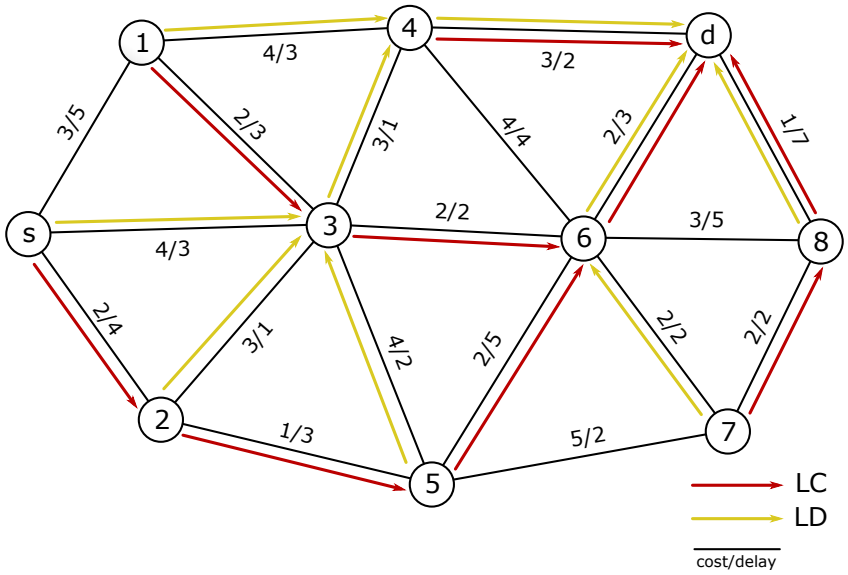


Figure 3.5.: Least-Cost (LC)and Least-Delay (LD) paths from any node to a given destination in an example graph with links denoted by cost/delay. DCUR, DCR, and IAK are algorithms combining these paths in order to find a Delay-Constrained Least-Cost (DCLC). In this example, with a deadline of 10, DCUR, which alternates between using the LC and LD edges, finds the path *s-2-3-6-d* with cost 9 and delay 10. DCR, which follows LD edges and switches once to the LC edges, finds the path *s-3-6-d* with cost 8 and delay 8. Finally, IAK, which follows LC edges and switches once to the LD edges, finds the path *s-2-3-4-d* with cost 11 and delay 8.

DCUR chooses, at each node, between the red and yellow outgoing edges of the node depending on the result of the projected delay test. In this example, with the delay constraint set to 10, DCUR would choose LC-LD- LC- LC (no loop occurs), thereby finding the path *s-2-3-6-d* with cost 9 and delay 10. Indeed, when at node 2, DCUR cannot follow the LC edge. If it did, it would reach node 5 with a delay of 7. Since the LD path from node 5 to the destination has a delay of 5, it would not be possible anymore to reach the destination with a delay lower or equal to 10. Note that this is sub-optimal as the optimal path (that, e.g., CBF would find) in this example is *s-3-6-d* with cost 8 and delay 8.
Sun and Langendörfer [94] then proposed a solution, called **Distributed delay Constrained Routing (DCR)**, to avoid the creation of loops and hence to prevent the algorithm from having to backtrack, thereby reducing runtime. DCR follows the LD path until the sum of *(i)* the delay of the path so far and *(ii)* the delay of the LC path from the current node to the destination is lower or equal to the delay bound. Starting from this point, the algorithm then follows the LC path until the end, since it is ensured that it will satisfy the delay constraint. Since the LD path is only followed from the source node, it can be computed by a simple SP run.
In the example of Fig. 3.5, still with the delay constraint of 10, DCR follows

the LD edge until node *3* as the sum of the delay of the path so far (no path and hence delay of 0) and the delay of the LC path from the current node (s) to the destination (path *s-2-5-6-d* with delay 15) is greater than the delay bound ($0 + 15 > 10$). Indeed, following the LC edges already from node s would lead to an infeasible path. Then, starting from node *3*, DCR switches to the LC path as the sum of the delay of the path so far (path *s-3* with delay 3) and the delay of the LC path from the current node (*3*) to the destination (path *3-6-d* with delay 5) is lower than the delay bound ($3 + 5 < 10$). Indeed, DCR is now sure that following the LC edges until the destination will lead to a feasible solution. Hence, DCR finds the path *s-3-6-d*, which is actually the optimal path. This example shows that, while DCR is simpler than DCUR, it can still provide, in some circumstances, a path closer to optimality.
Ishida et al. [96] then proposed the opposite strategy, i.e., to first follow the LC path and to switch to the LD path as soon as following the LC path would lead to a node from which the delay constraint cannot be satisfied anymore. For the same reason as for DCR, the LC path can be computed by a simple SP run. Based on the author names of [96], we refer to this algorithm as **Ishida Aman Kannari (IAK)**.
In the example of Fig. 3.5, still with the delay constraint of 10, IAK follows the LC edge until node *2* as this edge has a delay of 4 and the LD path from node *2* to the destination has a delay of 4, thereby ensuring that the delay constraint can still be met. From node *2*, for the same reason as for DCUR, IAK cannot follow the LC edge anymore. Indeed, it would not be possible anymore to reach the destination with a delay lower or equal to 10. Hence, IAK switches to the LD edges and finds the path *s-2-3-4-d* with cost 11 and delay 8.
DCUR, DCR, and IAK have been proposed with a distributed implementation in mind and allow only a limited range of choices at each node. Two algorithms have been proposed with the objective of enlarging the set of paths that can be found. First, Sriram et al. [95] proposed that each node maintains a list of ordered preferred output links. When path construction reaches a node, it selects its preferred output link for which the delay of the new path satisfies the delay constraint and that does not introduce a loop. A node may not have a preferred output link that satisfies these constraints. Then, path construction is backtracked to the previous node, which then selects its next preferred output link. If all preferred output links have been exhausted, then path construction is also backtracked to the previous node. Once the destination is reached, the algorithm terminates. Based on the author names of [95] we will refer to this algorithm as **Sriram Manimaran Siva (SMS)**. The list of preferred link is computed according to a *heuristic function*. In order to reduce runtime, the algorithm allows to limit the size of the list of preferred links at each node to a given parameter p. Nevertheless, depending on the heuristic function, this

makes the algorithm incomplete. The algorithm is complete only if p is, at each node, greater or equal to the degree[2] of the node, thereby ensuring that all links are considered. If $\Delta(G)$ denotes the maximum degree of the nodes in graph G, the algorithm is complete if

$$p \geq \Delta(G). \tag{3.7}$$

Sriram et al. [95] define three heuristic functions: *(1)* Residual Delay Maximizing (RDM), ordering links by their cost divided by the delay constraint minus the projected delay of the link (and ensuring that the edges belonging to the LC and LD paths towards the destination are included in the list), *(2)* Cost Delay Product (CDP), ordering links by their cost times their projected delay, and *(3)* Partition-Based Ordering (PBO), ordering links by cost value. RDM requires one SP tree run (LD), CDP two, and PBO none.
If the heuristic function is not efficient, the algorithm could explore an excessive number of paths before reaching the destination. This is especially true for dense topologies with many possible paths. To avoid this, Liu et al. [97] proposed **Selection Function Delay Constraind Least Cost (SF_DCLC)**, an algorithm similar to SMS. At each node, instead of computing a list of links and trying them one after the other, the algorithm chooses one output link based on a *selection function* (SF) which is proven to avoid loops and to lead to a solution if one exists. The links are assigned a weight equal to their cost plus *(i)* the cost of the LC path to the destination if the latter passes the projected delay test, or, if not, *(ii)* the cost of the LD path. Links for which the LD path is infeasible are not considered. The least-weight link is then chosen.

3.2.4.6. Other Approaches

For completeness, we briefly review in this section other QoS routing approaches that we do not include in our evaluation for the various reasons noted for the following algorithms. In order to the reduce the runtime of optimal algorithms, several fully polynomial ϵ-approximation algorithms have been proposed, e.g., [62, 58, 66, 60, 61, 65, 90]. The ϵ-approximation algorithms ensure to find a path whose cost is at most $(1 + \epsilon)$ times higher than the cost of the optimal path. Unfortunately, ϵ-approximation algorithms consider only integral costs and/or delays and are therefore not suitable for QoS routing with real-valued costs and/or delays.
Several algorithms have been proposed to accommodate imprecise state information, e.g., [67, 48, 53, 52, 51, 54]. In centralized network architectures, such as Software Defined Networking (SDN), is it reasonable to assume that the state is

[2]In a graph, the degree of a node corresponds to the number of edges connected to this node. In our scenario, we define the degree of a node as the number of *outgoing* edges the node has.

well-known and we hence do not consider the class of algorithms for imprecise state information. Also, note that algorithms considering imprecise information cannot provide strict (hard) QoS guarantees, rather these algorithms can only provide soft QoS guarantees. Similarly, algorithms based on probing techniques, e.g., [50, 48, 49, 53, 54], or relaxing the constraint, e.g., [55], can also only provide soft QoS guarantees. Our focus is on QoS routing algorithms that can provide strict QoS guarantees and we do therefore not consider the algorithms for imprecise state information, probing, or relaxed constraints in detail in this survey.

Algorithms based on genetic algorithms (GA) [101, 102] and on artificial bee colony optimization techniques [103] have also been proposed. Such randomized algorithms have typically a fairly high runtime and are therefore not well suited for online routing decisions. Our focus is on QoS routing algorithms that are suitable for online routing decisions and we do therefore not cover these algorithms in detail.

Pornavalai et al. [46, 47] simplify the bandwidth-jitter-delay constrained problem into an SP problem with maximum number of hops (i.e., a problem that can be solved in polynomial time) by using relationships between bandwidth, delay, jitter, and buffer capacity in weighted fair queuing (WFQ) set-ups. Our focus is on QoS routing algorithms that accommodate independent optimization and constraint metrics and we do therefore not consider [46, 47] in detail.

Table 3.2.: Comprehensive list of Constrained Shortest Path (CSP) and Multi-Constrained Shortest Path (MCSP) algorithms, which can be employed for Delay-Constrained Least-Cost (DCLC) QoS routing. The algorithms are categorized according to the underlying algorithmic strategy into algorithms based on priority queues, BF, Lagrange relaxation, as well as Least-Cost (LC) and Least-Delay (LD) paths. For each algorithm, we indicate the type(s), i.e., CSP or MCSPor k path versions thereof, as well as other key characteristics, including optimality property and the accepted parameters. We indicate the number of underlying algorithm runs, e.g., Iterative k Shortest Path (IkSP) and Static k Shortest Path (SkSP) algorithms (see Section 3.2.1 for definitions). When the exact number of runs depends on the specific scenario, the possible numbers of runs are indicated through a comma-separated list or a range (with the arrow ($\rightarrow$) symbol) within parentheses. Unbounded numbers of runs are indicated with the greater or equal ($\geq$) sign. We note that an algorithm using a SkSP algorithm can be implemented with an IkSP algorithm.

Algorithm	Type	Number of runs of underlying algorithms				Optimal	Complete	Distr.	Param.
		IkSP	SkSP tree	SkSP SP tree	SP				
Elementary Algorithms (Sec. 3.2.4.1)									
LDP	CSP				1		✓	if SP is	
FB [98]	CSP, MCSP				$(1 \rightarrow M+1)$		if CSP		
kMCSP	(k)CSP, (k)MCSP	1				✓	✓		
Priority Queue Based Algorithms (Sec. 3.2.4.2)									
CBF [129]	CSP					✓	✓		
A*Prune [89]	(k)CSP, (k)MCSP					✓	✓		
Algorithms Based on BF (Sec. 3.2.4.3)									
DCBF [88]	CSP			1			✓	✓	
kDCBF [88]	CSP		1				✓	✓	k_d, k_c
DEB [105]	CSP						✓		
Algorithms Based on the Lagrange Relaxation (Sec. 3.2.4.4)									
LARAC [68, 69, 70, 71]	CSP				≥ 1		✓		MD
LARACGC [69]	CSP	(0, 1)			≥ 1	if $\delta = 0$	✓		δ
SCRC [100]	CSP	(0, 1)			(1, 2)	✓	✓		
kLARAC [88]	CSP		≥ 1				✓		k
H_MCOP [59]	CSP, MCSP			1	(0, 1)		if CSP		λ
kH_MCOP [59]	CSP, MCSP		(0, 1)	1			if CSP		λ, k
NR_DCLC [91]	CSP			≥ 0	≥ 1		✓		
MH_MCOP [107]	CSP, MCSP			$(1 \rightarrow H+1)$	$(0 \rightarrow H+1)$		if CSP		H
E_MCOP [107]	CSP, MCSP	$(0 \rightarrow 2)$			$(1 \rightarrow M+1)$	✓	✓		
DCCR [99]	CSP		(0, 1)		(1, 2)		✓		k
SSR+DCCR [99]	CSP		(0, 1)		$(1 \rightarrow L+2)$		✓		L, k
Algorithms Based on LC and LD Paths (Sec. 3.2.4.5)									
DCUR [92, 93]	CSP			(1, 2)			✓	✓	
DCR [94]	CSP			(0, 1)	1		✓	✓	
IAK [96]	CSP			1	(0, 1)		✓	✓	
SMS-RDM [95]	CSP			1			if $p \geq \Delta(G)$	✓	p
SMS-CDP [95]	CSP			2			if $p \geq \Delta(G)$	✓	p
SMS-PBO [95]	CSP						if $p \geq \Delta(G)$	✓	p
SF_DCLC [97]	CSP			(1, 2)			✓	✓	

4. Network Resource Modeling: Deterministic Services (DetServ)

This chapter emphases the network modeling aspects of this thesis. There are plenty of algorithms, as seen in the state of the art section, capable solving the DCLC or Constrained Shortest Path (CSP) problem. These algorithms provide a foundation for a efficient Quality of Service (QoS) framework. As seen in Figure 4.1 the optimization process covers the cost function and the resources allocation, in addition. All these three sub problems influences the overall performance of the framework.
However, the interfacing to a deterministic network model is not a straightforward task. We describe in section 4.1 the interface requirements and the interface itself. Afterwards we provide a short introduction of Network Calculus (Section 4.2) focusing on modeling priority queuing schedulers. The basic notation and definitions are provided in section 4.3. Based on these notations we discuss the realization of static delay bounds (Section 4.4). This interface is implemented by our deterministic network resource models, the so called DetServ models.

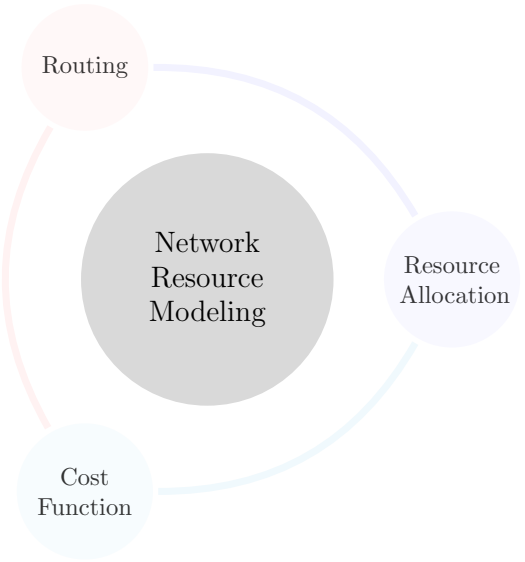


Figure 4.1.: dependencies of Delay-Constrained Least-Cost (DCLC) Routing, Resource Allocation and Cost Function

There are two main modeling opportunities:

- Multi-Hop Model (MHM) - Section 4.5
- Threshold-Based Model (TBM) - Section 4.6

In addition to this we provide models for considering the burst increase (Section 4.7) and Input Link Shaping (ILS) (Section 4.8) for the main models. The content of this chapter was presented in [5].

4.1. Interface of the Network Model

Since we change the routing problem to a greedy solution (i.e. only consider one new flow), the network modeling becomes more challenging. Even if the routing itself could be performed for a single flow (Equation 2.5)

$$\begin{aligned} \min_{P_f \in \mathcal{P}_{(o,d)}} & \text{Cost}(P_f) \\ D(P_f) &\leq t_f \\ L(P_f) &= 0, \end{aligned}$$

The network model has to ensure the original delay (Equation 2.2)

$$D(P_f) \leq t_f \quad \forall f \in \mathcal{F}$$

and loss (Equation 2.3)

$$L(P_f) = 0 \quad \forall f \in \mathcal{F}$$

bounds for all the flows. Therefore, the interface consists of the following four so-called *model functions*.

GETDELAY: computes the worst-case delay of a given queue link edge. This delay value has to be independent of the embedding of new flows, to satisfy Equation 2.2. Indeed if the delay value is static Equation 2.2 is always satisfied.

HASACCESS: checks whether or not there are still enough resources available for a given flow at a given queue link edge. This access control mechanism enables the implementation of the loss constraint (Equation 2.3). Indeed if the resource consumption of each queue in the network is tracked a further use of a queue could be prohibited if this would cause packet loss.

REGISTERPATH: updates the model state to reflect the embedding of a new flow.

DEREGISTERPATH: updates the model state to reflect the removal of a previously embedded flow.

The GETDELAY and HASACCESS methods correspond to the *get* arrow in Fig. 2.5, while REGISTERPATH and DEREGISTERPATH correspond to the *update* arrow. How these methods are implemented depends on how and which resources are allocated and managed at each queue. In the next Sections, we present our two novel DetServ models implementing these four model functions.

4.2. Network Calculus

Network Calculus [118] provides a mathematical framework for calculating deterministic upper delay bounds of a network. This mathematical model has been used for the dimensioning and offline planning of real-time systems [143, 136, 43]. In contrast, we employ network calculus for online modeling of the network at run time. Duan [116] developed an SDN-based network as a service framework and derived network calculus to evaluate data rates needed for fulfilling delay requirements in such a general framework. The network calculus results served as a basis for an offline mixed integer program (MIP) that evaluates admission control (resource availability) decisions and routes flows, albeit for prohibitive computational cost. In contrast, in the present thesis we develop a computationally efficient, online function split framework for run-time admission decisions and routing while maintaining the network calculus based real-time guarantees.

4.2.1. Basics: Theory Principles

In order to provide a deterministic model of the network, we propose to use *network calculus*. Network calculus [118] is a system theory for communication networks. From models of a considered flow and of the service a so-called *system* can offer, bounds on *(i)* the delay the flow will experience traversing the system, *(ii)* the backlog the flow will generate in the system, and *(iii)* the new model for the flow after it has passed the system can be computed. A system can range from a simple queue to a complete network. The theory is divided in two parts: *deterministic* network calculus, providing deterministic bounds, and *stochastic* network calculus, providing bounds following probabilistic distributions. Since we strive for deterministic modeling, we will only consider deterministic network calculus.
As mentioned, models of both a flow and a network system are needed. The modeling of a flow is done using a so-called *arrival curve* α. $\alpha(\tau)$ gives an upper

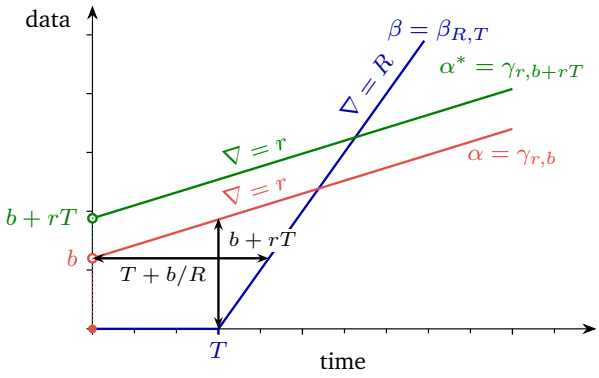


Figure 4.2.: Example of graphical computation of delay, backlog and output bounds using network calculus concepts. The delay and backlog bounds respectively correspond to the horizontal and vertical deviations between the arrival and service curves. In the particular case of an arrival curve $\alpha_{r,b}^{TB}$ and a service curve $\beta_{T,R}^{RL}$, the output bound α^* is obtained by shifting the initial arrival curve α up by rT.

bound on the amount of data a flow will send during any time interval of length τ. The α curve in Fig. 4.2 represents a *token bucket* flow: the flow is allowed to send bursts of up to b bytes but its sustainable rate is limited to r bytes/s. This *Token Bucket (TB) service curve* is denoted by α^{TB}.
The modeling of a network system is, for its part, done using a so-called *service curve* β. Its general interpretation is less trivial than for an arrival curve [135]. The particular service curve β shown in Fig. 4.2 can be interpreted as follows. Data might have to wait *up to* T seconds before being served at a rate of *at least* R bytes/s. This type of service curve is denoted by $\beta_{T,R}^{RL}$ and is referred to as a *Rate Latency (RL) service curve*.
From these two curves, the three above mentioned bounds can be computed. The delay and backlog bounds respectively correspond to the horizontal and vertical deviations between the arrival and service curves [135]. Le Boudec [118] provides the following two functions to compute the horizontal

$$h(\alpha, \beta) = \sup_{t \geq 0} (\alpha - \beta) \tag{4.1}$$

and vertical

$$v(\alpha, \beta) = \sup_{t \geq 0} [\inf (d \geq 0 \text{ such that } \alpha \leq \beta(t + d))] \tag{4.2}$$

deviation of the two curves. This two functions can be improved by only checking the knee points of α and β. Since both curves are piecewise linear functions only the knee points can contain the maximum deviation. This is shown in Fig. 4.2. The vertical deviation is ether T itself or $T + b/R$ which is:

$$v(\alpha, \beta) = \max (T, T + b/R) = T + b/R \tag{4.3}$$

The horizontal deviation is ether b itself or $b + rT$ which is:

$$h(\alpha, \beta) = \max(b, b + rT) = b + rT \tag{4.4}$$

In the general case, the way to compute α^*, the output arrival curve of the flow after having traversed the system, is not straightforward [135]. In the particular case where the arrival and service curves are $\beta^{RL}_{T,R}$ and $\alpha^{TB}_{r,b}$, we have $\alpha^* = \alpha^{TB}_{r,b} + rT = \alpha^{TB}_{r,b+rT}$ [135] (Fig. 4.2). This formula can be interpreted as follows. Since the flow can possibly wait up to T seconds before being served at a potentially infinite rate, its burst size can increase by up to rT bytes – the maximum amount of data that, by definition of the arrival curve of the flow, will arrive during these potential T seconds of waiting time.

4.2.2. Selected Results: Priority Scheduling

In the particular case of a non-preemptive strict priority scheduler with n queues traversed by token bucket flows [111], the service curve for priority queue i is given by [135]

$$\beta_i(t) = \left(Ct - t\sum_{j=1}^{i-1} r_j - \sum_{j=1}^{i-1} b_j - \max_{i+1 \le j \le n} \{l_j^{max}\} - l_i^{max} \right)^+, \tag{4.5}$$

where queue $i = 1$ is the highest priority queue, C is the capacity of the output link, and r_j, b_j and l_j^{max} are the rate, burst size and maximum packet size of the token bucket flow traversing queue j. This formula can be interpreted as follows. The service offered to a given queue i corresponds to the whole link capacity (first term) from which the capacity used by higher priority flows is deducted (second and third terms). Since we assume a non-preemptive priority scheduler, data in a high priority queue might have to wait for a packet of a lower priority queue to be transmitted before being served (fourth term). The fifth term models the store-and-forward behavior of switches. Indeed, the scheduler must wait for each packet to be completely received before serving it. Note that for cut-through switches, only the header length should be used here. Because the scheduler cannot provide negative service, the negative part of the resulting curve is reduced to zero ($(.)^+$ notation).
Eqn. 4.5 corresponds to a $\beta^{RL}_{T_i,R_i}$ curve where

$$T_i = \frac{\sum_{j=1}^{i-1} b_j + \max_{i+1 \le j \le n} \{l_j^{max}\} - l_i^{max}}{C - \sum_{j=1}^{i-1} r_j} \tag{4.6}$$

and

$$R_i = C - \sum_{j=1}^{i-1} r_j. \tag{4.7}$$

From Fig. 4.2, the delay and backlog experienced by the flow traversing queue i are respectively bounded by

$$d_i = \frac{\sum_{j=1}^{i} b_j + \max\limits_{i+1 \leq j \leq n} \{l_j^{max}\} - l_i^{max}}{C - \sum_{j=1}^{i-1} r_j} \tag{4.8}$$

and

$$x_i = b_i + r_i T_i, \tag{4.9}$$

and the new burst of the flow after the system is given by

$$b_i^* = x_i, \tag{4.10}$$

while its rate remains unchanged.

4.3. Notations

As introduced in section 2.2 the resource model consists of two graphs. The physical and queue link graphs are respectively denoted by $\mathcal{G}^p$ and $\mathcal{G}^q$. The physical graph $\mathcal{G}^q$ consists of the set of network nodes $\mathcal{N}$ and the set of physical edges $\mathcal{E}^p$ connecting this nodes. The capacity of a physical link $(u, v) \in \mathcal{E}^p$ is denoted by $R_{(u,v)}$. Queue link edges $(u, v, q) \in \mathcal{E}^q$ are connecting the same nodes $\mathcal{N}$ to form the queue link graph $\mathcal{G}^q = (\mathcal{N}, \mathcal{E}^q)$. We assume a non-preemptive strict priority scheduler with a set of queue links $\mathcal{E}^q_{(u,v)}$ at the physical link $(u, v, q) \in \mathcal{E}^q$. Edges in the queue link network are denoted by (u, v, q), where (u, v) is the corresponding physical link and $\mathrm{p} \in \{1, \ldots, \left|\mathcal{E}^q_{(u,v)}\right|\}$ is the priority of the corresponding queue at the physical link, $\mathrm{p} = 1$ being the highest priority.
The set of active (i.e., embedded) flows in the network is denoted by $\mathcal{F}$. For a given embedded flow $f \in \mathcal{F}$ or for a given flow f requesting an embedding,

- r_f denotes the rate (as defined in Sec. 4.2.1) of the flow,
- $b_f(P_{o_f,u})$ denotes the burst size (as defined in Sec. 4.2.1) of the flow at queue link (u, v, q) (as we have seen in Sec. 4.2.2 that the burst of a flow changes at each hop on the path $P_{o_f,u}$ which leads from origin o to the source node of the edge u),
- t_f denotes the end-to-end delay requirement of the flow,

- l_f^{max} denotes the maximum packet size of the flow, and
- $P_f \subseteq \mathcal{E}^q$ denotes the set of queue link edges through which the flow is routed (empty set if the flow is not embedded yet).

We denote the maximum packet size in the network by L^{max}. If it is not known, the maximum Ethernet frame size can be used.
For a given queue link edge $(u, v, q) \in \mathcal{E}^q$,

- $\mathcal{F}_{(u,v,q)} \subseteq \mathcal{F}$ denotes the set of flows routed through the queue link edge,
- $U^R_{(u,v,q)}$ denotes the sum of the rates of the flows routed through the queue link edge, i.e.,

$$U^R_{(u,v,q)} \triangleq \sum_{f \in \mathcal{F}_{(u,v,q)}} r_f, \tag{4.11}$$

- $U^B_{(u,v,q)}$ denotes the sum of the bursts of the flows routed through the queue link edge, i.e.,

$$U^B_{(u,v,q)} \triangleq \sum_{f \in \mathcal{F}_{(u,v,q)}} b_f(P_{o_f,u}), \tag{4.12}$$

- $l^{max}_{(u,v,q)}$ denotes the maximum packet size of the aggregate flow traversing the queue link edge, i.e.,

$$l^{max}_{(u,v,q)} \triangleq \max_{f \in \mathcal{F}_{(u,v,q)}} \{l_f^{max}\}, \tag{4.13}$$

- $WCB_{(u,v,q)}$ denotes the worst-case burst of the queue link edge,
- $WCD_{(u,v,q)}$ denotes the worst-case delay of the queue link edge, and
- $B_{(u,v,q)}$ denotes the buffer capacity of the queue corresponding to the queue link edge.

Using these notations, Eqn. 4.6 and 4.7, can be respectively rewritten as

$$T_{(u,v,q)} = \frac{\sum_{j=1}^{p-1} U^B_{(u,v,q)} + \max\limits_{p+1 \le j \le \left|\mathcal{E}^q_{(u,v)}\right|} \{l^{max}_{(u,v,q)}\} + l^{max}_{(u,v,q)}}{R_{(u,v)} - \sum_{j=1}^{p-1} U^R_{(u,v,q)}} \tag{4.14}$$

and

$$R_{(u,v,q)} = R_{(u,v)} - \sum_{j=1}^{p-1} U^R_{(u,v,q)} \tag{4.15}$$

where $\beta^{RL}_{R_{(u,v,q)},T_{(u,v,q)}}$ is the rate-latency service curve offered by a queue link edge $(u,v,q) \in \mathcal{E}^q$. This rate-latency service curve is used to calculate the online worst case delay

$$WCD^{online}_{(u,v,q)} = \frac{\sum_{j=1}^{p} U^B_{(u,v,q)} + \max\limits_{p+1 \leq j \leq \left|\mathcal{E}^q_{(u,v)}\right|} \{l^{max}_{(u,v,q)}\} + l^{max}_{(u,v,q)}}{R_{(u,v)} - \sum_{j=1}^{p-1} U^R_{(u,v,q)}} \tag{4.16}$$

which is only valid if the online worst case burst can be capt by the buffer.

$$WCB^{online}_{(u,v,q)} = U^B_{(u,v,q)} + U^R_{(u,v,q)} T_{(u,v,q)} \leq B_{(u,v,q)} \tag{4.17}$$

Additionally, in order to respect the QoS requirements of embedded flows, we must have,

$$\sum_{(u,v,q) \in P_f} WCB^{online}_{(u,v,q)} \leq t_f \qquad \forall f \in \mathcal{F}. \tag{4.18}$$

4.4. Requirement for the Models: Fixed Per-Queue Delay

Both bounds in Eqn. 4.17 and 4.18 depend on $U^B_{(u,v,q)}$, $U^R_{(u,v,q)}$ and $l^{max}_{(u,v,q)}$ for some j, i.e., on the burst size, rate and maximum packet size of other flows embedded on the same physical link. If a new flow is embedded on a link $(u,v) \in \mathcal{E}^p$, the worst-case delay (Eqn. 4.16) and buffer consumption (Eqn. 4.17) of some of the queues at the link will be updated, thereby possibly violating requirements of some previously embedded flows (Eqn. 4.18) or exiting the buffer space (Eqn. 4.17). As explained in Sec. 2.5, we do not want to check that the delay requirements of the already embedded flows are still satisfied (i.e., check Eqn. 4.18) after a new flow embedding. That means that the worst-case bounds $WCD_{(u,v,q)}$ have to be bounded independently of the status of the network. In such a way, if Eqn. 4.18 for a given flow f was satisfied when the flow was embedded, it will be kept satisfied for the whole runtime of the network.
The two different models we present in the next Sections differ in the way they fix the $WCD_{(u,v,q)}$ bounds. While the MHM upper bounds the variable parts of Eqn. 4.16, the TBM fixes $WCD_{(u,v,q)}$ itself and lets the variables vary until the fixed threshold is reached.

4.5. Multi-Hop Model (MHM)

Our first model, the Multi-Hop Model (MHM), extends the access control scheme proposed by Schmitt et al. [111] for one aggregation node in order to consider

multi-hop paths and physical buffer limits. This section is mainly based on the work presented in [1, 5].

4.5.1. Network Calculus Developments

The model finds an upper bound for $WCD^{MHM}_{(u,v,q)}$ by replacing the variable components in Eqn. 4.16 with upper bounds for them.
Firstly, the packet size of a flow cannot be greater than the maximum packet size in the network. That is,

$$l_f^{max} \leq L^{max} \quad \forall f \in \mathcal{F}. \tag{4.19}$$

Secondly, the model assumes that the resource allocation algorithm allocates a data rate $A^R_{(u,v,q)}$ to each queue link edge. The rate of the aggregate flow traversing a queue is then limited by the access control scheme to the rate allocated to this queue. That is,

$$U^R_{(u,v,q)} \leq A^R_{(u,v,q)} \quad \forall\ (u,v,q) \in \mathcal{E}^q. \tag{4.20}$$

Obviously, rates must be allocated such that

$$\sum_{p=1}^{\left|\mathcal{E}^q_{(u,v)}\right|} A^R_{(u,v,q)} \leq R_{(u,v)} \forall\ (u,v,q) \in \mathcal{E}^q. \tag{4.21}$$

From Eqn. 4.16 and 4.17, Eqn. 4.19 and 4.20 allow to compute the following upper bounds for the worst-case delay and backlog at a queue link edge[1].

$$WCD^{online}_{(u,v,q)} \leq \frac{\sum_{j=1}^{p} U^B_{(u,v,q)} + 2L^{max}}{R_{(u,v)} - \sum_{j=1}^{p-1} A^R_{(u,v,q)}} \tag{4.22}$$

$$WCB^{online}_{(u,v,q)} \leq U^B_{(u,v,q)} + A^R_{(u,v,q)} \frac{\sum_{j=1}^{p} U^B_{(u,v,q)} + 2L^{max}}{R_{(u,v)} - \sum_{j=1}^{p-1} A^R_{(u,v,q)}} \leq B_{(u,v,q)} \tag{4.23}$$

Finally, the burst of the aggregate flow traversing a queue has to be limited such that it does not generate any buffer overflow. If we refer to the maximum allowed burst at a queue as $A^B_{(u,v,q)}$, i.e.,

$$U^B_{(u,v,q)} \leq A^B_{(u,v,q)} \qquad \forall (u,v,q) \in \mathcal{E}^q, \tag{4.24}$$

[1]Note that Eqn. 4.21 ensures that the denominators in Eqn. 4.22 and 4.23 are strictly positive, thereby ensuring that the bounds are both positive and not infinite nor undefined.

these $A^B_{(u,v,q)}$ bounds must be computed such that

$$WCB^{MHM}_{(u,v,q)} = A^B_{(u,v,q)} + A^R_{(u,v,q)} \frac{\sum_{j=1}^{p} A^B_{(u,v,q)} + 2L^{max}}{R_{(u,v)} - \sum_{j=1}^{p-1} A^R_{(u,v,q)}} \leq B_{(u,v,q)} \tag{4.25}$$

The model also assumes that the resource allocation algorithm manages $A^B_{(u,v,q)}$. Eqn. 4.25 allows to recursively compute the $A^B_{(u,v,q)}$ values independently of the state of the network. $\alpha^{TB}_{A^B_{(u,v,q)},A^R_{(u,v,q)}}$ corresponds to the maximum token bucket arrival curve allowed to traverse a given queue link (u, v, q). We will denote it as $\alpha^{MHM}_{(u,v,q)}$.
As a result, Eqn. 4.22, can be rewritten as

$$WCD^{online}_{(u,v,q)} \leq \frac{\sum_{j=1}^{p} A^B_{(u,v,q)} + 2L^{max}}{R_{(u,v)} - \sum_{j=1}^{p-1} A^R_{(u,v,q)}} = WCD^{MHM}_{(u,v,q)} \tag{4.26}$$

where $WCD^{MHM}_{(u,v,q)}$ is the upper bound of the worst-case delay $WCD^{online}_{(u,v,q)}$ of a queue link $(u, v, q) \in \mathcal{E}^q$ used by the MHM and that is independent of the state of the network.

4.5.2. Model Operations

From these developments, the four DetServ model functions of the MHM are defined in Fig. 4.3.
The model uses $U^B_{(u,v,q)}$ and $U^R_{(u,v,q)}$ as state variables for each queue $(u, v, q) \in \mathcal{E}^q$. The registration and deregistration methods simply consist in updating these variables. The access control for a new flow simply consists of checking that Eqn. 4.20 and 4.24 are always satisfied. Based on the rate allocated by the resource allocation algorithm to each queue in the network, the $A^B_{(u,v,q)}$ and $WCD^{MHM}_{(u,v,q)}$ bounds can be computed once for each queue link edge $(u, v, q) \in \mathcal{E}^q$ and the four model functions then require low computation overhead.

4.5.3. Limitations of the Multi-Hop Model

The MHM requires a data rate and maximum burst to be allocated to each queue. These allocations then define the maximum rate and burst allowed at each queue, as well as the maximum delay of each queue. The access control checks the availability of two resources: burst and rate (HASACCESS method in Fig. 4.3). Hence, it can happen that the access to a queue is blocked because its rate budget is exhausted, while its burst limit $A^B_{(u,v,q)}$ is not reached. In such a

1: **function** GETDELAY((u, v, q))
2: **return** $WCD^{MHM}_{(u,v,q)}$ (Eqn. 4.26)
3:
4: **function** HASACCESS(f, (u, v, q))
5: **if** $U^B_{(u,v,q)} + b_f(P_{o_f,u}) \leq A^B_{(u,v,q)}$ **and** $U^R_{(u,v,q)} + r_f \leq A^R_{(u,v,q)}$ **then**
6: **return** true
7: **else**
8: **return** false
9:
10: **function** REGISTERPATH(f, P_f)
11: **for** $(u, v, q) \in P_f$ **do**
12: $U^B_{(u,v,q)} \leftarrow U^B_{(u,v,q)} + b_f(P_{o_f,u})$
13: $U^R_{(u,v,q)} \leftarrow U^R_{(u,v,q)} + r_f$
14:
15: **function** DEREGISTERPATH(f, P_f)
16: **for** $(u, v, q) \in P_f$ **do**
17: $U^B_{(u,v,q)} \leftarrow U^B_{(u,v,q)} - b_f(P_{o_f,u})$
18: $U^R_{(u,v,q)} \leftarrow U^R_{(u,v,q)} - r_f$

Figure 4.3.: The four DetServ model functions for the Multi-Hop Model (MHM). The model uses $U^B_{(u,v,q)}$ and $U^R_{(u,v,q)}$ as state variables for each queue $(u, v, q) \in \mathcal{E}^q$. The registration and deregistration of a path in the network simply consists in updating these variables. For its part, the access control simply consists in checking that the state variables never exceed their respective limits, which are defined is such a way that, if the variables stay below these limits, *(i)* the maximum backlog at a queue will never exceed the buffer size of the queue, thereby avoiding any buffer overflow, and *(ii)* the maximum delay for a queue will not exceed the delay returned by GETDELAY for this queue.

situation, it would be beneficial to artificially reduce the buffer size $A^B_{(u,v,q)}$ of the queue. Indeed, this would, by Eqn. 4.26, reduce $WCD^{MHM}_{(u,v,q)}$ of the queue itself and lower priority queues could then either *(i)* see their maximum delay reduced (by Eqn. 4.26) or *(ii)* see their maximum allowed burst or rate increased (by Eqn. 4.25). However, the opposite situation could also happen. That is, the buffer capacity could be the bottleneck, in which case it would be beneficial to trade-off rate in order to increase the maximum allowed bursts or reduce the maximum delays.

In other words, the MHM requires the resource allocation algorithm to be responsible for adjusting the trade-off between the resources, that is, to make an *a priori* choice between buffer space, data rate, buffer capacity and delay. However, adjusting this trade-off requires to know what is the bottleneck in the

network or at a given link. Would flows be rejected because there is no buffer capacity available anymore, no data rate available anymore, or because their delay cannot be satisfied? Unfortunately, answering this question requires to know the traffic demand, which is, because of our online approach (see Sec. 1.1), not the case.

4.6. Threshold-Based Model (TBM)

The Threshold-Based Model (TBM) solves the shortcoming of the MHM by choosing between buffer capacity and data rate as flows are added to the network, thereby allocating the rate and buffer capacity resources only when needed rather than pre-allocating them without knowing future flow requests.

4.6.1. Model Operations

In the TBM, the worst-case delay of each queue (Eqn. 4.16) is simply fixed by defining a threshold $A^{WCD}_{(u,v,q)}$. Then, flows are accepted in a queue as long as the worst-case delay of the queues at the same link do not exceed their respective thresholds.
This approach has two main benefits. First, as mentioned, the data rate and buffer space resources are allocated only when needed, rather than *a priori*, thereby leading to a better utilization of the resources. Second, the resource allocation algorithm is now simplified since it only has to optimize with respect to one variable (the time) rather than two (buffer space and data rate). In other words, the TBM replaces the three data rate, buffer space and delay resources by a *single* one: delay.
Unfortunately, this comes at the cost of a higher computational complexity for access control. Indeed, as $U^{R}_{(u,v,q)}$ is not bounded anymore, it is not anymore possible to compute a bound on the service curves offered to the different queues (i.e., on the $T_{(u,v,q)}$ and $R_{(u,v,q)}$ parameters). Adding a flow in a queue will update the service curve offered to lower priority queues (by Eqn. 4.14 and 4.15). Hence, when adding a flow in a queue (u, v, q), besides checking that $WCD^{online}_{(u,v,q)} \leq A^{WCD}_{(u,v,q)}$, the access control mechanism has to check that the thresholds of lower priority queues are also not exceeded. That is, the access control mechanism has to check that

$$WCD^{online}_{(u,v,j)} \leq A^{WCD}_{(u,v,j)} \quad \forall\, j : q \leq j \leq \left|\mathcal{E}^{q}_{(u,v)}\right|. \tag{4.27}$$

Besides, the access control scheme has to make sure that no buffer overflow can

be caused by the embedding of the new flow, i.e.,

$$WCB^{online}_{(u,v,j)} \leq B_{(u,v,j)} \quad \forall\, j : q \leq j \leq \left|\mathcal{E}^q_{(u,v)}\right|. \tag{4.28}$$

1: **function** GETDELAY((u, v, q))
2: **return** $A^{WCD}_{(u,v,q)}$
3:
4: **function** HASACCESS(f, (u, v, q))
5: **for** $i \in \{p, \ldots, \left|\mathcal{E}^q_{(u,v)}\right|\}$ **do**
6: $WCD^{TBM}_{(u,v,q)} \leftarrow$ Eqn. 4.29 including new flow
7: $WCB^{TBM}_{(u,v,q)} \leftarrow$ Eqn. 4.30 including new flow
8: **if** $WCD^{TBM}_{(u,v,q)} > A^{WCD}_{(u,v,q)}]$ **or** $WCB^{TBM}_{(u,v,q)} > B_{(u,v,q)}$ **then**
9: **return** false
10: **return** true
11:
12: **function** REGISTERPATH(f, P_f)
13: **for** $(u, v, q) \in P_f$ **do**
14: $U^B_{(u,v,q)} \leftarrow U^B_{(u,v,q)} + b_f(P_{o_f,u})$
15: $U^R_{(u,v,q)} \leftarrow U^R_{(u,v,q)} + r_f$
16: Update $l^{max}_{(u,v,q)}$
17:
18: **function** DEREGISTERPATH(f, P_f)
19: **for** $(u, v, q) \in P_f$ **do**
20: $U^B_{(u,v,q)} \leftarrow U^B_{(u,v,q)} - b_f(P_{o_f,u})$
21: $U^R_{(u,v,q)} \leftarrow U^R_{(u,v,q)} - r_f$
22: Update $l^{max}_{(u,v,q)}$

Figure 4.4.: The four DetServ model functions for the Threshold-Based Model (TBM). The threshold on the delay of a queue is chosen by the resource allocation algorithm. Access to a queue link edge $(u, v, q) \in \mathcal{E}^q$ is then checked by checking that the new worst-case bound does not exceed its threshold value. Besides, as the state of a queue influences the state of lower priority queues, the access control mechanism also has to check that the worst-case bounds of lower priority queues do not exceed their respective thresholds. Finally, the buffer capacity also has to be checked for the different queues.

Note that Eqn. 4.16 and 4.17 require the knowledge of the maximum packet size in lower priority queues. This means that, when embedding a flow in a queue,

higher priority queues also have to be checked since the maximum packet size might have changed. However, because best-effort traffic flows through the lowest priority queue, we cannot keep track of this value and we hence replace it by L^{max}. From this, we have

$$WCD^{TBM}_{(u,v,q)} = \frac{\sum_{j=1}^{p} U^{B}_{(u,v,j)} + L^{max} + l^{max}_{(u,v,j)}}{R_{(u,v)} - \sum_{j=1}^{p-1} U^{R}_{(u,v,j)}} \leq A^{WCD}_{(u,v,j)}, \tag{4.29}$$

and

$$WCB^{TBM}_{(u,v,q)} = U^{B}_{(u,v,j)} + U^{R}_{(u,v,j)} \frac{\sum_{j=1}^{p-1} U^{B}_{(u,v,j)} + L^{max} + l^{max}_{(u,v,j)}}{R_{(u,v)} - \sum_{j=1}^{p-1} U^{R}_{(u,v,j)}} \leq B_{(u,v,j)}, \tag{4.30}$$

which only depend on the state of higher priority queues. As a result, it is sufficient to only check lower priority queues when embedding a new flow.
The four model functions of the TBM are given in Fig. 4.4. As for the MHM, the registration and deregistration methods simply consist in updating the state variables. However, we here have one additional state variable: the maximum packet size at each queue. The delay of a queue link edge is now the one fixed by the resource allocation algorithm and the access control scheme simply verifies that Eqn. 4.27 and 4.28 would still be verified for the subject queue and the lower priorities queues if the flow is embedded.

4.6.2. Shortcomings of the TBM

The TBM, though having major advantages, presents two drawbacks. First, the complexity of the HASACCESS model function is increased by a factor of up to $\left|\mathcal{E}^{q}_{(u,v)}\right|$. Because the HASACCESS function is called each time the routing algorithm visits an edge, this might have a considerable influence on the overall request processing time. However, we will show in Sec. 6.3 that the increase in runtime is reasonable and acceptable for industrial scenarios. Second, the model presents an inherent blocking problem. Indeed, if a low priority queue is close to its delay threshold, it will block further embeddings in higher priority queues, even if these are still far from their own delay threshold. Consequently, the routing algorithm has now to operate cautiously when embedding flows in order to avoid such a blocking situation which would inevitably cause resource waste.

4.7. Computation of the Burst Increase

Though we mentioned that the burst of a flow changes at each hop, we did not explain how these changes can be computed on a *per-flow* basis and how this impacts delay computations. From Sec. 4.2, we know that an aggregate flow with arrival curve $\alpha^{TB}_{U^R_{(u,v,q)},U^B_{(u,v,q)}}$ traversing a queue offering a service curve $\beta^{RL}_{R_{(u,v,q)},T_{(u,v,q)}}$ will see its burst $U^B_{(u,v,q)}$ increased by $U^R_{(u,v,q)}T_{(u,v,q)}$, i.e.,

$$U^B_{(u,v,q)}{}^* = U^B_{(u,v,q)} + U^R_{(u,v,q)}T_{(u,v,q)}. \tag{4.31}$$

$U^B_{(u,v,q)}{}^*$ is the new burst of the entire aggregate. Nevertheless, the flows composing this aggregate might take different routes at the next hop and the individual burst increases of the individual flows composing the aggregate must be computed. For that reason the burstiness of a flow $b_f(P_{o_f,u})$ is depending on the path it has taken the node u.
There are four opportunities for implementing $b_f(P_{o_f,u})$:

BI-REAL Eqn. 4.31 can be rewritten as a multi hop version. Therefore, the burst increase of each hop the flow has taken to this particular node has to be summed up:

$$b_f(P_{o_f,u}) = b_f + r_f \sum_{(u,v,q)\in P_{o_f,u}} T_{(u,v,q)}. \tag{4.32}$$

The use of $T_{(u,v,q)}$ would cause that when a flow is embedded in a queue, the burst increases of other flows traversing the same link might change, possibly violating already performed access control checks. As explained in Sec. 2.5, we want to avoid such a situation and the burst increase of a flow must therefore be, as the worst-case delay of a queue, independent of the network state.

The $WCD_{(u,v,q)}$ is used to achieve a static behavior of the network model. Since $WCD_{(u,v,q)}$ is always bigger equal then the corresponding $T_{(u,v,q)}$

$$T_{(u,v,q)} \leq WCD_{(u,v,q)} \tag{4.33}$$

equation 4.32 can be reformulate to

$$b_f(P_{o_f,u}) = b_f + r_f \sum_{(u,v,q)\in P_{o_f,u}} WCD_{(u,v,q)}. \tag{4.34}$$

by not loosing the determinism.

On the one hand the **REAL** mode provides accurate bounds by considering the burst increase in the model. On the other hand the formulation is

depending on the whole path it has taken and is used in the HASACCESS method. In [6] we call this an M_∞ metric. This type of metrics lead to a violation of the Optimal substructure property. Which implies that optimality and completeness of state of the art routing algorithms could be lost.

BI-WCB To avoid the violation of the Optimal substructure property the Worst Case Burst increase could be considered. Therefore, we simply consider the delay constraint of the flow t_f as the delay the which was experienced so fare, knowing that this delay will be never exited.

$$b_f(P_{o_f,u}) = b_f + r_f t_f. \tag{4.35}$$

This consideration is valid but less tied then the **REAL** mode.

BI-WCB-RR Worst Case Burst - Real Reservation if the compromise of both. It uses **WCB** only during the routing phase, which allows to use optimal and complete algorithms. The **BI-REAL** mode is used during the reservation phase, which enables more tide bounds than the pure **WCB** mode.

BI-NO We note that, if the cycle time (or inter-arrival time of packets) of a flow is greater than its worst-case delay bound, then the burst increase can be neglected. Indeed, in such a case, a packet is ensured to reach its destination before the following packet is sent. As a result, packets of the same flow will not queue up at any queue and the burst of the flow will never increase. In this case the following burst increase formulation could be used:

$$b_f(P_{o_f,u}) = b_f. \tag{4.36}$$

The performance of this burst increase modes is evaluated in chapter 6.3.

4.8. Input Link Shaping (ILS)

4.8.1. Towards Lower Bounds

So far, we considered that the arrival curve of the aggregate flow entering a queue $(u, v, q) \in \mathcal{E}^q$ is $\beta^{RL}_{R_{(u,v,q)},T_{(u,v,q)}}$, that is, that the burst of the aggregate flow entering a queue is given by the sum of all the bursts of all the flows composing the aggregate (see Eqn. 4.12). Nevertheless, the individual flows come from physical links with limited capacities. Hence, the amount of traffic entering a given queue is further limited by the capacity of the links it is coming from. Considering this new bound on the traffic entering a queue, we can lower

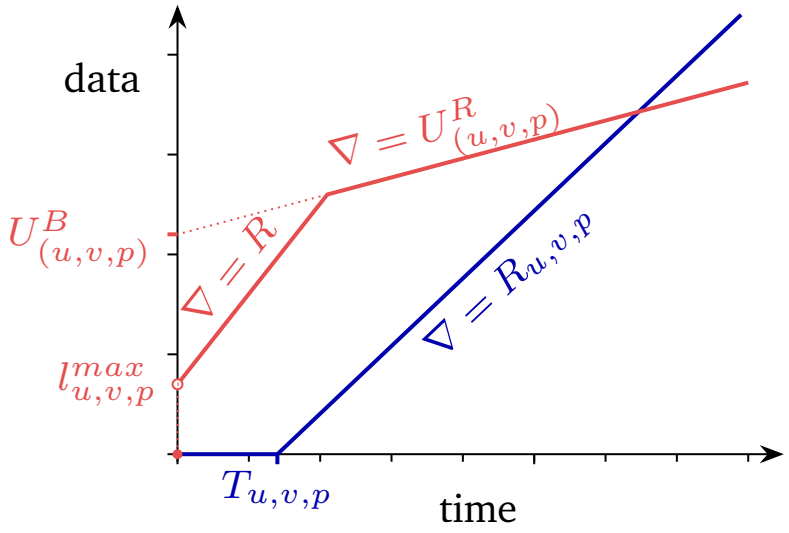


Figure 4.5.: Shaped arrival curve of an aggregate flow traversing a queue $(u, v, q) \in \mathcal{E}^q$ coming from an input link with rate R. The knowledge of the physical properties of the input link of the flow allows us to limit the burst and rate of the aggregate respectively to the maximum packet size $l^{max}_{u,v,p}$ of the flow and to the maximum rate R of the link. Graphically, we can easily see that such a shaping reduces the values of the backlog and delay bounds.

the corresponding arrival curves, yielding lower bound values and thereby potentially accepting more flows in the network.
The idea, to which we refer to as Input Link Shaping (ILS), is illustrated in Fig. 4.5 for a given queue (u, v, q) traversed by a set of flows coming from a common input link of capacity R. From the knowledge of the physical properties of the input link, besides its traditional arrival curve, the aggregate flow is additionally constrained by a token bucket arrival curve with rate R and burst $l^{max}_{u,v,p}$. A better arrival curve for a flow constrained by two different token bucket arrival curves being the minimum of these curves [135], the new arrival curve of the aggregate flow is of the form shown in Fig. 4.5. We can see that the backlog and delay bounds will always be smaller than if shaping was not taken into account, highlighting the benefit of ILS.

4.8.2. ILS Does Not Contradict Network Calculus

In Sec. 4.2, we have presented network calculus results for computing the output arrival curve of a flow after it has traversed a network node characterized by a given service curve. We now propose to cut off a part of this arrival curve by shaping it with the input link rate. Though this is intuitive, it might seem to contradict the network calculus results which say that a big burst could happen. The justification is the following. The results of network calculus theory are

solely based on the arrival and service curve concepts. While the service curve gives a lower bound on the service a network node will offer to a flow, it does not specify anything regarding the maximum service the node could offer, hence potentially allowing infinite service, i.e., infinite rate. Taking this into account, network calculus results consider that an infinite service could instantly output the current backlog as a single burst, which is why, in Eqn. 4.10, the output burst corresponds to the worst-case backlog. As a matter of fact, we know more than what the service curve concept provides to network calculus theory. Indeed, we know that the service provided by the network node could never be higher than the link rate. The shaping we introduce is hence augmenting network calculus results, rather than contradicting them.

4.8.3. Adapting the MHM

In the MHM, the worst-case delay of a queue is made independent of the network state by statically defining the maximum arrival curves allowed at each queue. Therefore, to keep the worst-case delay of a queue static, ILS must be introduced in a way that is also independent of the network state. For a given queue-link edge$(u, v, q) \in \mathcal{E}^q$, the worst-case burst that could ever enter the queue is nL^{max} where n is the number of links entering node u. The worst-case rate is for its part given by the sum of the rates of the individual incoming links. Therefore, the arrival curve $\alpha^{MHM}_{(u,v,q)}$ considered so far can be replaced by

$$\alpha^{MHM-ILS}_{(u,v,q)} = \min \left\{ \sum_{x:(x,u)\in\mathcal{E}^q} \left(\alpha^{TB}_{R_{x,u},L^{max}}\right), \alpha^{MHM}_{(u,v,q)} \right\}. \tag{4.37}$$

To present the worst case burst $WCB^{MHM-ILS}_{(u,v,q)}$ and the worst case delay $WCD^{MHM-ILS}_{(u,v,q)}$ in a clearly arranged way we swap the presentation stile to the curve based formulation presented in section 4.2.1. There two functions where introduced. One to compute the vertical deviation between a arrival curve and a service curve ($v(\alpha, \beta)$) and one to compute the horizontal deviation between a arrival curve and a service curve ($h(\alpha, \beta)$). The service curve of MHM can be derived form equation 4.5 and 4.26. Which is:

$$\beta^{MHM}_{(u,v,q)} = \left(\beta^{RL}_{R_{(u,v)},2\frac{L^{max}}{R_{(u,v)}}} - \sum_{j=1}^{p-1} \alpha^{MHM}_{(u,v,j)} \right)^{+}. \tag{4.38}$$

Based on the arrival curve $\alpha^{MHM-ILS}_{(u,v,q)}$ and the service curve $\beta^{MHM}_{(u,v,q)}$ the worst case burst

$$WCB^{MHM-ILS}_{(u,v,q)} = h\left(\alpha^{MHM-ILS}_{(u,v,q)}, \beta^{MHM}_{(u,v,q)}\right) \leq WCD^{MHM}_{(u,v,q)} \leq B_{(u,v,q)} \tag{4.39}$$

and the worst case delay

$$WCD^{MHM-ILS}_{(u,v,q)} = v\left(\alpha^{MHM-ILS}_{(u,v,q)}, \beta^{MHM}_{(u,v,q)}\right) \leq WCD^{MHM}_{(u,v,q)} \tag{4.40}$$

are calculated. Please not that the equations 4.39 and 4.40 are a drop in replacement for the original MHM formulation (Equ. 4.25 and 4.26). The new formulation will never exit the bounds of the original MHM but has the potential to provide more tied once. In addition this formulation is only considered by the resource allocation algorithm which computes $A^R_{(u,v,q)}$ and $A^B_{(u,v,q)}$ by using the new formulation. Once these computations are done, the four DetServ model functions described in Fig. 4.3 are manly left unchanged. Only the GETDELAY method has to return the new static delay bound $WCD^{MHM-ILS}_{(u,v,q)}$.

4.8.4. Adapting the TBM

While present, the benefits of ILS for the MHM are limited. Indeed, since we only keep track of worst-case arrival curves, ILS also has to be done worst-case, i.e., considering the worst-case packet size and rates coming from each input link.

For the threshold-based model, the arrival curves are computed live. Therefore, the maximum packet size and rate for each incoming link can also be computed on the fly. This can be done by introducing three new state variables $U^R_{(m,u,v,q)}$, $U^B_{(m,u,v,q)}$ and $l^{max}_{(m,u,v,q)}$ keeping track respectively of the rate, burst and maximum packet size of the aggregate flow coming from the physical edge (u, v) and traversing the queue-link edge (u, v, q). Instead of considering the arrival curve consisting of the sum of all the arrival curves of the flows entering the queue, the contribution of each input link can now be shaped individually. That is, the arrival curve considered at a queue (u, v, q) is now

$$\alpha^{TBM-ILS}_{(u,v,q)} = \sum_{x:(x,u)\in\mathcal{E}^q} \left(\min\left\{\alpha^{TB}_{R_{(x,u)},l^{max}_{(x,u,v,q)}}, \alpha^{TB}_{U^R_{(x,u,v,q)},U^B_{(x,u,v,q)}}\right\}\right), \tag{4.41}$$

i.e., a sum of shaped arrival curves.

An example for two input links is shown in Fig. 4.6. One can see that the summed up arrival curve can have up to n knee points, where n is the number of physical input links.

For the same reasons as for the MHM, but with increased impact since shaping is done with the current real values, the computed worst-case delay and backlog

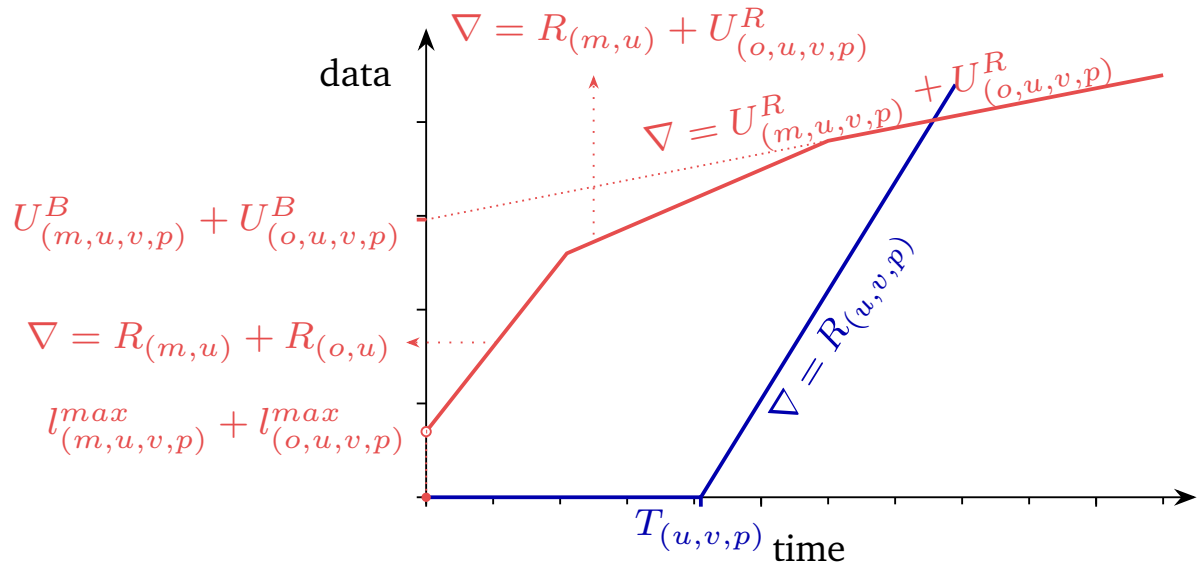


Figure 4.6.: Example of shaped arrival curve for the TBM. The aggregate flow traversing queue (u, v, q) comes from two input links (m, u) and (o, u). Each input link has shaped the traffic it carries as shown in Fig. 4.5 and the resulting aggregate, corresponding to the sum of the two shaped arrival curves, is then composed of three segments with decreasing slopes. The backlog and delay bounds can then be reached at any angular point of both curves. One can see that the bounds will always be lower than if shaping was not taken into account.

values will be lower. As a consequence, the limits $A^{WCD}_{(u,v,q)}$ and $B_{(u,v,q)}$ will be reached later, thereby potentially allowing more flows to be accepted.
Fig. 4.7 fhows the new ILS avare imlementation of the TBM. Obviously, the GET-DELAY method does not change in comparison to Fig. 4.4. The REGISTERPATH and DEREGISTERPATH methods have to be updated to keep track of the new state variables. For its part, the HASACCESS method only has to be changed at lines 6-7. Since the arrival curves are not token buckets anymore, the formulas for computing the worst-case delay $WCD^{TBM}_{(u,v,q)}$ (Eqn. 4.29) and backlog $WCB^{TBM}_{(u,v,q)}$ (Eqn. 4.30) are not valid anymore and these values have now to be computed geometrically as it is done for the adaption of the MHM. First we define the new service curve for the ILS enabled TBM as

$$\beta^{TBM-ILS}_{(u,v,q)} = \left(\beta^{RL}_{R_{(u,v)}, 2\frac{L^{max}}{R_{(u,v)}}} - \sum_{j=1}^{p-1} \alpha^{TBM-ILS}_{(u,v,j)} \right)^{+}. \tag{4.42}$$

From this, we have the worst case delay

$$WCD^{TBM-ILS}_{(u,v,q)} = h\left(\alpha^{TBM-ILS}_{(u,v,q)}, \beta^{TBM-ILS}_{(u,v,q)}\right) \leq WCD^{TBM}_{(u,v,q)} \tag{4.43}$$

1: **function** GETDELAY(f,(u, v, q))
2: **return** $A^{WCD}_{(u,v,q)}$
3:
4: **function** HASACCESS(f, (m, u, v, q))
5: **for** $i \in \{p, \ldots, \left|\mathcal{E}^q_{(u,v)}\right|\}$ **do**
6: $WCD^{TBM-ILS}_{(u,v,q)} \leftarrow$ Eqn. 4.43 including new flow
7: $WCB^{TBM-ILS}_{(u,v,q)} \leftarrow$ Eqn. 4.44 including new flow
8: **if** $WCD^{TBM-ILS}_{(u,v,q)} > A^{WCD}_{(u,v,q)}]$ **or** $WCB^{TBM-ILS}_{(u,v,q)} > B_{(u,v,q)}$ **then**
9: **return** false
10: **return** true
11:
12: **function** REGISTERPATH(f, P_f)
13: **for** $(m, u, v, q) \in P_f$ **do**
14: $U^B_{(m,u,v,q)} \leftarrow U^B_{(m,u,v,q)} + b_f(P_{o_f,u})$
15: $U^R_{(m,u,v,q)} \leftarrow U^R_{(m,u,v,q)} + r_f$
16: Update $l^{max}_{(m,u,v,q)}$
17:
18: **function** DEREGISTERPATH(f, P_f)
19: **for** $(m, u, v, q) \in P_f$ **do**
20: $U^B_{(m,u,v,q)} \leftarrow U^B_{(m,u,v,q)} - b_f(P_{o_f,u})$
21: $U^R_{(m,u,v,q)} \leftarrow U^R_{(m,u,v,q)} - r_f$
22: Update $l^{max}_{(m,u,v,q)}$

Figure 4.7.: The four DetServ model functions for the TBM with ILS. The threshold on the delay of a queue is chosen by the resource allocation algorithm. Access to a queue link edge $(u, v, q) \in \mathcal{E}^q$ is then checked by checking that the new worst-case bound does not exceed its threshold value. Besides, as the state of a queue influences the state of lower priority queues, the access control mechanism also has to check that the worst-case bounds of lower priority queues do not exceed their respective thresholds. Finally, the buffer capacity also has to be checked for the different queues.

and the worst case burst

$$WCB_{(u,v,q)}^{TBM-ILS} = v\left(\alpha_{(u,v,q)}^{TBM-ILS}, \beta_{(u,v,q)}^{TBM-ILS}\right) \leq WCB_{(u,v,q)}^{TBM} \tag{4.44}$$

to check if the given delay and buffer constraints are not violated. Pleas note that this new formulation will never exit the bounds of the original TBM but has the potential to provide more tied once.

4.8.5. Impact on the Performance of the MHM

As mentioned, because the MHM performs shaping based on worst-case values, we expect the impact on the amount of flows that can be embedded to be quite low. Nevertheless, as everything is computed during initialization, the request processing time of the MHM should not be affected by ILS. Hence, for the MHM, ILS has only benefits, though limited.

4.8.6. Impact on the Performance of the TBM

On the contrary, the TBM performs shaping based on the current traffic. Hence, the impact of ILS on the amount of flows that can be accepted in the network is expected to be greater than for the MHM. While ILS does not slow down the MHM, the runtime of the TBM should be much more affected. Indeed, the increased amount of knee points in the arrival curves does not allow anymore the computation of the worst-case delay and burst with formulas. From the convexity of the region between the curves (see Fig. 4.6), the delay (resp. backlog) bound can be computed by comparing the horizontal (resp. vertical) deviation at each knee point of the two curves. This inevitably slows down the HASACCESS method. Hence, ILS is expected to have a major impact on the TBM, both in terms of increased performance and increased runtime.

Table 4.1.: Summary of Main Notations

General model	
$\mathcal{N}$	set of nodes of a graph ($\mathcal{G}$)
$\mathcal{E}$	set of edges of a graph ($\mathcal{G}$)
$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	graph consisting of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$)
Network model	
(u, v)	edge of a physical link from node u to node v
(u, v, q)	edge of a Queue link from node u to node v via queue q
(m, u, v, q)	edge of a Queue link from node u to node v via queue q from physical edge m
$\mathcal{E}^p = \{(u, v)\}$	set of edges of a graph ($\mathcal{G}$)
$\mathcal{G}^p = (\mathcal{N}, \mathcal{E}^p)$	graph consists of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$)
$\mathcal{E}^q = \{(u, v, q)\}$	set of edges of a graph ($\mathcal{G}$)
$\mathcal{G}^q = (\mathcal{N}, \mathcal{E}^q)$	graph consists of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$)
$\mathcal{E}^q_{(u,v)} \subseteq \mathcal{E}^q$	Set of queue links (u, v, q) traversing physical link (u, v)
$\left\lvert \mathcal{E}^q_{(u,v)} \right\rvert$	number of queues of a physical link (u, v)
Flow model	
$\mathcal{F}$	set of flows (f) embedded in the Graph $\mathcal{G}$
o_f, d_f	Origin and destination nodes of flow f
r_f	Avg. (mean) bit rate [bit/s] of flow f
b_f	Burstiness parameter [Byte] of flow f
t_f	End-to-end delay limit [s] of flow f
Path model	
$\mathcal{P}_{(o,d)}$	path of a flow f from origin o to destination d over edges in $\mathcal{E}$.
P_f	path of a flow f which contains the edges (m, u, v, q) from to
$\mathrm{Cost}(P_f)$	Cost of path P_f
$D(P_f)$	End-to-end worst-case delay of path P_f
Network resource model	
$R_{(u,v)}$; $R_{(u,v,q)}$	service data rate of an edge
$T_{(u,v,q)}$	service delay of an edge
$WCD_{(u,v,q)}$	Worst Case Delay of an edge
$WCB_{(u,v,q)}$	Worst Case Burst of an edge
$B_{(u,v,q)}$	buffer capacity of an edge
$\mathcal{A}$	set of resource allocation parameter
$A^R_{(u,v,q)}$; $A^R_{(m,u,v,q)}$	Bit rate allocated to an edge
$A^B_{(u,v,q)}$; $A^B_{(m,u,v,q)}$	Buffer capacity alocated to an edge
$A^{WCD}_{(u,v,q)}$	Worst Case Dealy alocated to an edge
$\mathcal{U}$	set of resource utilisation parameter
$U^R_{(u,v,q)}$; $U^R_{(m,u,v,q)}$	Bit rate utilisation of an edge
$U^B_{(u,v,q)}$; $U^B_{(m,u,v,q)}$	Buffer capacity utilisation of an edge

5. Optimization Problems

Before we can discuss the different optimization problems, a definition of the optimization goal has to be stated. We assume that there is an exponentially distributed arrival process, which creates new requests with a rate $\lambda_{\mathcal{F}}$. The termination of a flow is also defined as an exponentially distributed process with an average holding time $h_{\mathcal{F}}$. This assumption is possible because each independent automation module within a factory will request real-time connections. Since the number of automation modules is increasing the number of independent connection requests increases. This leads to the fact that the arrival and the termination process could be modeled as an exponentially distributed process. The traffic intensity the system has to cover is defined as

$$y = \lambda_{\mathcal{F}} h_{\mathcal{F}}. \tag{5.1}$$

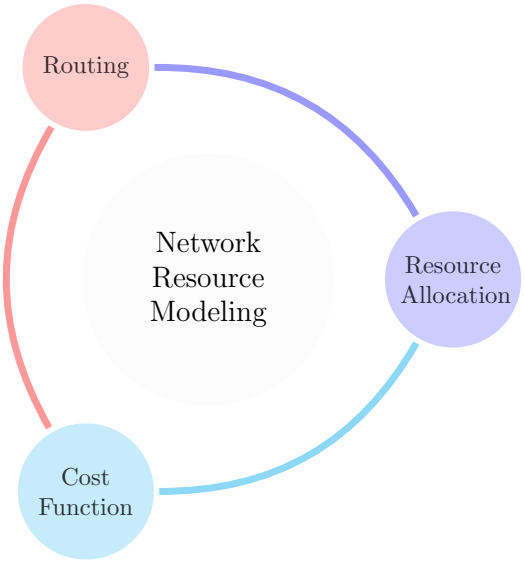


Figure 5.1.: dependencies of Delay-Constrained Least-Cost (DCLC) Routing, Resource Allocation and Cost Function

Erlang formula B defines the blocking probability BP as a function of the number of available connection N and the traffic intensity y.

$$BP(N, y) = \frac{\frac{y^N}{N!}}{\sum_{i=0}^{N} \frac{y^i}{i!}} \tag{5.2}$$

The objective of the real-time Quality of Service (QoS) framework is to maximize the traffic intensity (y) in a given network by varying the paths of all the embedded flows and increase the flow f arrival rate ($\lambda_{\mathcal{F}}$).

$$\max_{\lambda_{\mathcal{F}}} y \tag{5.3}$$

Equation 5.2 describes the probability of blocking for classical telecommunication systems. The calculation of the number of channels N for a multi hop network is not straightforward. However, we can calculate the blocking probability BP by an online simulation of the QoS framework. This blocking probability (BP) should be smaller than a maximum blocking probability (BP_{max}).

$$BP \leq BP_{max} \tag{5.4}$$

Figure 5.1 shows that the industrial QoS framework divides into four parts. In the previous chapter the network resource models were discussed. This chapter covers the three optimization problems/algorithms. First we discuss how the routing algorithm (Section 5.1) influences the performance of the industrial QoS framework. The routing algorithms performance mainly depends on the cost function (Section 5.2). The cost functions defines which resources are rare and thereforemore expensive. The resource allocation (Section 5.3) adapts the static resource assignment of the network resource model to work around such rare resources. This chapter is mainly based on [3].

5.1. Routing Problem

Routing, i.e., determining a route (path) from a source node to a destination node through a sequence of intermediate switching nodes, is an elementary function of the network layer in communication networks. Given the importance of routing for communication networks, a diverse array of routing algorithms have been designed.
Providing QoS is an important requirement for a wide range of communication network settings and applications. For instance, multimedia network applications require QoS from the network service, as do many network applications in industrial networks [144] and the smart grid [145] as well as networked control

systems [146]. The required QoS is often in the form of delay bounds (constraints) for the data packets traversing the network. Accordingly, extensive research has developed routing algorithms that satisfy given delay constraints while minimizing some cost metric, i.e., so-called DCLC routing algorithms. This problem statement is of the following form:

$$\min_{P_f \in \mathcal{P}_{(o,d)}} \text{Cost}(P_f)$$
$$D(P_f) \leq t_f$$

which corresponds to the definition of the routing algorithm needed in the function split (Section 2.4). DCLC routing algorithms and similar routing algorithms that support QoS networking are often referred to as *QoS routing algorithms*.
Generally, the route determination (computation) is either carried out in distributed nodes, e.g., the control modules in individual distributed Internet Protocol (IP) routers, or by a centralized controller, e.g., a Software Defined Networking (SDN) controller [147, 148, 39, 149]. Distributed routing algorithms have been intensely researched for traditional IP routing, e.g., [32, 33, 31], and more recently for ad hoc networks, see e.g., [28, 27, 30, 29]. In the mid 1990s, the development of QoS paradigms for the Internet, see e.g, [26, 25], led to a renewed interest in examining routing and spurred the development of a plethora of QoS routing algorithms, which mainly targeted distributed computation. In sharp contrast, the emergence of the SDN paradigm [21, 22] has shifted the research focus to centralized network control, including centralized routing computations [23, 24].
In section 3.2 we have summarized the routing algorithms of the state of the art. Out of this set of QoS routing algorithms we have to select the routing algorithm which maximizes the optimization goal (Equation 5.3) at an acceptable runtime. However, it is not clear that an optimal routing algorithm will also lead to a higher traffic intensity. Therefore, the routing algorithm selection has to be evaluated within the context of the entire QoS framework.

5.2. Cost Function design Problem

The cost behavior of the cost function influences the global optimality of a greedy routing system. Depending on the design of this cost function the overall blocking probability could be minimized. Only if the cost function expresses the negative impact of this path for future embeddings correctly, will this path lead to a higher network utilization. An optimal cost function needs information about the future requests, the DCLC Routing algorithm behavior,

the Resource Allocation algorithm and the network resource model. Only if all this aspects are considered in a cost function,can global optimality can be achieved. Unfortunately, there is no optimal cost function or even the possibility of proofing the cost functions optimality for big topologies. Therefore, we define seven cost functions within this section and discuss and evaluate their performance.
The general definition of the path dependent cost function $\mathrm{Cost}(P_f)$ leads to a full combinatorial search during the routing process [6]. Therefore, a reduction of the problem complexity is needed. We define $\mathrm{Cost}(P_f)$ as the flowing:

$$\mathrm{Cost}(P_f) = \sum_{(m,u,v,q)\in P_f} \mathrm{Cost}_x(m,u,v,q) \tag{5.5}$$

The cost function provides a cost value for a queue link edge. A queue link edge consists of the flowing parameter:

u source of an link

v destination of an link

q queue of an link

m link leading to the link

The considered cost functions consist of four basic values:

priority (p) provides the priority value of a Queue link from node u to node v via queue q [$\mathrm{p}(u,v,q)$]. The priorty value range is defined between 0 and $\mathrm{p}_{\max}$. 0 is the higest priority and $\mathrm{p}_{\max}$ the lowest.

maximal priority ($\mathrm{p}_{\max}$) provides the maximum priority value which is the value of the lowest priority Queue link from node u to node v via queue q [$\mathrm{p}_{\max}(u,v,q)$].

number of flows (nf) provides the number of flows traversing a physical link from node u to node v [$\mathrm{nf}(u,v)$], for a Queue link from node u to node v via queue q [$\mathrm{nf}(u,v,q)$] and for a Queue link from node u to node v via queue q from physical edge m [$\mathrm{nf}(m,u,v,q)$].

feasibility (fes) provides the feasibility value of a physical link from node u to node v [$\mathrm{fes}(u,v)$], a Queue link from node u to node v via queue q [$\mathrm{fes}(u,v,q)$] and a Queue link from node u to node v via queue q from physical edge m [$\mathrm{fes}(m,u,v,q)$]. The feasibility value is defined as the multiplier to the edge resource consumption on which the access control is still valid.

constant value is necessary to formulate complex cost functions.

These basic building blocks are used to build more complex cost functions. We group this cost functions into two classes. The first one is the class of the static cost functions. The second one is the class of the dynamic cost functions.

5.2.1. Static Cost Functions

Static cost functions are based on network metrics which are considered to be constant. These cost functions have the advantage that the computational effort is negligible. However, the capabilities of resolving bottleneck situations is also limited. In this section, we discuss three static cost functions.
Cost_1 (eq. 5.6) is the most simplistic cost function. At every edge the algorithm requests costs, the cost value one will be returned. A least cost algorithm will return the path which has the smallest possible hop count in the first place. If the hop count is identical, the edge with the highest priority on the first relaxed link will be taken due to the implementation of the algorithms. This is a approximation of a Least Delay Path (LDP) search.

$$\mathrm{Cost}_1(m, u, v, q) = 1 \tag{5.6}$$

Cost_2 (eq. 5.7) is a hybrid cost function. It finds the shortest path in terms of hop count if the priorities are the same. The queue with the highest priority comes with a cost of 2 since $\mathrm{p}(u, v, q)$ returns zero in that case. If the number of priority queues increases the cost value of the queue with the lowest priority converges to one. The maximum cost difference between a high and a low priority queue is one. This indicates that the algorithm can find paths with up to the double hop length of the shortest path.

$$\mathrm{Cost}_2(m, u, v, q) = 1 + \frac{1}{1 + \mathrm{p}(u, v, q)} \tag{5.7}$$

Cost_3 (eq. 5.8) is the linear version of Cost_2. The costs of a queue are between 1 and $\mathrm{p}_{\max}$. From this it is obvious that a $\mathrm{p}_{\max}$ times longer path will be used, if it consist only of low priority queues, to avoid high priority queues.

$$\mathrm{Cost}_3(m, u, v, q) = \mathrm{p}_{\max}(u, v, q) - \mathrm{p}(u, v, q) \tag{5.8}$$

5.2.2. Dynamic Cost Functions

Dynamic cost functions adapt their cost value based on state full variables of the network. Since this computation has to be performed online this cost functions

will have a higer computational complexity than the static cost functions. However, due to the dynamicity of the cost functions they potentially adapt better to bottleneck situations.
Cost_4 (eq. 5.9) finds the shortest path in terms of hop count by having at least a cost at one. The goal of the second part of the equation is to provide a high cost for edges heavily used and a low cost for ones that are nearly empty. This is achieved by calculating the distance ratio to the case where the resources are occupied. This is achieved by calculating $(\mathrm{fes}(u,v)-1)$. To normalize the values among all edges it is multiplied by the number of flows served by this edge ($\mathrm{nf}(m,u,v,q)$). The result is the additional average number of flows which will fit into this edge. The multiplicative inverse of average number of flows will provide low costs for empty edges and high costs for the full ones.
As a result the routing algorithm will use the path which contains the edges with the highest amount of free resources. Please note that Cost_4 is depending on the input link. In [6] we call this an M_1 metric. Please note that normal routing algorithms might become suboptimal if they use this cost function.

$$\mathrm{Cost}_4(m,u,v,q) = 1 + \frac{1}{(\mathrm{fes}(m,u,v,q)-1)\,\mathrm{nf}(m,u,v,q)} \tag{5.9}$$

To avoid the usage of a M_1 metric we define Cost_5 (eq. 5.10) and Cost_6 (eq. 5.11). They both folow the idea of Cost_4 but provide a smaller amount of cost values. Cost_5 (eq. 5.10) considers the resource usage of the whole physical link (u,v). This implies that all q-links of the physical link will have the same cost. Cost_6 (eq. 5.11) considers the resource usage of a q-link (u,v,q). In summary, all three cost functions (Cost_4,Cost_5 and Cost_6) share the same calculation principle, but execute it on a different complexity level.

$$\mathrm{Cost}_5(m,u,v,q) = 1 + \frac{1}{(\mathrm{fes}(u,v)-1)\,\mathrm{nf}(u,v)} \tag{5.10}$$

$$\mathrm{Cost}_6(m,u,v,q) = 1 + \frac{1}{(\mathrm{fes}(u,v,q)-1)\,\mathrm{nf}(u,v,q)} \tag{5.11}$$

Since Cost_4,Cost_5 and Cost_6 are pricing the edges of the graph based on the demand they are able to reflect bottleneck situations in there cost value. Constrained Shortest Path (CSP) algorithms using this cost functions will use the less costly non bottleneck edges, if possible.
We extend the pure feasibility based cost function Cost_5. In addition to the search for the non bottleneck physical edges we use the high priority queues first. This behavior is realized with Cost_7. The idea for this cost function is based on the experiments made trough the evaluations.

$$\text{Cost}_7(m,u,v,q) = 1 + \frac{1}{(\text{fes}(u,v)-1)\,\text{nf}(u,v)(\text{p}_{\max}(u,v,q)-\text{p}(u,v,q))} \tag{5.12}$$

5.2.3. Discussion

The impact of the cost functions is highly dependent on the used routing algorithm (Section 5.1) and the used resource model (Section 4). Indeed the different suboptimal behaviors of routing algorithm could lead to an different impact on the performance of a cost function. The implications of the resource modeling is even more obvious. A resource model which does not consider input link shaping is not able to benefit from an input link dependent cost function. To deal with this, an evaluation is needed which compares all possible settings of cost functions, routing algorithms, resource models and Resource allocation algorithms in different network scenarios. This evaluation is extensively done in Section 6.

5.3. Resource Allocation Problem

With our network resource model, the resource allocation can readily be executed periodically in the background to optimize the queue link configuration of each physical link. Generally, a potential resource allocation algorithm has to optimize the allocation of the resource for the different models. For the Multi-Hop Model (MHM) the allocated rate $A^R_{(u,v,q)}$ and the allocated burst $A^B_{(u,v,q)}$ have to be adapted. The Threshold-Based Model (TBM) needs only adjustments of the allocated worst case delay bound $A^{WCD}_{(u,v,q)}$. We refer to the set of adjustable parameter as $\mathcal{A}$. The resource allocation algorithm should change this parameters to avoid bottleneck situations of the network resources. Bottlenecks could arise for each of the three types of resources:

- data rate, indicated through the utilized mean transmission bit rate $U^R_{(u,v,q)}$
- buffer space, indicated through the utilized buffer $U^B_{(u,v,q)}$
- delay limit utilization, indicated through the difference between the delay limits t_f and the worst-case path delays $D(P_f)$.

For an actual network operating with the proposed function split approach, only the online routing needs to be executed when a new flow requests admission. As the arrivals of new flow requests or departures of existing flows occur typically on a slow time scale, significant changes in the resource utilization of the queue

links occur only slowly. Thus, executing the resource allocation periodically in the background will typically be sufficient for conducting the routing (and admission control) with a resource allocation that is (nearly) up-to-date and close to optimal for the currently carried flows.
Please notice that changes in the allocation parameter are only allowed if they do not violate the buffer constraint (MHM: Equation 4.25; TBM: Equation 4.30) of the network resource models so that the packet loss constraint (Equation 2.3)

$$L(P_f) = 0 \quad \forall f \in \mathcal{F}$$

stays valid. In addition, it is only allowed to do changes which still ensure the end-to-end delay bound (Equation 2.2)

$$D(P_f) \leq t_f \quad \forall f \in \mathcal{F}.$$

The optimal solution of the general resource allocation problem is beyond the scope of this thesis. However, in order to illustrate a possible resource allocation approach for our overall function split QoS framework, we propose an elementary resource (rate) allocation algorithm and the Tunable Resource Allocation Algorithm (TRAA) in the next subsections.

5.3.1. Elementary Resource Allocation Algorithm

Initially, for the commencement of operation of a network, link rates are allocated according to historic traffic observations for similar networks and predictions of traffic patterns, similar to [154]. Then, the resource allocation algorithm in Figs. 5.2 and 5.3 is periodically employed in the background (offline) to sequentially examine rate allocations at the individual physical connections (u, v) element of the set of physical connections $\mathcal{E}^p$. The current state of the network resource model (Section 4) is copied over to a virtual network resource model, which is then used (in the background) for executing the resource allocation algorithm.
The brute-force search in Line 10 of Fig. 5.2 is over a prescribed reasonably large set $\mathcal{AP}$ of rate allocations $A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$ to all the queue links $q \in \mathcal{E}^q_{(u,v)}$ traversing the considered physical link (u, v). For instance in our numerical evaluations in Section 6.2.3 we consider the allocation of the total physical transmission bit rate 1 Gb/s in 5 Mbyte/s steps to the individual queue links.
For a given considered rate allocation to all the queue links $A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$, the algorithm in Fig. 5.3 estimates the number of flows that can be supported in two main steps. In a first step (Lines 2–4) the already admitted flows are added one-by-one to the virtual network resource model. If all admitted flows can still be

1: **procedure** RESOURCEALLOCATION
2: **for all** $(u, v) \in \mathcal{E}^p$ **do**
3: Virtual Netw. Res. Model $\leftarrow$ Netw. Res. Model
4: Execute Steps on Virtual Netw. Res. Model
5: $\mathcal{F}_{(u,v)}$ = Set of admitted flows traversing (u, v)
6: **for all** $f \in \mathcal{F}_{(u,v)}$ **do**
7: DEREGISTERPATH(P_f)
8: $F_{\max} = 0$
9: **for all** $A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}} \in \mathcal{AP}$ **do**
10: $F =$ FLOWCOUNTEST$(A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}})$
11: **if** $F > F_{\max}$ **then**
12: $F_{\max} = F$
13: $A^R_{\max} = A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$
14: **if** $F_{\max} > |\mathcal{F}_{(u,v)}|$ **then**
15: Netw. Res. Model$\leftarrow A^R_{\max}$
16: **for all** $(u, v, q) \in \mathcal{E}^q_{(u,v)}$ **do**
17: Evaluate $WCD^{MHM}_{(u,v,q)}$ from Eqn. (4.26)
18: Evaluate $A^B_{(u,v,q)}$ from Eqn. (4.25)
19: **for all** $f \in \mathcal{F}_{(u,v)}$ **do**
20: DEREGISTERPATH(P_f),
21: REGISTERPATH(DOROUTING(f))

Figure 5.2.: Resource allocation algorithm executed periodically in background: The algorithm cycles through all physically links (u, v) of the network. The current real network resource model is first copied over to a virtual network resource model to examine candidate resource (rate) allocations $A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$ to the set $\mathcal{E}^q_{(u,v)}$ of queue links traversing physical link (u, v). The candidate allocations $A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$ are from a prescribed set $\mathcal{AP}$. The resource allocation $A^R_{\max}$ supporting the highest number of flows, as determined by the flow count estimation algorithm in Fig. 5.3, is implemented in the real network resource model (provided $A^R_{\max}$ supports more flows than the current allocation).

accommodated, the second step (Lines 5–9) estimates the number of additional flows that the node can carry with the considered rate allocation. For this estimation, new flows are generated from a statistical flow model. The statistical flow model could, for instance, average the flow model parameter values (see Table 4.1) of the admitted flows $\mathcal{F}_{(u,v)}$ or uniformly randomly draw each added flow from $\mathcal{F}_{(u,v)}$ (which is considered in our evaluations in Section 6.2).

We note that the resource allocation algorithm in Fig. 5.2 always starts from a valid state since the admission control has only admitted flows whose delay limits can be met with the currently considered network resource allocation. The

```
1: procedure FLOWCOUNTEST((A^R_{(u,v,q)∈A_(u,v)})
2:     for all Admitted flows f ∈ F_(u,v) do
3:         if REGISTERPATH(DOROUTING(f)) = False then
4:             return 0
5:     FlowCount = |F_(u,v)|
6:     New flow n = StatModel(F_(u,v))
7:     while REGISTERPATH(DOROUTING(f)) = True do
8:         New flow n = StatModel(F_(u,v))
9:         FlowCount++
10:    return FlowCount
```

Figure 5.3.: Flow count estimator algorithm for testing a candidate resource (rate) allocation $A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$: The admitted flows that are currently traversing the examined physical link (u, v) are added one by one to the virtual network resource model. After all admitted flows have been successfully added, additional new flows are generated according to a statistical model of the flows currently traversing the link (u, v). The total number of flows (already admitted flows plus new flows from the statistical model) that the node can accommodate subject to the QoS constraints is returned to the resource allocation algorithm in Fig. 5.2.

search for a new rate allocation pattern on the virtual network resource model delivers a pattern that supports at least the currently admitted flows (and potentially additional new flows). Thus, already admitted flows are guaranteed to still have a valid path when a new resource allocation pattern is adopted in Line 15 of Fig. 5.2. Line 17–18 uses Eq. 4.26 to update the queue link delay budgets $WCD^{MHM}_{(u,v,q)}$, which are used to evaluate the end-to-end delay requirement $D(P_f)$. In addition the allocated buffer capacity $A^B_{(u,v,q)}$ is updated with Eq. 4.25. The flows are shifted to their valid paths with the new resource allocation pattern through executing the flow removal and addition in Line 24 of Fig. 5.2. Running this resource allocation algorithm periodically in the background following the general strategies for periodic background processes in networks [155] adapts the resource allocation to the admitted traffic flows and, if possible, allows for the admission of new flows.

5.3.2. Tunable Resource Allocation Algorithm (TRAA)

There is a fundamental trade-off within the resource allocation problem. Utilizing links higher will cause also higher delays. A valid strategy to handle this trade-off properly is to increase the delay border, enabling a higher utilization at bottleneck links. This will lead to a higher utilization at the bottlenecks in the network. On the other hand a delay decrease at the low utilized links should be performed to enable a low end-to-end delay. In addition to this, a rebalancing of the resources is needed. For example the MHM could have some burst left even

1: **procedure** RESOURCEALLOCATION
2: S_{pl} get set of all physical links
3: sort S_{pl} by utilization
4: S_{hu} is the $TRAA_{er}$ percent of the high utilized links from S_{pl}
5: S_{lu} is the $TRAA_{er}$ percent of the low utilized links from S_{pl}
6: **for all** S_{lu} **do**
7: $\mathcal{A}$ = Optimize Equations 5.13 with -$TRAA_{dv}$
8: update $\mathcal{A}$ parameter at model
9: **for all** S_{hu} **do**
10: $\mathcal{A}$ = Optimize Equations 5.13 with -$TRAA_{dv}$
11: update $\mathcal{A}$ parameter at model

Figure 5.4.: Tunable Resource Allocation Algorithm (TRAA) listing

if all allocated rate is consumed. This burst will cause queuing delay even if it is impossible to consume it. A resource allocation scheme has to eliminate such kind of situations.

Figure 5.4 shows the algorithm listing of the Tunable Resource Allocation Algorithm (TRAA). The algorithm should be called if $TRAA_{nf}$ flows were embedded since the last execution. The algorithm itself sorts all the physical links by its utilization. This sorted set is used to get the $TRAA_{er}$ present of the high and low utilized links. The first optimization step is the decrease of the delay level for the low utilized links and the update of the parameter. Since the delay deviation of flows $\mathcal{F}$ is increased the increase of the delay level at the high utilized links could be done.

To solve this challenges two optimization problems are defined. Problem formulation 5.13 shows how the resource balancing and the change of the link delay level is realized. The first step is the definition of the optimization goal. The goal is the maximization of the link feasibility by varying the corresponding set of resource allocation parameter. This will optimize the resource balancing. The second step is to check that the changes of the set of resource allocation parameter do not lead to an violation of the end-to-end delay constraint t_f. The next constraint limits the overall movement of the delay value by the configurable parameter $TRAA_{dv}$. If this value is positive the link delay level will be increased. A negative $TRAA_{dv}$ will decrease the link delay level. The last constraint defines the problem as unfeasible if no improvement of this link is possible.

$$\begin{aligned} &\max_{\mathcal{A}} \operatorname{fes}(u,v) \\ &D(P_f) \leq t_f | P_f \in \mathcal{F} \\ &\sum_{q \in \mathcal{E}^q_{(u,v)}} \frac{D(u,v,q,\mathcal{A})}{D(u,v,q)} \leq (1 + TRAA_{dv}) |\mathcal{E}^q_{(u,v)}| \\ &1 < \operatorname{fes}(u,v) \end{aligned} \tag{5.13}$$

The impact of the TRAA is highly dependent on the used routing algorithm (Section 5.1) and the used cost function (Section 5.2). Indeed the different suboptimal behaviors of routing algorithms could lead to a different impact on the performance of the TRAA. To deal with this an evaluation is needed which compares all possible settings of cost functions, routing algorithms, resource models and Resource allocation algorithms in different network scenarios. This evaluation is extensively done in Section 6.

6. Evaluation

The design of the industrial Quality of Service (QoS) framework solves already some parts of the problem statement. The Deterministic end-to-end delay guarantees (Problem 1), the unrestricted topologies (Problem 4) and the Cross traffic (Problem 6) tolerance of the framework is covered by the deterministic network resource models. The function split tempts to enable an online usage (Problem 2) of the industrial QoS framework. However, a detailed performance analysis which proofs the runtime behavior (Problem 2) , the flow capacity (Problem 3) and the sensitivity to topology scaling (Problem 7) is missing. This gap will be closed by this chapter.

Since the three common Software Defined Networking (SDN) controller (Floodlight, Opendaylight and ONOS) are implemented in Java, we use Java for implementing the industrial QoS framework, too. A component based entity system was used as the general design pattern. This pattern enables to program modular functionalities, which is well suited for implementing the different network resource models.

In total three evaluations are presented. The function split uses Constrained Shortest Path (CSP) algorithms for reducing the problem complexity. So the CSP algorithms performance will influence the performance of the industrial QoS framework. In Section 3.2 all suitable CSP algorithms are surveyed. However, since the state of the art did not provide any complete performance comparison of this CSP algorithms an algorithm selection could not be performed directly. In Section 6.1 this gap will be closed by simulation.

Since the Function Split is a simplification of the original problem the results of the Function Split based industrial QoS framework could be suboptimal. In Section 6.2 the the performance impact of the complexity reduction of the function split are evaluated against a Mixed Integer Program (MIP) solution.

There are many cross dependencies between the different implementation aspects of the industrial QoS framework. An overall performance analysis of the entire QoS framework with finally all its configuration options is provided in Section 6.3. Within this section a method, solving the algorithm selection problem for the industrial QoS framework is provided. Finally the best framework configuration is used to judge if the framework provides sufficient performance for the industrial use case.

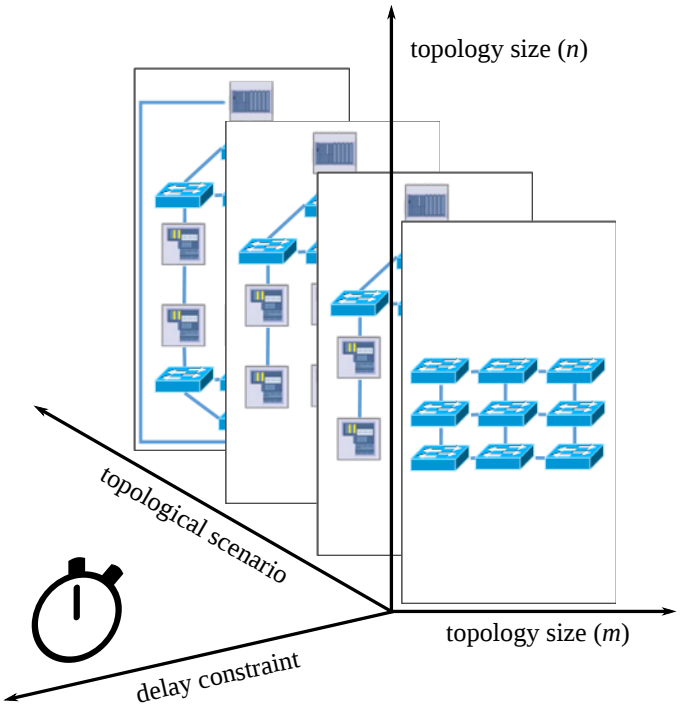


Figure 6.1.: Illustration of four dimensions of performance evaluation framework for delay-constrained least-cost (DCLC) routing algorithms: type of topology, scaling of the network (topology) in two dimensions, and delay constraint.

6.1. CSP Algorithm Evaluation

In Section 3.2 we surveyed the stat of the art of CSP routing algorithms. In total 26 algorithms where identified (see Table 6.1). However, no comparison of all these algorithms is available in the state of the art. To close this gap a large-scale evaluation based on a consistent Java based re-implementation of all the studied algorithms is performed in this section.
We classify the unicast QoS routing algorithms according to the underlying routing strategy into several main categories, including priority queue based algorithms, Bellman-Ford based algorithms, Lagrange relaxation based algorithms, as well as algorithms that follow the least-cost and least-delay paths (see Table 6.1).
In order to facilitate a comprehensive evaluation of QoS routing algorithms, we introduce a four dimensional (4D) evaluation framework, as illustrated in Fig. 6.1. The first dimension corresponds to the type of topology. The second and third dimensions correspond to the scaling of a given type of topology into two dimensions that characterize the "size" of the network. The fourth dimension corresponds to the tightness of the delay constraint.
The comprehensive evaluation of the existing QoS routing algorithms with this

Table 6.1.: Comprehensive list of Constrained Shortest Path (CSP) and Multi-Constrained Shortest Path (MCSP) algorithms, which can be employed for Delay-Constrained Least-Cost (DCLC) QoS routing. The algorithms are categorized according to the underlying algorithmic strategy into algorithms based on priority queues, Bellman-Ford (BF), Lagrange relaxation, as well as Least-Cost (LC) and Least-Delay (LD) paths. For each algorithm, we indicate the type(s), i.e., CSP or Multi-Constrained Shortest Path (MCSP)or k path versions thereof, as well as other key characteristics, including optimality property and the accepted parameters. We indicate the number of underlying algorithm runs, e.g., Iterative k Shortest Path (IkSP) and Static k Shortest Path (SkSP) algorithms (see Section 3.2.1 for definitions). When the exact number of runs depends on the specific scenario, the possible numbers of runs are indicated through a comma-separated list or a range (with the arrow ($\rightarrow$) symbol) within parentheses. Unbounded numbers of runs are indicated with the greater or equal ($\geq$) sign. We note that an algorithm using a Static k Shortest Path (SkSP) algorithm can be implemented with an Iterative k Shortest Path (IkSP) algorithm.

Algorithm	Type	Number of runs of underlying algorithms IkSP	SkSP tree	SkSP SP tree	SP	Optimal	Complete	Distr.	Param.
Elementary Algorithms (Sec. 3.2.4.1)									
LDP	CSP				1		✓	if SP is	
FB [98]	CSP, MCSP				$(1 \rightarrow M+1)$		if CSP		
kMCSP	(k)CSP, (k)MCSP	1				✓	✓		
Priority Queue Based Algorithms (Sec. 3.2.4.2)									
CBF [129]	CSP					✓	✓		
A*Prune [89]	(k)CSP, (k)MCSP					✓	✓		
Algorithms Based on BF (Sec. 3.2.4.3)									
DCBF [88]	CSP			1			✓	✓	
kDCBF [88]	CSP		1				✓	✓	k_d, k_c
DEB [105]	CSP						✓		
Algorithms Based on the Lagrange Relaxation (Sec. 3.2.4.4)									
LARAC [68, 69, 70, 71]	CSP				≥ 1		✓		MD
LARACGC [69]	CSP	(0, 1)			≥ 1	if $\delta = 0$	✓		δ
SCRC [100]	CSP	(0, 1)			(1, 2)	✓	✓		
kLARAC [88]	CSP		≥ 1				✓		k
H_MCOP [59]	CSP, MCSP			1	(0, 1)		if CSP		λ
kH_MCOP [59]	CSP, MCSP		(0, 1)	1			if CSP		λ, k
NR_DCLC [91]	CSP			≥ 0	≥ 1		✓		
MH_MCOP [107]	CSP, MCSP			$(1 \rightarrow H+1)$	$(0 \rightarrow H+1)$		if CSP		H
E_MCOP [107]	CSP, MCSP	$(0 \rightarrow 2)$			$(1 \rightarrow M+1)$	✓	✓		
DCCR [99]	CSP		(0, 1)		(1, 2)		✓		k
SSR+DCCR [99]	CSP		(0, 1)		$(1 \rightarrow L+2)$		✓		L, k
Algorithms Based on LC and LD Paths (Sec. 3.2.4.5)									
DCUR [92, 93]	CSP			(1, 2)			✓	✓	
DCR [94]	CSP			(0, 1)	1		✓	✓	
IAK [96]	CSP			1	(0, 1)		✓	✓	
SMS-RDM [95]	CSP			1			if $p \geq \Delta(G)$	✓	p
SMS-CDP [95]	CSP			2			if $p \geq \Delta(G)$	✓	p
SMS-PBO [95]	CSP						if $p \geq \Delta(G)$	✓	p
SF_DCLC [97]	CSP			(1, 2)			✓	✓	

novel 4D evaluation framework provides valuable insights into the behaviors of the algorithms. Our evaluation has yielded a very large data set; we only present the most significant and insightful evaluation data in this article. We have made the entire evaluation data publicly available at [158], an interactive web interface that allows for convenient navigation through the 4D evaluation space.
We have observed from our evaluations that it is not possible to elect an algorithm as "the best QoS routing algorithm". Indeed, we show that the performance of the algorithms strongly depends on the considered specific sub-space of the 4D evaluation space. Nevertheless, we identify two algorithms (out of a total of 26 compared algorithms) that achieve the best cost-runtime trade-off for most cases. Furthermore, we observe the general trend that algorithms based on a shortest path (SP) algorithm have shorter runtimes than algorithms based on a shortest path tree (SP tree) algorithm, which in turn have shorter runtimes than algorithms relying on a k shortest path (kSP) algorithm to reach a given optimality level. The content of this section is manly based on [4].

6.1.1. Four-Dimensional (4D) Evaluation Framework

Generally, the performance of an algorithm depends on the specific scenario in which it is executed. In order to evaluate the behaviors of the different algorithms across a wide set of scenarios, we introduce an evaluation framework that evaluates QoS routing algorithms along four critical dimensions. First, we define four *topologies*, which we describe in Sec. 6.1.1.1. A topology describes both the underlying structure of the network and the nodes that communicate with each other in the network. Second and third, we scale these topologies in two directions. Fourth, we distinguish requests based on the level of strictness of the delay constraint, see Sec. 6.1.1.2. Sec. 6.1.1.3 presents the evaluation procedure and the metrics used, while Sec. 6.1.1.4 identifies the evaluated algorithms.

6.1.1.1. Topology and Scaling

As first dimension of our evaluation framework, we define four topologies (shown in Fig. 6.2) based on three different base topologies. Although our survey is generic and all the algorithms can be applied to any CSP problem, we focus on industrial topologies where we expect centralized QoS routing to be extensively employed [3]. Nevertheless, the topologies we define are also common in and representative of data center, metro, grid, and enterprise networks. On the contrary, wide-area (star topology) networks are not covered, as strict centralized QoS routing in such environments is unlikely. All topologies can be scaled according to two scale parameters m and n that represent the

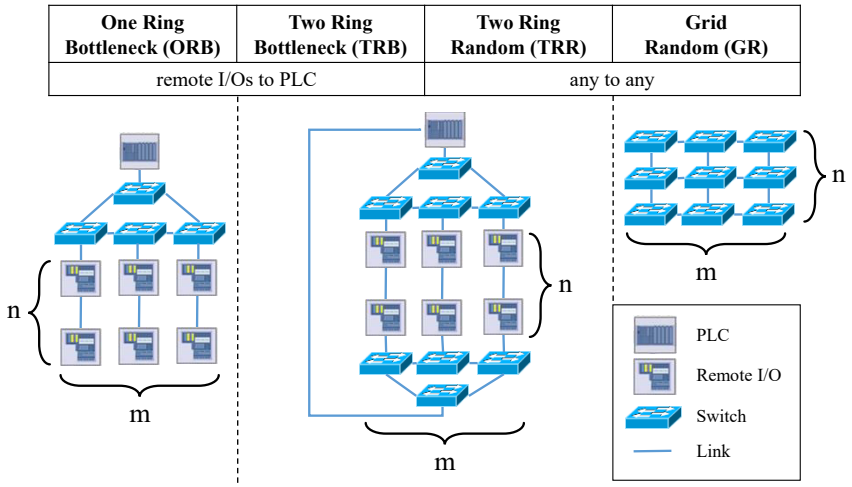


Figure 6.2.: The four topologies considered in the evaluation are based on three different base topologies which can be scaled in two different directions.

number of communicating nodes and switches as defined in the following for the four different topologies. The second and third dimensions of our evaluation framework correspond to varying the two scale parameters m and n from 4 to 13, thereby defining 100 different scalability levels. The four topologies are referred to as One Ring Bottleneck (ORB), Two Ring Bottleneck (TRB), Two Ring Random (TRR) and Grid Random (GR).

One Ring Bottleneck (ORB) The ORB topology consists of a base ring of $m + 1$ switches. A so-called Programmable Logic Controller (PLC) is connected to one switch of this ring. A branch composed of a series of n remote input/output nodes (I/Os), e.g., sensors and actuators, is connected to each of the other m switches of the ring. Thus, there are a total of mn I/Os. Remote I/Os have an internal switch allowing traffic to flow along the branches. Traffic is only considered from the remote I/Os to the PLC.

Two Ring Bottleneck (TRB) The TRB topology extends the ORB topology with an additional ring consisting of $m + 1$ switches. The $m + 1$ switches connect the loose (bottom) ends of the m branches of remote I/Os (of the ORB topology) to the PLC. Traffic is still considered only from the remote I/Os to the PLC.

Two Ring Random (TRR) The TRR topology is the same as the TRB topology, but traffic is now considered between any pair of remote I/Os. As the remote I/Os, the PLC is able to forward traffic not destined for it.

Grid Random (GR) The GR topology is a grid of width m and height n. In the GR topology, traffic is considered between any pair of nodes.

We do not consider random topologies generated based on models, such as the Waxman model [106]. Instead, striving for a fair and reproducible evaluation, we only use deterministic topologies.
Each directed link is considered to have four output priority queues and routing is then performed on the corresponding queue-link topologies. For each physical link, the costs of the four queue-link edges with priority levels $p,\ p = 1$ (high priority), $2, 3, 4$ (low priority), are set to the values $1 + 1/p$ so as to favor the usage of low priority queues. The delay values are obtained with Schmitt's formula 4.6. Thus, the costs and delays of the four queue-link edges are respectively set to 2 and 0.48 ms, 1.5 and 1.26 ms, 1.33 and 2.83 ms, as well as 1.25 and 7.55 ms.
Clearly, the number of queues as well as the cost and delay settings influence the performance of the algorithms and could be defined as additional comparison dimensions. However, in order to keep the evaluation tractable, we keep them static.

6.1.1.2. Delay Constraint Tightness

The delay constraint of routing requests can range from loose values for which the LC path is feasible to tight values for which no feasible path exist. Within this range, we define seven subranges of equal size, which we refer to as *delay levels*. The fourth dimension of our evaluation framework corresponds to varying the delay constraint of routing requests between these different delay levels.

6.1.1.3. Evaluation Procedure and Metrics

Each algorithm is evaluated along the four dimensions of our evaluation framework. For each particular topology and combination of the scale parameters m and n, we sequentially simulate 20,000 routing requests. The first 1000 requests are used as warm-up for the Java HotSpot optimizer and their results are not considered. For each request, the source and destination are generated uniformly randomly from the possible set of combinations defined by the topology and scale parameters. The delay constraint is distributed uniformly randomly among the seven delay levels [and then uniformly randomly within the selected delay level (delay constraint subrange)] so as to prevent the Java HotSpot optimizer from optimizing for a specific delay level. (If all test runs for a specific delay level are run successively, the Java HotSpot optimizer could exploit the consideration of a particular delay level in successive runs. This could happen because of the online code analyses done by Java HotSpot. The online compiler might find some optimization opportunities which are only possible when the delay level stays equal. [161])

For a given Algorithm Under Test (AUT) and request (source, destination and delay level), we run three algorithms. First, we run Constrained Bellman-Ford (CBF) in order to obtain the cost z_{opt} of the optimal solution. Second, we run the AUT to determine the AUT cost z'. The Cost Inefficiency (CI) of the AUT is then evaluated in % compared to the cost of the optimal path according to Eqn. (3.3). Third, we run an LD search using A* (which is then equivalent to an Least Delay Path (LDP) search). We define the runtime of the AUT divided by the runtime of the LD search as the runtime ratio of the AUT. This normalization allows to filter out runtime variations due to the varying load on the testing machines. Indeed, both algorithms are run one after the other, i.e., within a short time window during which the load on the testing machine can be assumed to be constant.

6.1.1.4. Algorithms Selection

Table 3.2 summarizes the algorithms that we have identified as suitable for the considered unicast QoS routing in Sec. 3.2.4. We implemented all these 26 algorithms in Java 8[1] and, for each of them, ran our evaluation procedure. The specific parameter settings for parameterized algorithms will be given in Sec. 6.1.2. We will identify parameterized algorithms by the name of the original algorithm to which we append the dash-separated parameter values in the same order as in Table 3.2. For example, Lagrange Relaxation based Aggregate Cost Gap Closing (LARACGC) with $\delta = 25\%$ will be referred to as *LARACGC-25*. We omit the λ parameter of Heuristic for Multi-Constrained Optimal Path (H_MCOP) and k Heuristic for Multi-Constrained Optimal Path (kH_MCOP) since it has no influence in the CSP case.

6.1.2. Evaluation Results

Sec. 6.1.2.1 presents the evaluation results for the fourth dimension, i.e., the behavior of the algorithms for the different delay levels. Sec. 6.1.2.2 then focuses on the three first dimensions. Due to the high number of algorithms and the highly detailed results on how they behave and perform, it is not possible to present and discuss all results for all algorithms in detail in this thesis. Therefore, we only present the most interesting algorithms and discuss the most important conclusions. We have made the entire set of raw results and graphs for all the algorithms publicly available at `http://www.lkn.ei.tum.de/lora` [158]. We found that k Multi-Constrained Shortest Path (kMCSP), A*Prune, LARACGC, Santos Coutinho-Rodrigues Current (SCRC), Exact Multi-Constrained Optimal

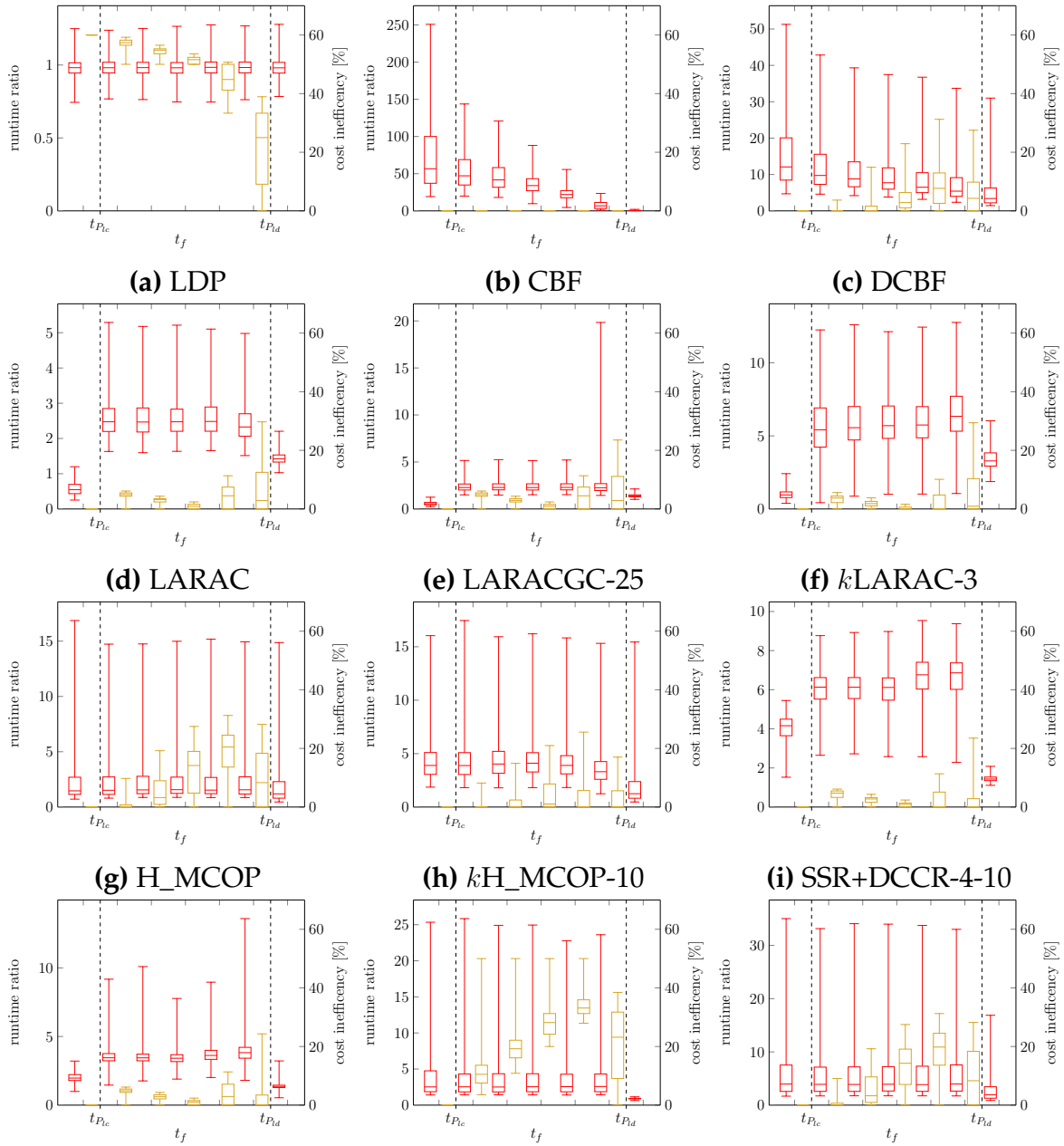
[1] We acknowledge that the results may be subject to our specific implementations; however, we tried to be fair and optimize all implementations as much as we could.

Path (E_MCOP), and the three Sriram Manimaran Siva (SMS) variations were not able to complete the evaluation in a reasonable amount of time compared to CBF. This leads to our first observation that algorithms using an IkSP algorithm to reach optimality have a very long runtime. Indeed, the considered queue-link topologies are dense with high numbers of possible paths. Thus, the number of paths to discover until reaching optimality is also high, yielding intractable runtimes for kMCSP, LARACGC, SCRC, and E_MCOP. A*Prune and SMS are not based on an IkSP algorithm but their structure is such that, if their initial search direction is not the correct one, they have to explore a high number of paths to reach the destination. The negative impact of this approach is accentuated by the high density of the considered queue-link topologies.

6.1.2.1. Fingerprints: Influence of the Delay Constraint Tightness

We analyze the fourth dimension using so-called *fingerprint* graphs (Fig. 6.3). The fingerprint graph for a given combination of topological and scale parameters m and n, shows the distribution of the runtime ratio (left, in red) and CI (right, in yellow) of an algorithm for the seven different delay levels (loose levels on the left and tight levels on the right). Since we have four different topologies with 100 different scalability levels (combinations of m and n values), each algorithm has 400 fingerprint graphs. Nevertheless, we observed that the shapes of all fingerprint graphs for a given algorithm are similar; Fig. 6.3 shows fingerprints for the GR topology with scale parameters $m = n = 10$. Since the shapes of these graphs characterize the different algorithms we refer to these graphs as *fingerprints*: they nearly uniquely identify an algorithm based on its behavior and are (nearly) always the same for a given algorithm. Only the absolute values vary depending on the topology and its scaling. These variations will be discussed in Sec. 6.1.2.2.

Elementary Algorithms Since the elementary LDP algorithm (see Section 3.2.4.1) does not take cost into account, its CI is the benchmark for the worst acceptable CI (Fig. 6.3a). As expected, the CI of LDP gets better for tighter constraints since the LD path becomes closer to the optimal solution. In terms of runtime, as LDP is compared with itself, the LDP fingerprint shows that the accuracy of our runtime metric is reasonable (the 0.5 and 99.5 percentiles are reasonably close to one and the median is approximately one). In additional evaluations, which we cannot include due to space constraints, we observed that Fallback (FB) exhibited, as expected, exactly the same CI behavior as LDP; except when the LC path is feasible, where FB is optimal, at the cost of one additional Shortest Path (SP) run for all other cases.



(a) LDP **(b)** CBF **(c)** DCBF

(d) LARAC **(e)** LARACGC-25 **(f)** kLARAC-3

(g) H_MCOP **(h)** kH_MCOP-10 **(i)** SSR+DCCR-4-10

(j) SSR+DCCR-2-5 **(k)** DCR **(l)** SF_DCLC

Figure 6.3.: Fingerprints for selected QoS routing algorithms. These graphs show, for the GR topology with $m = n = 10$, i.e., 10×10 switching nodes, the runtime ratio (runtime of algorithm normalized by runtime of LD search, plotted in red on left) and cost inefficiency (in yellow on right) of the algorithms for the seven different delay levels t_f (delay constraint subranges, whereby loose delay constraints are on the left and tight delay constraints are on the right). $t_{P_{ld}}$ and $t_{P_{lc}}$ denote the delays of the LD and LC paths, respectively. Since the rightmost delay level corresponds to an infeasible problem (delay constraint t_f lower than the delay $t_{P_{ld}}$ of the LD path), no cost inefficiency value is shown and the runtime then corresponds to the time required to detect that the problem is infeasible. While the cost inefficiency scale is the same for all the algorithms, the runtime scales have to differ because of the high variability between the algorithms. The lower and upper whiskers of the boxplots, respectively, correspond to the 0.5% and 99.5% percentiles.

Priority Queue Based Algorithms The benchmark for the highest acceptable runtime ratio is given by CBF, an optimal algorithm based on a priority queue (Section 3.2.4.2) (Fig. 6.3b). CBF was the fastest optimal algorithm. Since CBF terminates when the paths it expands have delays higher than the constraint, it terminates earlier for tighter constraints and its runtime therefore improves as the delay constraint gets tighter. The CI of CBF is always zero since CBF is optimal.

Algorithms Based on BF The BF based Delay-Constrained Bellman-Ford (DCBF) algorithm (Fig. 6.3c) has a CI fingerprint with slightly decreasing runtimes and increasing CI for increasingly tight delay constraints. For tight delay constraints, the delay test during the BF run fails more often and hence allows BF to terminate earlier (as it stops when no relaxation occurs in an iteration) and DCBF therefore gets faster as the delay constraint gets tighter. In additional evaluations we have observed that k Delay-Constrained Bellman-Ford (kDCBF) (not included in Fig. 6.3) has similar shapes, however which much lower CIs and longer runtimes. For example, kDCBF-2, divides the CI by a factor of approximately two, but increases runtime by a similar factor.

Algorithms Based on Lagrange Relaxation Similar to FB, Lagrange Relaxation based Aggregate Cost (LARAC) (Fig. 6.3d) can find the optimal solution with one LC search when the LC path is feasible. Fig. 6.3d (for the left-most, i.e., loosest delay constraint level t_f) shows that this run is roughly two times faster than an LD search. This is due to the fact that the delay and cost values have ranges of different absolute sizes and the guess function of A* is better for the costs because the costs have a smaller range size than the delay values, i.e., have a range size closer to the one of the least-hop count used for the guess (which is zero). When the problem is infeasible, LARAC notices the infeasibility with an additional LD search. For intermediate delay levels, LARAC requires a few additional SP runs, hence leading to slightly higher runtimes. Nevertheless, these additional runs are worth it as we can observe that the CI of LARAC is then pretty low.
While LARACGC did not complete the evaluation within a reasonable amount of time, LARACGC with $\delta = 25\ \%$ (Fig. 6.3e) did. As LARAC has a CI higher than 25% only for the tightest feasible delay level, LARACGC-25 only behaves differently than LARAC for this tightest feasible delay constraint. As expected, LARACGC-25 then brings the CI to less than 25% but at a high runtime cost even for such a small gap closing (as the CI of LARAC is at most 30%). This indicates that the gap closing is expensive in terms of runtime and probably not worth it. The high runtime is likely due to dense queue-link topology structures of our

evaluation networks and confirms that algorithms based on an IkSP algorithm are not efficient for dense network topologies.
We observe from Fig. 6.3f that k Lagrange Relaxation based Aggregate Cost (kLARAC) with $k = 3$ has the same shape as LARAC, however with longer runtime and lower CI. This is expected, since kLARAC runs an k Shortest Path (kSP) at each iteration, allowing to find lower cost paths but with longer runtime. We nevertheless observe that this cost reduction comes with a much less pronounced runtime increase than for LARACGC.
The fingerprint of H_MCOP (Fig. 6.3g) shows the difference in runtimes between SP searches and SP tree searches. Indeed, for detecting an infeasible problem, H_MCOP first computes a reverse SP tree. As can be seen, this has a much longer runtime than the single LD search of LDP. More precisely, the H_MCOP median runtime is only slightly longer, but the 99.5% percentile of the runtime is much higher than for LDP. This shows that comparing the runtime of algorithms in terms of "Dijkstra runs" independently of whether these are SP or SP tree runs, as done in some papers, is not a valid metric. For all other cases, H_MCOP requires an additional forward SP search. The H_MCOP runtime for these delay levels is hence always similar and slightly higher (by 0.5 since it is an LC search) than for the infeasible delay level. In terms of CI, H_MCOP interestingly presents a fingerprint of different shape than LARAC, LARACGC-25, and kLARAC-3. While the different LARAC versions have a U-shaped CI fingerprint, H_MCOP reaches higher CI for problems with tighter constraints but improves again for the tightest feasible delay level. In terms of absolute values, the CIs of H_MCOP are usually slightly worse than for the different LARAC versions, except when the delay constraint is loose, where H_MCOP and LARAC perform similarly. When using Chong's algorithm with $k = 10$ (Fig. 6.3h), we see that the runtime is only slightly increased while the CI is substantially improved. Indeed, kH_MCOP-10 reaches optimality in nearly 50% of the cases.
In additional evaluations we found that Nonlinear Relaxation Delay Constraind Least Cost (NR_DCLC) (which is not shown in Fig. 6.3) has a similar, but slightly better, CI fingerprint compared to H_MCOP; which is expected since NR_DCLC uses Heuristic for Multi-Constrained Path (H_MCP) (i.e., H_MCOP) as underlying algorithm. On the other hand, the NR_DCLC runtime is much longer, except in the cases where the LC path is feasible or where the problem is infeasible, in which cases NR_DCLC uses SP runs to detect these situations. Within the feasible delay levels, the runtime of NR_DCLC gets shorter as the delay constraint gets tighter. Indeed, NR_DCLC starts with an LD search and then improves on this path. When the delay constraint gets tighter, the LD path is closer to the optimal solution and NR_DCLC hence has less work to do. Additional evaluations have shown that Modified Heuristic for Multi-Constrained Optimal Path (MH_MCOP) (which is not shown in Fig. 6.3) improves the CI of H_MCOP

by a factor of around two at the expense of a twofold runtime increase. The H parameter can then be used to tweak the CI/runtime trade-off. While the CI fingerprint of MH_MCOP is similar to the one of H_MCOP, the MH_MCOP runtime fingerprint exhibits a Gaussian bell curve shape. This is due to the fact that MH_MCOP improves on the solution of H_MCOP. Hence, the amount of work it has to perform depends on the CI of H_MCOP, which is similar to a Gaussian bell curve.
In additional evaluations we also found that Delay-Cost-Constrained Routing (DCCR)-3 (which is not shown in Fig. 6.3) has a high CI (between 20% and 45% in most cases). On the other hand, Search Space Reduction Delay-Cost-Constrained Routing (SSR+DCCR) (6.3i and Fig. 6.3j) is interesting. Since SSR+DCCR improves the LARAC solution or one of the intermediate LARAC results, SSR+DCCR has a similar CI fingerprint as LARAC. Interestingly, SSR+DCCR especially improves the solution of LARAC when the delay constraint is tight but still feasible. We observe that SSR+DCCR is, similar to LARACGC, closing the gap of LARAC. Nevertheless, SSR+DCCR appears more powerful than LARACGC for our dense network scenarios since SSR+DCCR runtimes stay reasonably short. As expected, SSR+DCCR-2-5 reduces the runtime compared to SSR+DCCR-4-10; whereas the CI is not strongly affected. Hence, the tuning of the SSR+DCCR parameters requires additional evaluations and is left for future research.

Algorithms Based on the LC and LD Paths The studies on this type of algorithms usually assume that the LD and LC trees can be computed once and then reused for each request, thereby leading to a low request provisioning time. However, in our scenario, we assume that the delay and cost of the edges can change inbetween requests and we therefore have to recompute the tree for each request. Delay-Constrained Unicast Routing (DCUR), Distributed delay Constrained Routing (DCR), and Ishida Aman Kannari (IAK) always follow edges belonging to the LC and LD paths towards other nodes. For our specific queue-link cost and delay settings, which are identical for each physical edge, we have observed that all these algorithms either follow the LC path and then switch to the LD path until the end, or vice versa. This results in the same cost and these four algorithms hence present exactly the same CI behavior. Although this shows that further study is required with different queue-link cost and delay settings, it also highlights that features of algorithms do not always bring some benefit. Indeed, while DCR and IAK only switch once between following the LC and LD paths towards the destination, DCUR can switch any number of times. Since DCUR executes several SP tree runs, it is slower than IAK and DCR which only run one SP search and one SP tree search. However, these

extra computations appear to be useless in our scenario. Because the SP tree search is the most expensive search and because DCR runs an LC SP tree search which enjoys a better guess function than an LD SP tree search, DCR is on average slightly faster than IAK. We show therefore only DCR in Fig. 6.3k. The fingerprint of DCR again shows the difference between SP and SP tree runs. While DCR detects an infeasible problem quite fast with an LD SP search, the following LC SP search for the other cases is much more time consuming, at least in the worst-case.
The runtime ratio of Selection Function Delay Constraind Least Cost (SF_DCLC) (Fig. 6.3l) corresponds to two SP tree searches, except for the infeasible problem where one is enough. As SF_DCLC has more options at each hop, it achieves a better CI fingerprint than DCUR (which is the same as DCR, as noted above) but still with a similar shape. Interestingly, we observe that SF_DCLC and H_MCOP have very similar CI fingerprints, although they are very different in terms of implementation. Examining the output of the algorithms closely, we noticed that, for identical requests, they always returned paths with identical costs. In particular, we observed that SF_DCLC and H_MCOP both prefer one path over the other either based on the cost or on the delay metric depending on whether these paths are feasible or not. Even though SF_DCLC proceeds node by node and H_MCOP within an SP search, both algorithms find typically the same paths (i.e., they nearly always find identical paths and in the rare cases where the paths are different, the paths have identical costs), at least for our specific delay and cost distributions.

Summary In general, it is interesting to note that most algorithms exhibit a higher variability of the CIs when the delay constraint is tight (but still feasible). This can be explained by the fact that there are relatively few possible paths. Hence, if the best one is not chosen, the cost can quickly increase. Interestingly, Chong's algorithm [87] appears to be a good tool to resolve this issue, as has been demonstrated by kH_MCOP, and SSR+DCCR. In additional evaluations (not included in Fig. 6.3), we found that DCCR and kDCBF also show this behavior.

6.1.2.2. Heatmaps: Impact of Network Topology and Scale

In order to observe the behaviors of the algorithms for the different topologies and scalability levels, we collapse the fourth dimension (delay constraint tightness) of our evaluation framework by retaining only the average runtime ratio and CI over all delay constraint levels[2]. This yields the heatmaps shown in

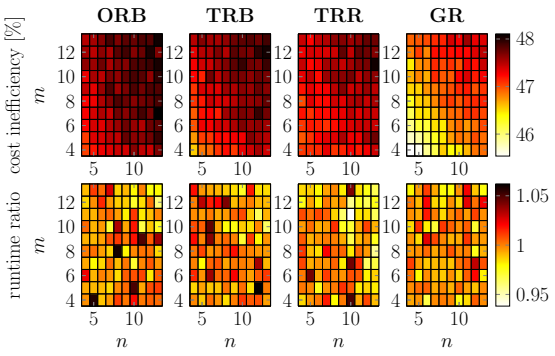
[2]For both the runtime ratio and the CI, only values between the 1% and 99% percentiles were considered for the computation of the average.

Fig. 6.4. Specifically, each cell of each sub-figure in Fig. 6.4 corresponds to a fingerprint plot, whereby the fingerprint plots for the GR topology with $n = m = 10$ have been shown in Fig. 6.3, the other fingerprints are available at [158]. While observing the scalability of the different algorithms with these heatmaps, the reader should pay attention that the scalability of the algorithm is compared to an LD search. That is, if an algorithm presents the same runtime ratio for all the scalability levels of a topology, that does not mean that its runtime is always the same but rather that the considered algorithm has *similar scaling behavior* as an LD search.

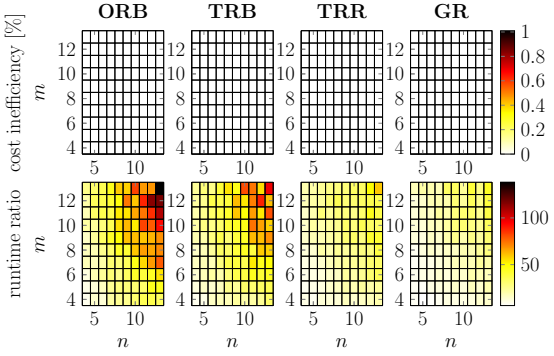
Elementary Algorithms: LDP as Runtime Benchmark The lower four plots for the LDP runtime, see Fig. 6.4a, confirm that our runtime metric has, on average, an inaccuracy of less than 6%. These inaccuracies are mainly due to the inaccuracy of the CPU clock. This indicates that our LDP based runtime metric provides a valid runtime reference benchmark across the three evaluation dimensions of topologies and scale parameters m and n.

Priority Queue Based Algorithms The heatmaps of CBF (Fig. 6.4b) illustrate the limitation of CBR: the CBF runtime grows exponentially with the size of the network, which is consistent with the observations in [129].

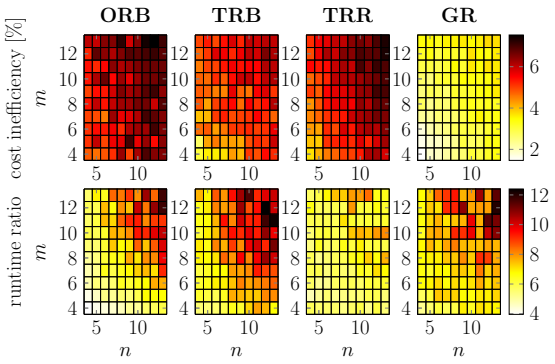
Algorithms Based on BF The DCBF CI (Fig. 6.4c) is only slightly affected by the size of the topology. Interestingly, the DCBF CI is much better, i.e., lower by a factor of two, in the GR topology compared to the other three considered topologies. The intuitive explanation for this DCBF behavior is as follows. DCBF may be sub-optimal if it "relaxes too much". DCBF relaxes a node when (i) the cost of the new path is lower, and (ii) the new path satisfies the projected delay test. Two cases are possible: Case 1: The cost of the new path is lower and its delay is also lower than the current path at a node. The relax operation is hence justified.
Case 2: The cost of the new path is lower and its delay is higher, but still satisfies the projected delay test. Since the new path has a higher delay than the older path, it can happen that DCBF has to take a higher cost path to satisfy the delay constraint. Therefore, although at the current node, the path was cheaper, because of its higher delay, DCBF then has to follow a higher cost path to reach the destination in time. This is the DCBF sub-optimality scenario, where DCBF is not optimal anymore because the final path would have been cheaper by keeping the original path (which was more expensive at one node). This DCBF sub-optimality scenario occurs more frequently when paths have to share nodes, because only one path will be kept at each node. The grid topology has a lot



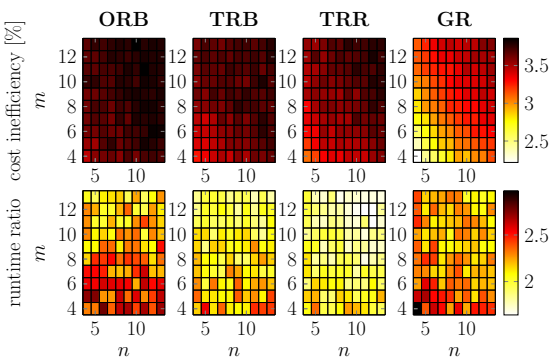
(a) LDP



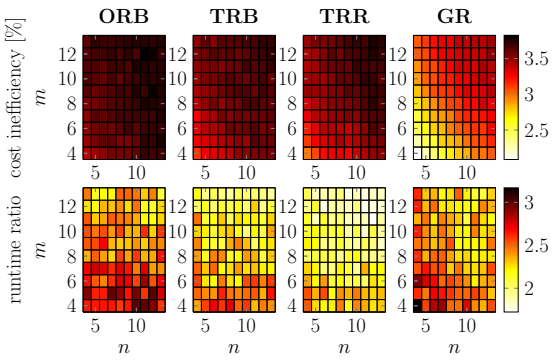
(b) CBF



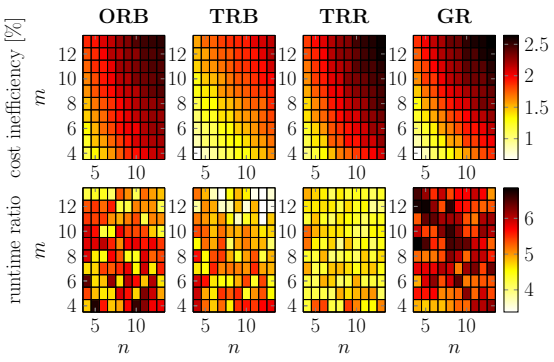
(c) DCBF



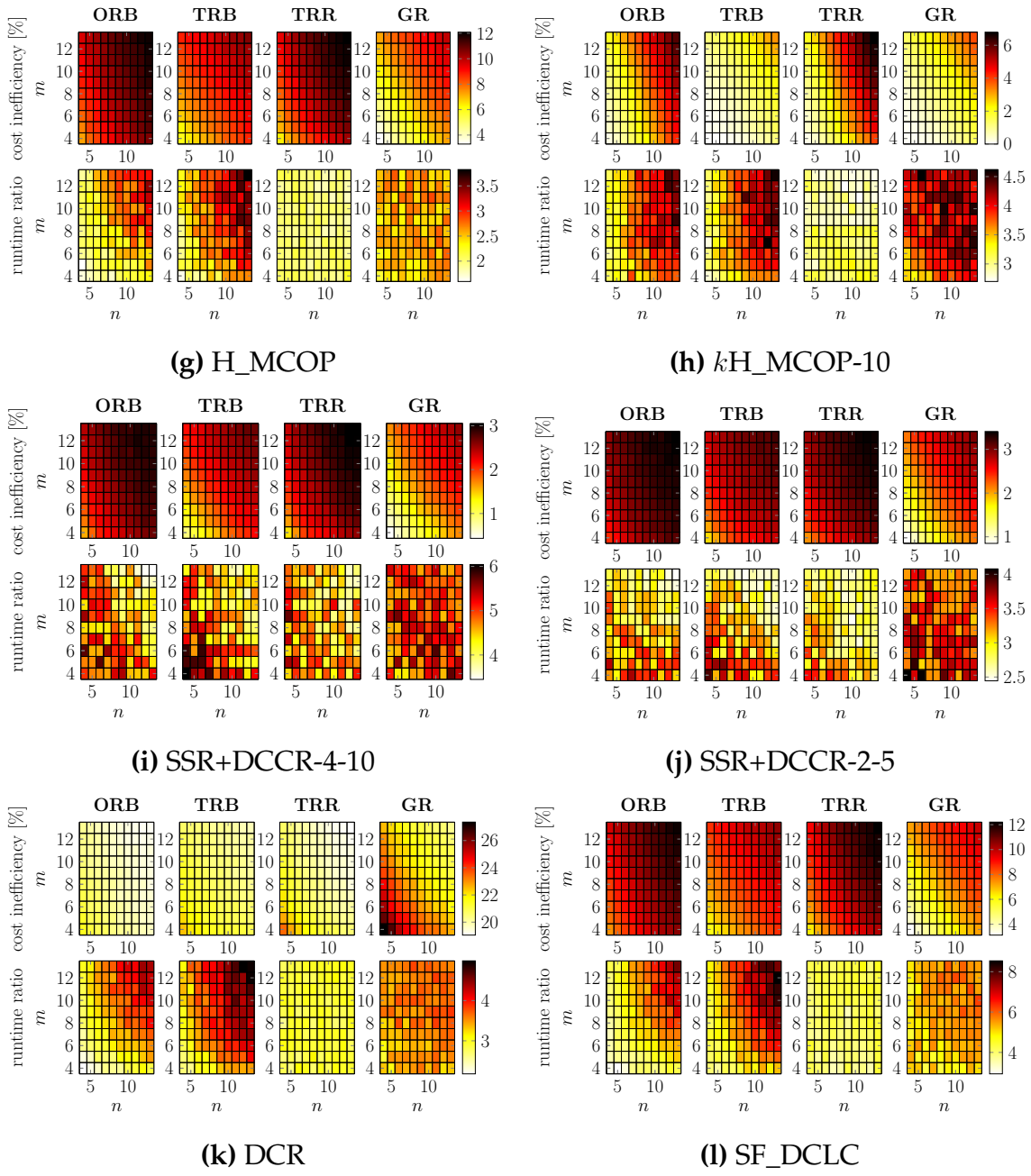
(d) LARAC



(e) LARACGC-25



(f) kLARAC-3



(g) H_MCOP **(h)** kH_MCOP-10

(i) SSR+DCCR-4-10 **(j)** SSR+DCCR-2-5

(k) DCR **(l)** SF_DCLC

Figure 6.4.: Heatmaps showing the behaviors of selected QoS routing algorithms for different topologies and scalability levels. For a given algorithm, the four upper heatmaps show the CI for the four different topologies, and the four lower heatmaps show the runtime ratio. A given heatmap shows the CI or runtime ratio as a function of the scale parameters $n = 4, 5, \ldots, 13$, and $m = 4, 5, \ldots, 13$, i.e., for a total of 100 different scalability levels. Each cell corresponds to the average results of 20,000 requests (with randomly drawn delay constraints from across the seven considered delay constraint levels and corresponding subranges) simulated for this specific n and m combination. To prevent outliers from biasing the results, only values between the 1% and 99% percentiles are considered for computing the average. Unfortunately, because of the high variability between the algorithms, the scales are different for each algorithm.

of diversity (dense graph), thus DCBF sub-optimality arises only rarely. In the other topologies, paths frequently share nodes (because of the branches in the topologies), which leads to frequent DCBF sub-optimality scenarios and therefore to a higher CI.

In additional evaluations that are not included in Fig. 6.4, we observed that dual extended Bellman-Ford (DEB) has a high CI (between 12 % and 35 % on average). The high DEB CI is due to the fact that DEB tries to reduce the CI by checking paths of different hop counts. Nevertheless, in our queue-link topologies, we have many paths of identical length and the CI can be dramatically changed simply by choosing different queues at each hop and hence without changing the path length.

Algorithms Based on Lagrange Relaxation As already observed for the fingerprints, we observe that the runtime increase of LARACGC-25 (Fig. 6.4e) compared to LARAC (Fig. 6.4d) is not worth the slight CI reduction achieved with LARACGC-25. The runtime increase we observe here is not substantial since it only happens for one of the seven delay levels (as observed in the fingerprints, see Section 6.1.2.1). Interestingly, while LARAC and LARACGC-25 behave better, in terms of CI, for the GR topology, kLARAC-3 (Fig. 6.4f) behaves better for the TRB topology. Fig. 6.4d, Fig. 6.4e, and Fig. 6.4f indicate an interesting property of LARAC algorithms: in terms of runtime, they scale better than an LD search, but only in the m scale direction; the n scale dimension affects them as it affects an LD search. In additional evaluations we observed that different MD parameters for LARAC do not affect the LARAC scalability behavior, but only change the absolute values [158].

The H_MCOP runtime (Fig. 6.4g) scales worse than an LD search in both directions (m and n) for the ORB and TRB topologies. On the other hand, the H_MCOP runtime exhibits much better scaling behavior for the two other topologies (TRR and GR). While H_MCOP reaches low CI for small topologies, we observe that the H_MCOP CI grows quickly for larger topologies. Fig. 6.4h shows that using Chong [87] with H_MCOP does not change its scalability. Indeed, while kH_MCOP-10 is then able to reach optimality for small topologies, its CI grows quickly as the topology sizes increase. Nevertheless, this dramatic CI reduction only leads to a slight increase in runtime (by roughly 1 unit). In additional evaluations, MH_MCOP exhibited an identical scaling behavior, though still improving the CI by a factor of around two at the expense of approximately doubling the runtime.

SSR+DCCR (Fig. 6.4i and 6.4j), scales similarly to the underlying LARAC algorithm.

Algorithms Based on the LC and LD Paths DCR (Fig. 6.4k) and SF_DCLC (Fig. 6.4l) present a similar scaling behavior, in terms of runtime, as H_MCOP. Thus, the similar H_MCOP, DCR, and SF_DCLC runtime scaling behaviors appear to indicate the scaling behavior of an SP tree search compared to an SP search. DCR, and hence DCUR, and IAK, exhibit the interesting behavior that their CI improves as the topology scales up. This is unfortunately not true for the runtimes of DCR, DCUR, and IAK. These are the only algorithms showing this behavior. SF_DCLC and H_MCOP show the exact same cost (CI) behavior, hence confirming our observation that they actually always return equal cost paths (see Section 6.1.2.1).

General Impact of Topology We conclude the discussion of the heatmaps in Fig. 6.4 by briefly summarizing the general impact of the type of topology. We observe from Fig. 6.4 that most algorithms have a better CI for the GR topology than the other three topologies, except DCR (and DCUR and IAK) whose CI behavior is opposite to all the others. This observation appears to indicate that switching between LC and LD paths brings improvements for large topologies. Another common observation is that all algorithms have generally shorter runtime for the TRR topology than the TRB topology. The only difference between the TRB and TRR topologies is the set of communicating nodes. Nevertheless, we observe that this small difference has a major impact on the runtime of most algorithms (though similar in that they get faster in TRR).

6.1.2.3. Which Algorithm is Best?

After analyzing the behaviors of all the algorithms, we are in a position to address the question: *which algorithm is the best?* From our observations, the answer is: *it depends*. Indeed, none of the algorithms is better than all others in terms of both runtime and CI for all topologies, scalability levels, and delay levels.

The first "it depends" consideration is in regard to the relative importance of cost and runtime. While kLARAC, kH_MCOP, and all optimal algorithms are good solutions if the cost is the most important criterion, algorithms, such as LDP, FB, or H_MCOP, should be preferred if a very short runtime is critical. LARAC and SSR+DCCR are algorithms that achieve relatively good performance for both the cost and runtime performance metrics.

Secondly, the selection of the best QoS routing algorithm depends on the specific region in the 4D evaluation space where the algorithm is supposed to operate. Indeed, we have seen that for small topologies and/or tight delay constraints, CBF remains a very good candidate. As the topology grows, algorithms with

better scalability are needed. In terms of cost, DCR, DCUR, and IAK are the only algorithms with decreasing CI for large topologies and these algorithms are therefore good choices for very large topologies. In terms of runtime, only the different LARAC and SSR+DCCR variations scale better than an LD search and are therefore also good candidates for large topologies. Therefore, the only way of selecting the best QoS routing algorithm for a given scenario is to consider the evaluation for the specific planned usage scenario.
Nevertheless, we can identify LARAC and SSR+DCCR as being among the best QoS routing algorithms at any point of the 4D evaluation space. That is, for any topology, and topology scale, and delay level, LARAC and SSR+DCCR are among the best performing algorithms. Indeed, on average, for the simulated topologies and network scales, both LARAC and SSR+DCCR keep their runtime ratio lower than four and their CI lower than 4 %. While LARAC and SSR+DCCR scale well in terms of runtime, their CI grows only slightly for large topologies. Moreover, their behavior on the fourth dimension, i.e., for the different delay levels, is quite stable. Last but not least, both LARAC and SSR+DCCR accept several parameters that allow to tailor them to specific usage scenarios.

6.2. Function Split Evaluation

The function split (Section 2.4) simplifies the original routing problem (Section 2.3) by transforming it to a simple CSP problem. The original problem was defined as finding a path for every flow out of the paths leading form source to destination fulfilling the delay and loss constraint:

$$\begin{aligned} P_f \in \mathcal{P}_{(o_f,d_f)} \quad \forall f \in \mathcal{F}. \\ D(P_f) \leq t_f \\ L(P_f) = 0, \end{aligned}$$

Since delay and loss are depending on the load caused by the flows using the links, finding a solution for this problem is not trivial. The function splits enables the solution of a CSP problem, instead:

$$\begin{aligned} \min_{P_f \in \mathcal{P}_{(o,d)}} \text{Cost}(P_f) \\ D(P_f) \leq t_f \end{aligned}$$

The packet loss is handled by the network resource model. The online delay value was replaced by a static worst case bound. This keeps all CSP solutions valid for the lifetime of a flow. This problem simplification might lead to a loss in optimality.

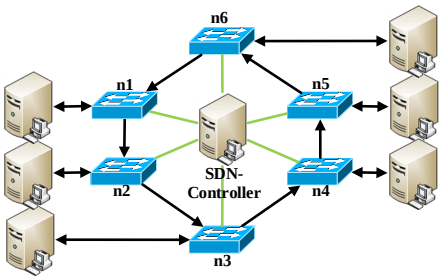


Figure 6.5.: Illustration of unidirectional ring physical link topology for evaluation of the end-to-end real-time QoS framework with function split between resource (transmission bit rate) allocation to priority queue links and routing through queue links. Each link has 1 Gb/s and four priority queues, each with 90,000 Byte buffer (as per NEC PF5240 switch specifications).

In this section the initial evaluation of the Function Split is presented. We compare a MIP based solution [1] of the overall problem with the first implementation of the Function Split [3]. The goal of this is an understanding of the trade-off between runtime and optimality. In addition insights in the three important resources (Data Rate, Buffer Capacity and Time) in a packet switched real-time system can be gathered. The content of this section is mainly based on [3].

6.2.1. Network Set-up and Traffic Mixes

For the simulation-based evaluations in this section, we consider a unidirectional ring, which is a typical elementary industrial network structure, to illustrate the performance characteristics of the proposed function split QoS framework. The ring has six switching nodes and six source/destination nodes, see Figure 6.5. Each switching node output port (link) has four output queues operating with strict priority scheduling (without preemption). Note that the resulting queue link network has for three hops already 64 possible paths. Each flow belongs to one of the four service classes in Table 6.2. Specifically, we consider the ensemble of 968 traffic mixes obtained by letting each of the four traffic service classes contribute 5, 10, 15,. . ., 85 % of the total traffic. Independently of the service class, we draw for each flow a random hop distance according to a short (S) hop distance scenario with 70% of the flows transmitted over one hop, 20% over two hops, and 10% over three hops and a long (L) hop distance scenario with 10% 1 hop, 10% 2 hops, 20% 3 hops, 30% 4 hops, and 30% 5 hops flows.

Table 6.2.: Traffic characteristics and delay limits of the considered industrial traffic service classes; each traffic class has a packet size of 64 Byte.

Service Class	Mean Bit Rate r_c [kByte/s]	Burst Size b_c [Byte]	Delay Limit t_c [ms]
$c = 1$	10	100	5
$c = 2$	10	100	10
$c = 3$	10	100	20
$c = 4$	10	100	50

6.2.2. Routing Cost Functions

We introduce two per-hop cost components and corresponding cost functions. Please note that this cost components are predecessors of the cost functions defined in section 5.2. We use them for the comparison of the function split only. Our "simple" per-hop cost component counts the number of queues q with a priority level less than or equal to the priority level $p(u, v, q)$ of the queue q traversed by the considered flow. With $q = 1$ denoting the lowest priority and $q = |\mathcal{E}^q_{(u,v)}|$ denoting the highest priority, the number of queues with lower (or equal) priority than queue q is simply $c_s(u, v, q) = q$.
Our "buffer-aware" per-hop cost component counts the buffer usage $B^{f_{\text{new}}}_{\max} - B_{\max}$ due to a new flow f_{new} being added to the existing set of flows. This buffer usage is normalized by the allocated buffer space $\mathbf{A}_B[u, v, q]$ and the burstiness $b_{f_{\text{new}}}$ of the new flow:

$$c_b(u, v, q) = \frac{U^{B}_{(u,v,q)^{+f}} - U^{B}_{(u,v,q)}}{A^{B}_{(u,v,q)} b_f}. \tag{6.1}$$

The total cost of an end-to-end path is the sum of the per-hop cost components $c(u, v, q)$ (e.g., $c(u, v, q) = c_s(u, v, q)$ or $c_b(u, v, q)$) along the traversed path, i.e.,

$$C(P_i) = \sum_{(u,v,q)\in P_i} c(u, v, q). \tag{6.2}$$

6.2.3. Evaluation Procedures

6.2.3.1. Online Routing and Admission Control

In order to obtain detailed insights into the performance of our online routing algorithm, including the online admission control for new flows, we consider the online routing initially in isolation from the offline resource allocation algorithm in Section 5.3.1. In particular, we consider 2925 rate allocation patterns $\mathcal{AP}$

obtained by allocating the $R = 1$ Gb/s physical link rate in 5 MByte/s steps to the $|\mathcal{E}^q_{(u,v)}| = 4$ priority queue links ($q = 1, 2, 3, 4$) for each physical link. For each rate allocation pattern, we successively add in flows according to a given considered traffic mix (out of a total of 968 considered traffic mixes). For each new flow, we run the online routing and admission control algorithm. We gradually add in as many flows as admissible subject to the QoS requirements. We thus find the best and the worst resource allocation pattern, i.e., the pattern that leads to the most carried (admitted) flows and the fewest carried flows in the network among the 2925 considered rate allocation patterns.

6.2.3.2. Offline Resource Allocation

For the evaluation of the offline resource allocation from section 5.3 we simulate the continuous operation. of the network for each of the 968 traffic mixes. Specifically, we initialize the network with a uniform (equal) split of the physical transmission bit rates to the queue links. Then we continuously try to add in more flows of the considered traffic mix (using the online routing and admission control) while the resource allocation proceeds once in the background through all six switching nodes.

6.2.3.3. Comparison Benchmark: MIP Solution

We compare with the results generated by the offline MIP solution to the joint problem of routing and resource allocation in [1]. This MIP formulation maximizes the number of flows

$$\max |\mathcal{F}|$$

while flowing the distribution given traffic mixes. And it is constrained by the formal problem specified in Section 2.3:

$$\begin{aligned} P_f \in \mathcal{P}_{(o_f,d_f)} \quad \forall f \in \mathcal{F}. \\ D(P_f) \leq t_f \\ L(P_f) = 0, \end{aligned}$$

This comparison provides a benchmark for the performance of (i) our online routing and admission control algorithm, which is only optimal for the routing of each flow, and (ii) our offline resource allocation algorithm which sequentially considers rate allocations at individual nodes. The MIP solution finds an overall optimum for the joint routing and resource allocation.

Table 6.3.: Comparison of computation times for short hop distance scenario: online routing and admission control for a single flow vs. offline (background) resource allocation (one iteration through all 6 switching nodes) vs. MIP for resource allocation and routing.

	Online Routing and Adm. Control	Offline Resource Allocation	MIP: Res. Alloc. and Routing
25th percentile	2.85 μs	24.9 s	148 s
mean	3.47 μs	27.1 s	333 s
75th percentile	3.95 μs	29.9 s	333 s

6.2.4. Computation Time

In Table 6.3, we compare the computation time of the offline MIP algorithm [1] with the total computation time for the elementary resource allocation algorithm (Section 5.3.1) and the online routing (and admission control) from [3]. Please note that the online routing process presented in [3] is a predecessor of the work presented in this thesis. Nevertheless, the results still provide insights in the optimality of the function split. We consider the short hop distance scenario with up to three hops, for which the MIP is computationally feasible (the MIP becomes computationally prohibitive for the long hop distance scenario). All times were measured for computations on the same physical computing hardware in order to allow for meaningful computation time comparisons of the different approaches. The computation times in Table 6.3 are based on the 968 different traffic mixes, i.e., the variations of the computing times represent the variations due to the different traffic mixes. We observe that the mean total computation time (for a given arbitrary traffic mix) for the online routing and admission decision for a new flow seeking admission to the network is less than 5 μs. While the delay for making routing and admission decisions in the controller does not impact the packet delay of established flows on the forwarding (data) plane, the very fast routing and admission control computation in combination with real-time QoS control channel flows that could be established between switches and the controller could ensure delay bounds for flow establishment.
One iteration of the resource allocation algorithm through all six switching nodes, which is executed in the background, i.e., does not impede the online routing and admission control, takes on the order of 30 s. In contrast, the MIP takes on the order of a few hundred seconds.

6.2.5. Utilization of Links, Buffers, and Delay Limits

We present cumulative distribution function (CDF) plots (across the 968 traffic mixes) in Figures 6.6, 6.7, and 6.8 for:

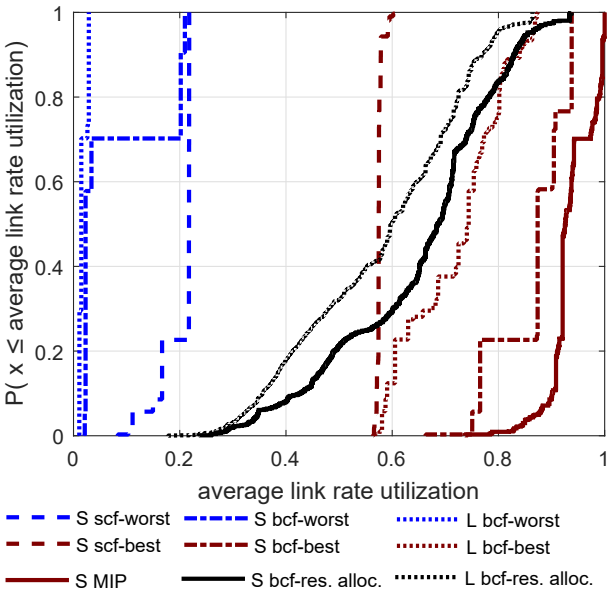


Figure 6.6.: Comparison of the cumulative distribution function (CDF, across the 968 traffic mixes) of the average link (rate) utilization for the online routing evaluation for fixed rate allocations from Section 6.2.3.1 (showing curves for the worst and the best rate allocation), the evaluation of online routing in conjunction with the background resource allocation algorithm from Section 6.2.3.2 (bcf-res. alloc.), and the MIP benchmark evaluation from Section 6.2.3.3 The short (S, up to 3 hops) and long (L, up to 5 hops) hop distances are considered. The simple cost function (scf) and the buffer-aware cost function (bcf) are considered for the evaluation from Section 6.2.3.1.

- The **average link (rate) utilization** defined as the mean utilization of all links in the network. (Figure 6.6)
- The **average buffer utilization** defined as the mean utilization of all buffers in the network. (Figure 6.7)
- The **average delay deviation** defined as the average of the deviation of the end-to-end worst-case path delay $D(P_f)$ form the delay limit t_f in percent, i.e., the average of $(t_f - D(P_f))/t_f$, over all flows f carried in the network. (Figure 6.8)

We focus initially on the online routing and MIP results for the link rate utilization in Fig. 6.6 and the buffer utilization in Fig. 6.7 for the short hop distance scenario. The results for the link utilization in Figure 6.6 indicate higher bandwidth utilizations with the buffer-aware routing cost function (bcf) than the simple cost function (scf). For the best resource (rate) allocation, the bcf routing achieves up to 93 % link utilization compared to up to close to 60 % link utilization with scf routing (and compared to up to 100 % with the MIP).

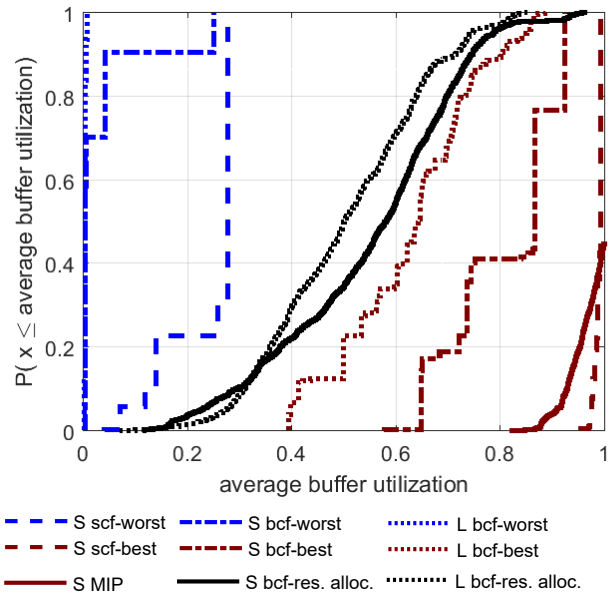


Figure 6.7.: Comparison of the CDF or the average buffer utilization of the overall network for the three evaluation procedures from Sections 6.2.3.1– 6.2.3.3 for the simple (scf) and buffer aware (bcf) routing cost functions for short (S) and long (L) hop distance scenarios.

At the same time, we observe from Figure 6.7 that the scf routing leads to a higher average buffer utilization than bcf routing. For the best resource allocation (that maximizes the number of flows), the scf routing almost fully utilizes the buffers; whereas, bcf routing utilizes approximately 65–92 % of the buffers. The simple cost function counts the number of queues with equal or lower priority and thus strives to route flows through low priority queues. However, due to the principles of deterministic network calculus, the low priority queues have only relatively low guaranteed available transmission bit rates $R_{(u,v,q)}$, see Eq. (4.15), since the transmission bit rates allocated to higher priority queues are not considered to be available for the low priority queue. The low level of guaranteed available transmission bit rate S leads to a high worst-case maximum buffer occupancy (utilization), see Eq. (4.8), for the low priority queues. In contrast, the buffer aware cost function considers directly the maximum buffer occupancy, see Eq. (6.1), and accordingly routes flows through a mix of low and high priority queues so as to "save" buffer space. Comparing Figs. 6.6 and 6.7 we observe that for the considered networking scenario, the buffer space becomes the bottleneck resource for the scf routing: The scf routing utilizes nearly all available buffers for average link utilizations close to 60 %. In contrast, bcf routing achieves up to 93 % link utilization while requiring less than 92 % of the available buffer for the worst-case requirements.
We observe that for a given routing cost function, there is a wide gap between the utilization achieved with the best and worst resource (rate) allocation patterns

$A^R_{(u,v,q)\in\mathcal{A}_{(u,v)}}$. We plot the utilization results for both the best and worst rate allocation to illustrate the impact of the rate allocation on the performance of the introduced function split QoS framework. Moreover, we observe from Fig. 6.6 that the elementary resource allocation algorithm from Section 5.3.1 (bcf-res. alloc. curve) also covers a fairly wide range of utilization levels. The elementary algorithm that considers rate allocations at individual nodes gives poor rate allocations for some traffic mixes, as indicated by the wide gap to the bcf-best curve at the bottom of Fig. 6.6. However, for over 60 % of the traffic mixes the elementary resource allocation is within 25 % of the maximal supported number of flows of the bcf-best scenario. The refinement of the resource allocation within the function split QoS framework, e.g., by jointly considering rate allocations to several switching nodes is an important direction for future research.

We observe from Figs. 6.6 and 6.7 that the long (L) hop distance scenario gives generally lower link and buffer utilizations than the short (S) hop distance scenario. The bcf routing with offline resource allocation (bcf-res. alloc. curve), for instance, has 5–10 % lower link and buffer utilizations for the long hop distance than for short hop distance. The bcf-best approach achieves bandwidth utilizations up to around 84 % for the long hop distance scenario compared to up to 93 % for the short hop distance scenario. This is mainly because the long hop distance flows require rate and buffer allocations at more successive switch queue links and are therefore more difficult to accommodate, leading to fewer admitted flows and hence lower utilization levels.

Figure 6.8 indicates that the buffer-aware cost function (for the best rate allocation) has a slightly higher average delay deviation between the delay limits and the worst-case path delays than the simple cost function. This means that the buffer-aware cost function gives on average a shorter path delay $D(P_f)$, i.e., utilized less of the available delay limit t_f than the simple cost function. However, even the MIP has an average deviation of over 60 %, i.e., the path delays $D(P_f)$ are less than 40 % of the delay limits t_f for all traffic mixes. The delay deviation curves are fairly close together, except for the simple cost function (with the worst rate allocation) which utilizes on average only about 10 % of the delay limits for roughly 70 % of the traffic mixes. Overall, these delay deviation results indicate that the delay limits are on average only partly utilized, i.e., the packet traffic arrives typically well before the deadlines. We also observe from Fig. 6.8 that the long hop distance scenario utilizes 10–20 % more of the delay budget than the short hop distance scenario.

Overall, the results indicate that the introduced split function QoS framework makes very quick run-time admission decisions while providing deterministic worst-case QoS guarantees. In conjunction with suitable offline (background) re-

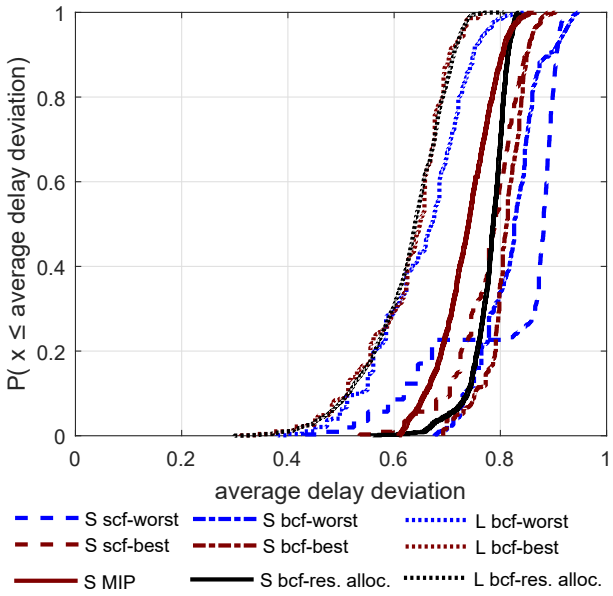


Figure 6.8.: Comparison of the average delay deviation (difference between delay limit t_f and worst-case path delay $D(P_f)$) of the overall network for the three evaluation procedures from Sections 6.2.3.1– 6.2.3.3 for the simple and buffer aware routing cost functions for short (S) and long (L) hop distance scenarios.

source allocation, the connection carrying capacity and bandwidth utilization of our split function QoS network comes for short hop distances in the considered ring topology example network to within 7 % of the performance of a monolithic MIP that jointly optimizes routing and resource allocation.

6.2.6. Conclusion

Our evaluations for two routing cost functions indicate that the proposed function split approach makes accurate routing and admission control decisions within a few microseconds, compared to hundreds of seconds required by a monolithic mixed integer program (MIP) approach that jointly optimizes routing and resource allocation. At the same time, our approach achieves up to approximately 93 % average link utilization in an example industrial communication scenario compared to close to 100 % utilization by the MIP approach.

6.3. Industrial QoS Framework Evaluation

In section 6.1 an evaluation of CSP routing algorithms is presented. There potentially high performance CSP routing algorithms were identified. The impact of the function split is evaluated and discussed in Section 6.2. However, this evalu-

ations only cover a single performance aspect of the industrial QoS framework. All solutions discussed in this thesis contribute differently to the performance of the industrial QoS Framework. There might exist cross dependencies between different implementations. For example some cost functions need an optimal routing algorithm to provide a good performance. To discover these dependencies an evaluation which explores the performance of all possible combinations of solutions in different communication scenarios is needed.
This evaluation is presented in this section. First the scenarios and the solution space are described in section 6.3.1. Since the result of the Monte Carlo method based evaluation is a huge dataset, a stepwise analysis is needed. Section 6.3.3 shows the results of the raw datasets. These results represent the main impacts on the overall performance. To identify the best specific real-time communication system, a more detailed analysis is needed. An analysis based on the results of section 6.3.3 is done in section 6.3.4. Therefore, the low performance configuration parameter identified in section 6.3.3 are not longer considered as part of the solution space. The best system setting is presented in section 6.3.5.

6.3.1. Scenario

Any heuristic algorithm is a compromise between runtime and the quality of its result. Since the service the QoS framework provides is calculation, registration, optimization and deregistration of a end-to-end real-time connection, the evaluation metric has to cover all these lifecycle steps. Based on this we introduce the flowing trade-off metrics:

Maximum Traffic Intensity is the Erlang value the overall system can serve with out rejecting any connection request during the entire simulation.

Lifecycle Runtime is the calculation time of a particular flow spent in the different lifecycle stages.

These metrics are depending on the following configuration dimensions:

Resource Model is defined in Section 4. It provides three independent configurable functionalities:

Access Control is the core functionality of the Network Resource Model. There are two opportunities to implement how the supervision of the QoS bounds is realized. In this evaluation Multi-Hop Model (MHM) (Section 4.5) and Threshold-Based Model (TBM) (Section 4.6) are used.

Input Link Shaping (ILS) (Section 4.8) provides a better resource usage at the cost of a more complex model. It could be used by the framework.

Burst Increase (Section 4.7) models the fact that the burst b_f of the flow increases if the flow is delayed at a queue. There are four opportunities for the system. The first option is to ignore the burst increase. So there is **NO** burst increase. The second option is to transform the token bucket parameter of the flow so that they include the worst case burst increase (**WCB**). The third option is an improved version of WCB. Instead of using the WCB token bucket also in the reservation phase, the real delay value could be used. So the worst case burst is used during routing phase and the real delay during the reservation phase (**WCB-RR**). The last opportunity is to use the real delay value also during the routing phase (**REAL**).

Routing Algorithm are described in section 5.1. From the set of available algorithms CBF is chosen because it is cost optimal, LARAC is chosen because it is the most promising heuristic in terms of speed and cost optimality and LDP is chosen because it is the fastest heuristic.

Cost Function are important for the global optimality of the system. All seven cost functions defined in section 5.2 are possible options.

Resource Allocation Algorithm provides an offline optimization of the system. For this evaluation we could use the Tunable Resource Allocation Algorithm (TRAA) or not. If it is used, the parameters have to be chosen.

Topology is selected a TRB. For the scalability factors m and n values from 4 to 10 could be chosen.

Traffic Mix The Traffic is randomly generated. The burstiness b_f is a uniform distributed random value between 100 and 1000 byte. The flow data rate r_f is a uniform distributed random value between 10000 and 100000 byte/s. The delay constraint t_f is a uniform distributed random value between 5 and 100 ms.

During the discussion of the results we will use a state space diagram (see Figure 6.9) to provide an overview of the data set which is currently part of the evaluation. It contains the routing algorithms, access control, Input Link Shaping (ILS), burst increase, the resource allocation algorithm parameter and the used topology. Please note that there is a special case "TRAA = NO" which indicates that there is no resource allocation algorithm running. These diagrams show in white if a configuration option is not part of the particular evaluation. In this example only the TRB topology is considered for evaluation. Blue color indicates the configuration options of interest. In this example the evaluation targets the CSP routing algorithms. Green color indicates which configuration

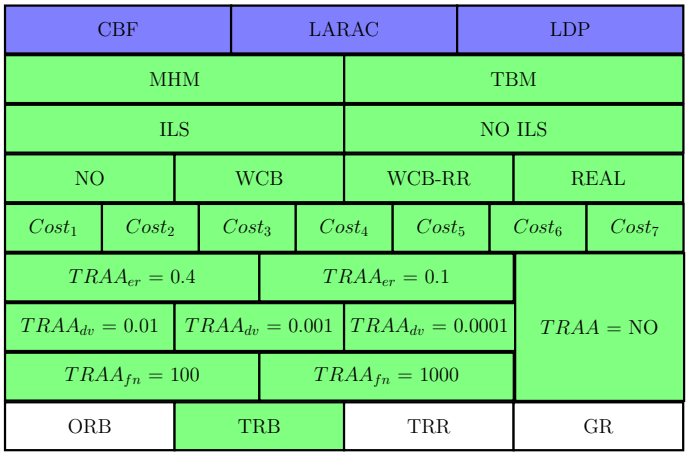


Figure 6.9.: Example state space diagram

Table 6.4.: Number of system setting combinations

	Combinations
Routing algorithm	3
Access Control	2
ILS	2
Burst Increase	4
Cost Function	7
Resource Allocation Algorithm	2x3x2 + 1 = 13
Topology	1
Topology Size	7 x 7 = 49
Traffic Mix	1
Total possible Settings	214032

options are part of the data sets used for the evaluation. In this example there is only the already mentioned filtering of the topologies.
There are two main challenges for evaluating the performance (Maximum System Traffic Intensity; System lifecycle Runtime) by exploring all the configuration dimensions. First, the efficient computation of the Maximum System Traffic Intensity is a not trivial task. Second, the total amount of possible configurations is 214032 (see Table 6.4). To explore all these combinations statistically significant each specific setting needs at least 5-10 simulations. This leads to a total simulation amount of more than one million simulations.

6.3.2. System Evaluation Framework

To find the best configuration for real scenarios two challenges have to be solved. First, there is a need for defining two metrics which reflect the industrial

QoS framework runtime and the efficiency in resource usage (Section 6.3.2.1). The second challenge is to deal with the large solution space. To perform a brute force search more than a million simulations have to be performed (see Section 6.3.1). Since the computational resources are limited, the evaluation have to be performed on a smaller data set. This is discussed in section 6.3.2.2.

6.3.2.1. Metric Computation

To perform a trade-off analysis of the industrial QoS framework two complementary metrics are defined. We first introduce the Maximum Traffic Intensity as a measure of the efficiency in resource usage. The second metric the Lifecycle Runtime reflects the computational complexity of a specific framework configuration.

Maximum Traffic Intensity computation As already discussed in Section 5 the industrial QoS framework could be seen as a traditional telecommunication system. Flow requests are generated with an Poisson distributed arrival rate $\lambda_{\mathcal{F}}$. These flows follow the traffic mix defined in the scenario. In addition there is also a Poisson distributed holding time. With these two values the traffic intensity is:

$$y = \lambda_{\mathcal{F}} h_{\mathcal{F}} \tag{6.3}$$

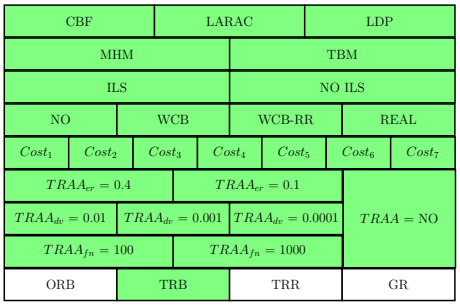
To calculate the maximum traffic intensity we use a static average flow f holding time of 100 seconds. In a time based simulation this arrival termination process is evaluated. First the simulation runs until a steady state (flow creation and termination probability is the same) is reached. This is necessary because the reject probability is only constant in the steady state. Then, the simulation runs for 5000 flow embeddings. During the whole simulation, we check if a flow is rejected by the system. A binary search is used to find the flow arrival rate $\lambda_{\mathcal{F}}$ which leads to no flow rejection during the simulation time. We configure an accuracy gap of a maximum error of 1%. After the binary search has stopped, the best feasible flow arrival rate $\lambda_{\mathcal{F}}$ is used to compute the Maximum Traffic Intensity.

Lifecycle Runtime computation To compute the Lifecycle Runtime all flow related computations are measured separately during the steady state simulation of the 5000 flows. The routing time, the flow calculation, registration and deregistration of each flow is measured separately. In addition the total runtime of the TRAA is measured too. The Lifecycle Runtime is the sum of the average value of calculation, registration and deregistration and the TRAA divided by the number of flows (5000).

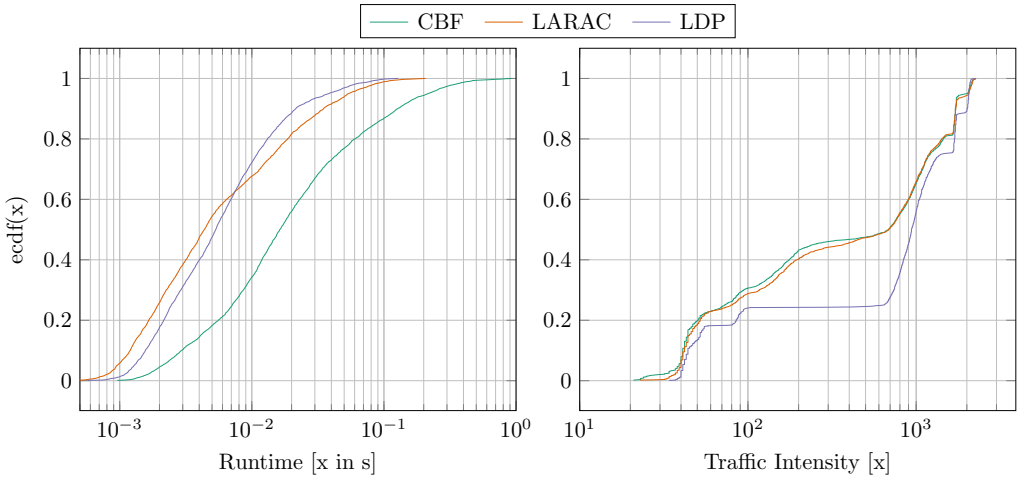
6.3.2.2. Search Space limitation

Since the search space is too big to produce statistically significant results an exhaustive search is not an option. Instead, a Monte-Carlo-method based evaluation was done, to compute a small subset of the solution space randomly. In total 22500 simulations were computed. The evaluation is done on basis of this simulations for each dimension separately. To compare the options of a dimension the Empirical Cumulative Density Functions (ECDF) for each option and metric have to be computed. The result will be visualized in ECDF plots for Maximum Traffic Intensity and Lifecycle runtime. By doing this it is possible to identify the high performance options for each dimension with random settings in all other dimensions.

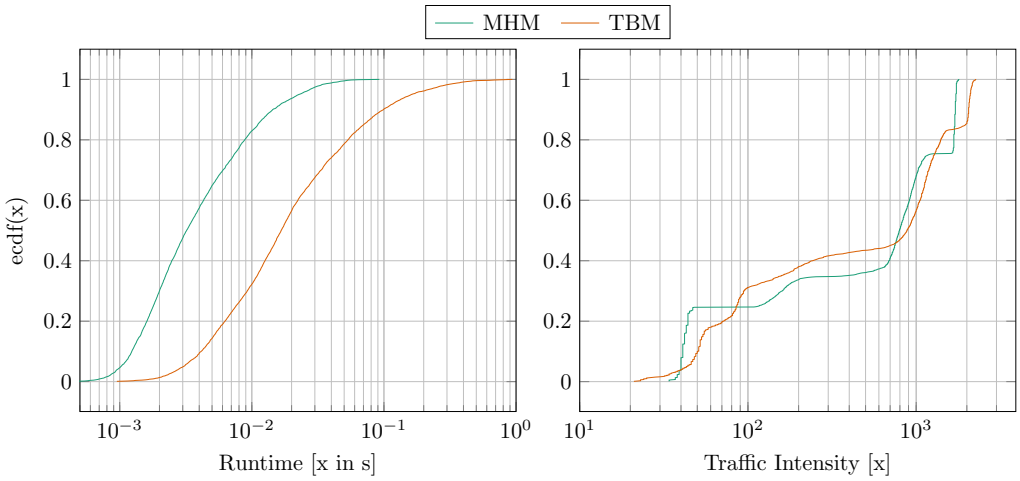
6.3.3. High Level Sensitivity Analyses



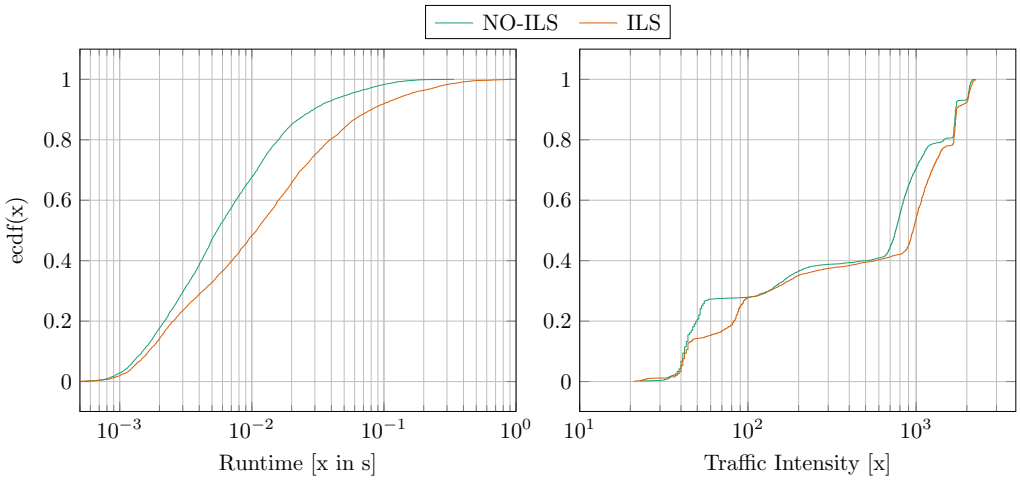
As a first analysis step, the identification of dimensions which mainly influence the performance, have to be performed. Therefore, we analyze the twin plots of runtime and traffic intensity for each dimension. We only consider the dimensions which show a clear separation in the performance graph on the overall configuration space. Please note that the high level sensitivity analysis is only performed on a single topology. Since the topology itself has a main influence on the framework performance it is necessary to focus only one topology. Otherwise, the identification of the framework configuration options influence becomes harder because the strong impact of the topologies hides it. The TRB is selected as a compromise between the different topologies. While ORB is simple, TRR and GR are too complex. Only four dimensions provide a clear separation in there performance graphs so we will discuss them in this section. The remainder of this section is structured as follows. First, we discuss the impact of the routing algorithm dimension on the industrial QoS framework. Second, we continue this discussion with the access control dimension. Third, we elaborate the impact of the ILS dimension. Finally, we discuss the impact of the burst increase dimension.



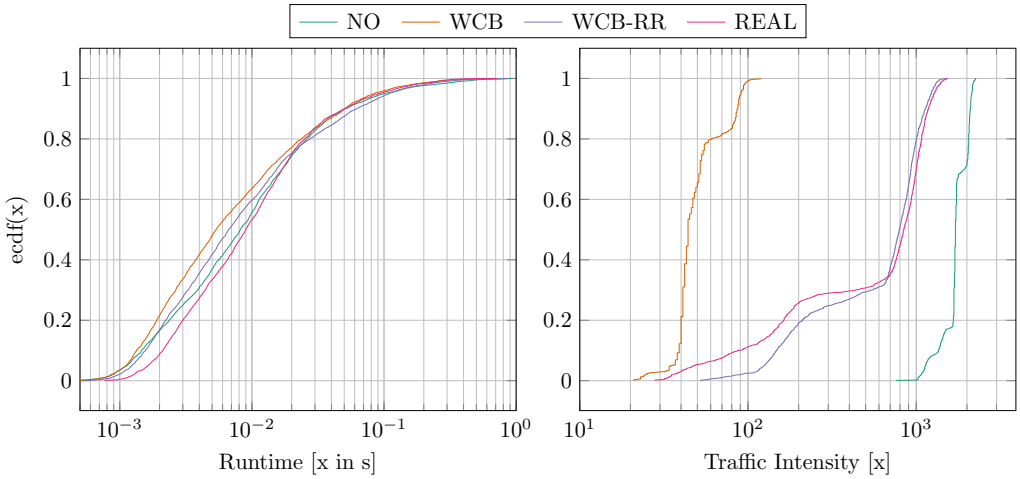
(a) Routing algorithm dimension



(b) Access Control dimension

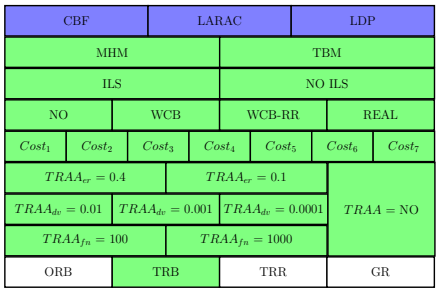


(c) Input Link Shaping dimension



(d) Burst Increase dimension

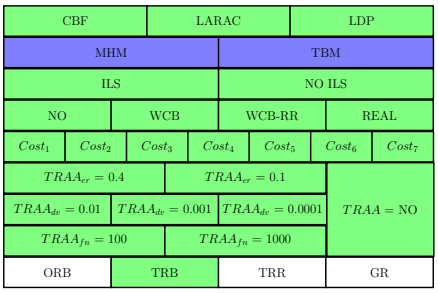
Figure 6.10.: High Level Sensitivity Analyses



Impact: Routing algorithm (Fig. 6.10a) The behavior of the routing algorithms is mainly aligned with the results of the routing algorithm evaluation presented in section 6.1 (see. Figure 6.4). First, we discuss the runtime behavior of the algorithms. The CBF algorithm is the slowest one. Surprisingly LARAC is only 62% of the cases slower than LDP. This effect is plausible because LARAC and LDP use both A* to compute shortest paths. However, the A* performance is highly depending on the quality of the guess function. Since the value ranges of the cost functions (1-10) are much smaller than the ranges of the delay value ($\mu s - ms$), the guess values of the cost function is mor accurate than the guess of the delay. This results in a higher runtime for the least delay run.

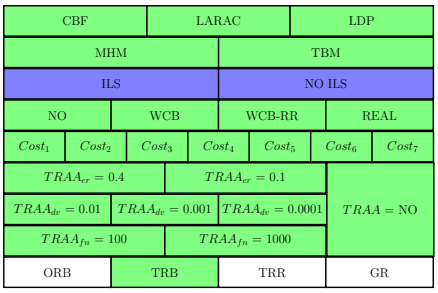
Surprisingly the LDP algorithm performs best in terms of traffic intensity. This is a clear indicator that the majority of the cost functions (Cost_1, Cost_2, Cost_3, Cost_4, Cost_5, Cost_6 and Cost_7) perform worse than just the least delay path. To find the best routing algorithm, it is necessary to repeat the evaluation with the subset of high performance cost functions. The analysis of the cost function and the follow up analysis of the routing algorithms will be discussed within the next section.

Impact: Access Control strategies (Fig. 6.10b)



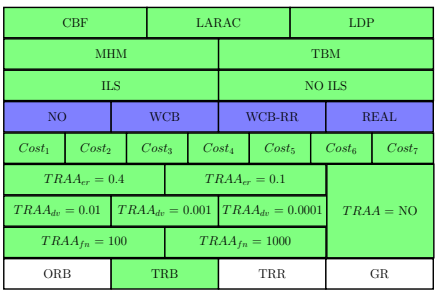
The analysis of the Access Control strategies runtime clearly proves their expected runtime behavior (Section 4.6.2). The MHM provides, due to its simplicity, the best performance in terms of runtime. In worst case the TBM is ten times slower than the MHM. The clear separation of the access control implementations indicates that the Access control has a main influence on the runtime. The results do not show a clear separation of the MHM and the TBM in terms of traffic intensity. The TBM is not capable to outperform the MHM in all of the cases. The bad performance of TBM proves the impact of the blocking problem mentioned in section 4.6.2. To conclude the result the access control strategy clearly influences the runtime behavior of the industrial QoS framework. The influence on the maximum traffic intensity could not be proven with Figure 6.10b. This indicates that other dimensions (like fore example the cost functions) have an higher influence on the maximum traffic intensity. Finding this cross dependency is subject of section 6.3.4.

Impact: Input Link Shaping (Fig. 6.10c)



The ILS behaves as expected in Section 4.8.5 and Section 4.8.6. The runtime of the overall framework is higher or equal if ILS is used. But it also clearly increases the maximum traffic intensity. We can show this behavior also in Figure 6.10c. So depending on the needs, ILS should be used if traffic intensity is the critical factor or not if a low runtime is needed.

Impact: Burst Increase strategies (Fig. 6.10d)

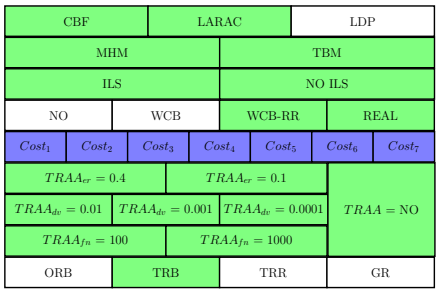


The runtime of the WCB, WCB-RR and REAL burst increase implementation increases in this order which corresponds to the increasing implementation complexity. This behavior is valid for 80% of the cases. The remaining 20% show nearly the same runtime behavior for all burst increase strategies. Interestingly the runtime of the NO mode is higher than the WCB even if they should be the same during the path computation phase. The increase comes from the higher effort during the optimization phase because of the high amount of flows handled by the framework. However, the

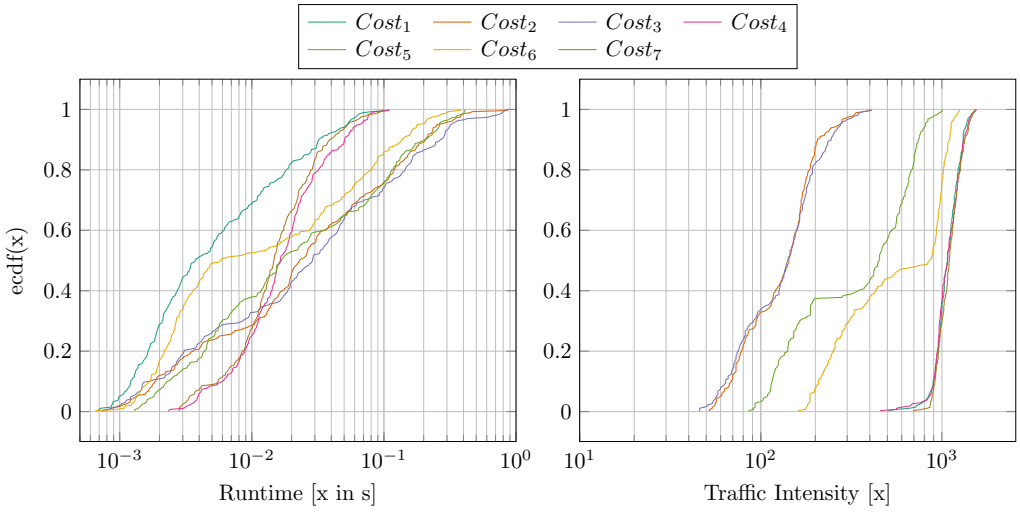
different burst increases do not have a big influence on the runtime compared to other dimensions.
The traffic intensity performance is highly diverse. The WCB method provides the worst traffic intensity due to its conservativeness. In contrast the NO mode provides the best traffic intensity because of not considering the burst increase at all. Because of the NO mode it is possible to serve 50 times more flows than with WCB mode. WCB-RR and REAL mode are quite equal in their performance. They reach in best case half the performance of the NO mode. However, even if the NO mode provides the best performance we do not consider it for a possible solution. It is only valid to ignore the burst increase under the condition that the packet inter arrival time is bigger than the end-to-end deadline. This condition could limit the application too much. The WCB mode can also be excluded due to its poor performance in terms of traffic intensity.

6.3.4. Solution space reduction

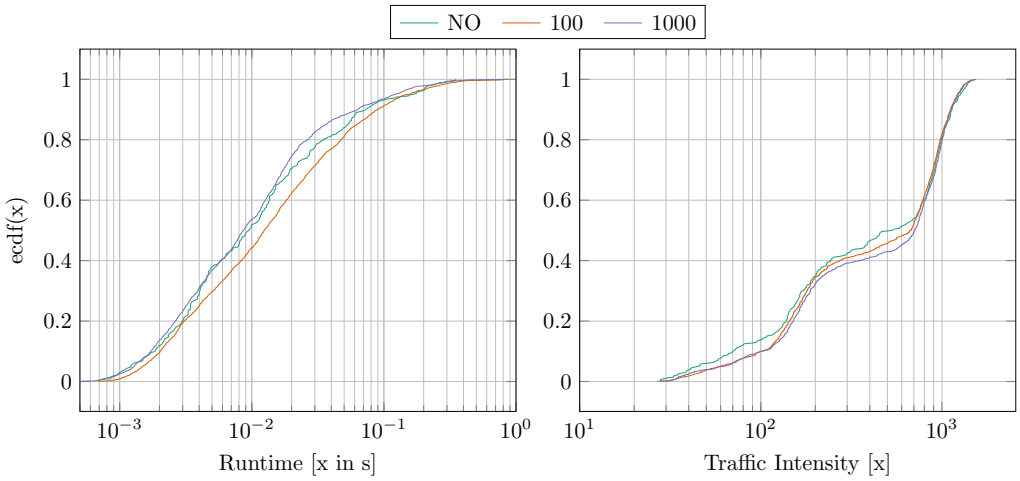
In this section an analysis of the other dimensions is provided. These dimensions have too many cross dependences to the main dimensions identified in section 6.3.3. It is necessary to evaluate them by reducing the influence of the main dimensions. Therefore, we perform the following analysis on a reduced dataset. The cost functions and the Resource Allocation Algorithm were evaluated without considering the LDP routing algorithm. In addition, the burst increase options NO and WCB are not considered because of their strong impact on the traffic intensity and their limited applicability to the industrial use case. Indeed, the NO burst increase is only valid under specific circumstances (see Section 4.7) and the WCB mode provides a best case maximum traffic intensity of hundreds of flows which is not sufficient.



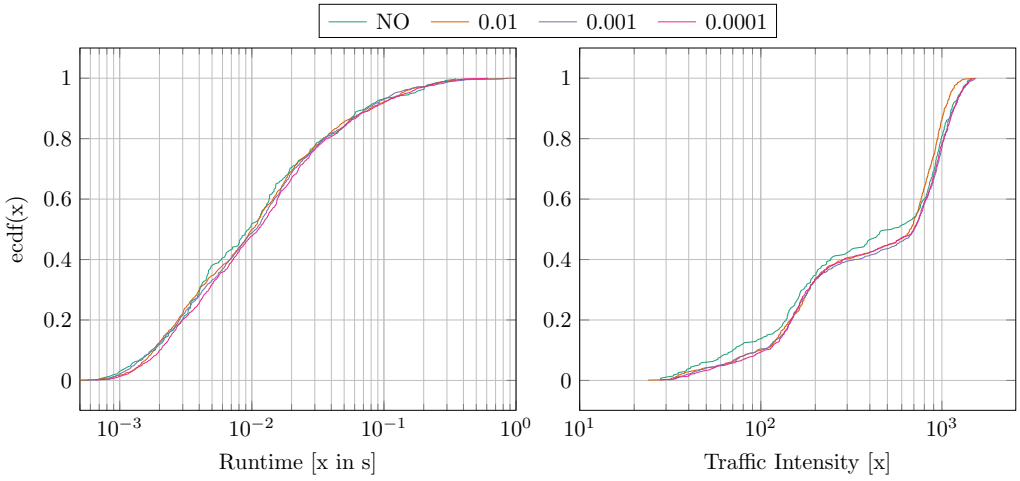
Impact Cost function (Fig. 6.11a) To identify the high performance cost function, the LDP algorithm should not be considered during this analysis. The LDP algorithm ignores the cost function which leads to a not negligible disturbance of the results. In addition, we only consider the cases where WCB-RR and REAL burst increase mode was used. Cost function Cost_1, Cost_4 and Cost_7 show the best performance in terms of traffic intensity on the filtered dataset (see Figure 6.11a). Cost_1 performs a approximation of a LDP search. Cost_4 prices the queue link (u, v, q) by the number of flows it is able to accept in addition. The queue link gets more expensive if the number of flows decreases. Cost_7 is a



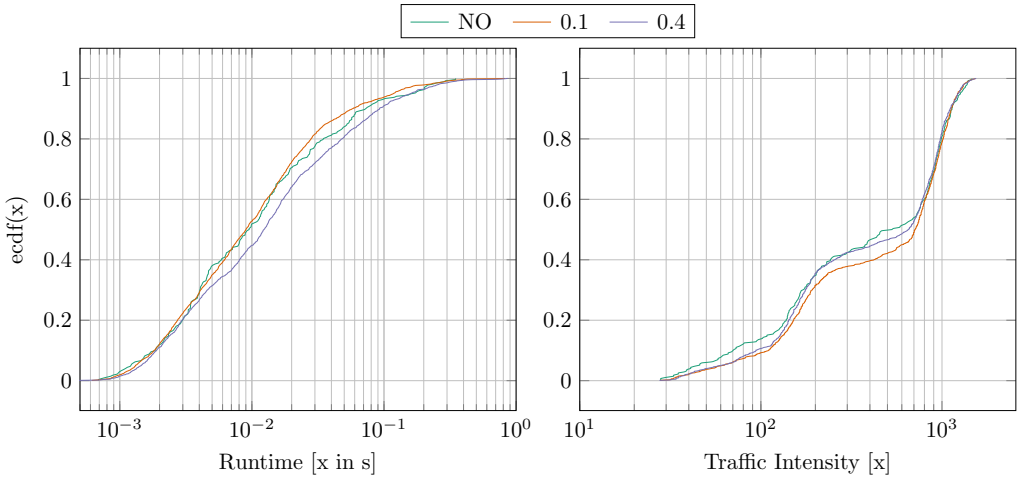
(a) Comparison of the impact of cost function



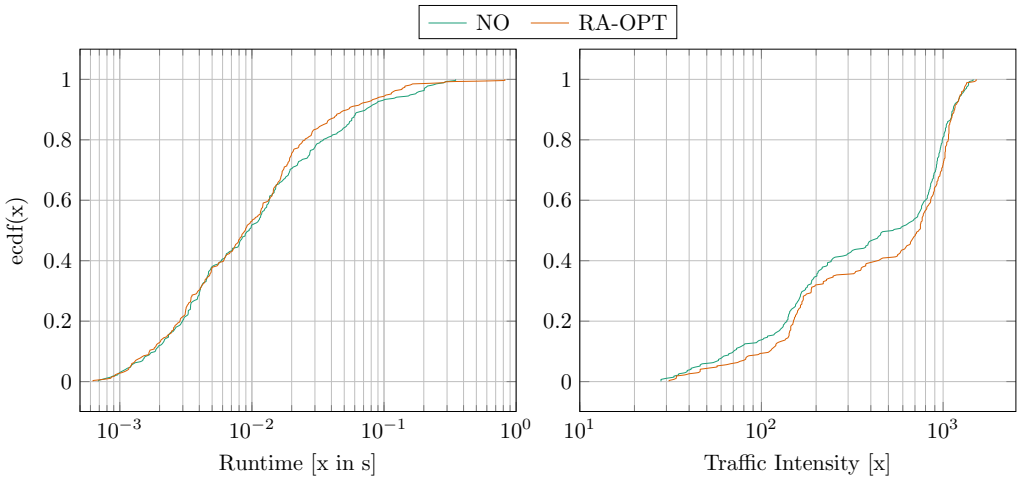
(b) Comparison of the impact of TRAA parameter $TRAA_{nf}$



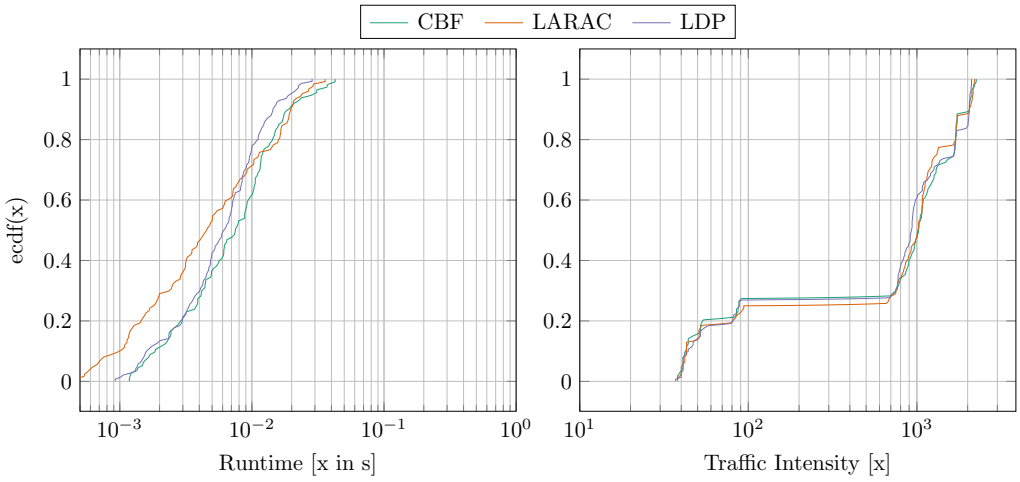
(c) Comparison of the impact of TRAA parameter $TRAA_{dv}$



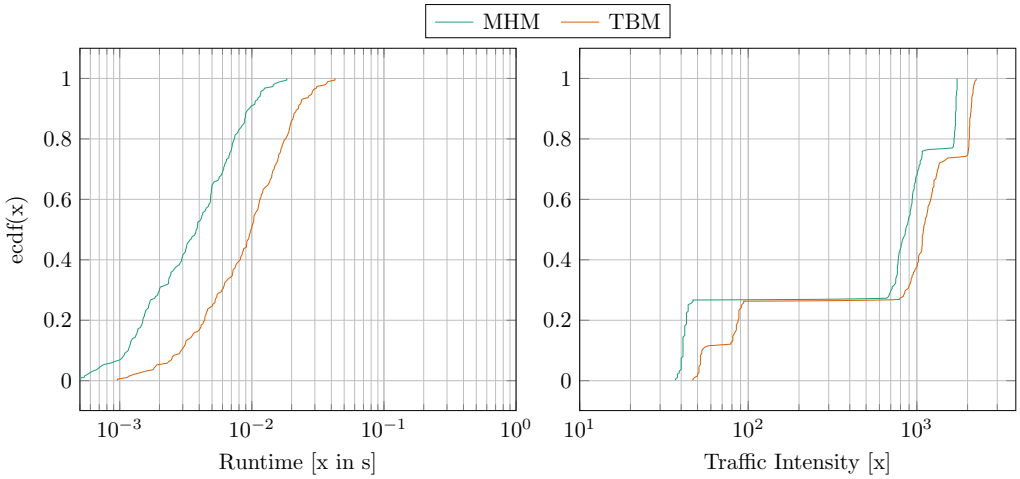
(d) Comparison of the impact of TRAA parameter $TRAA_{er}$



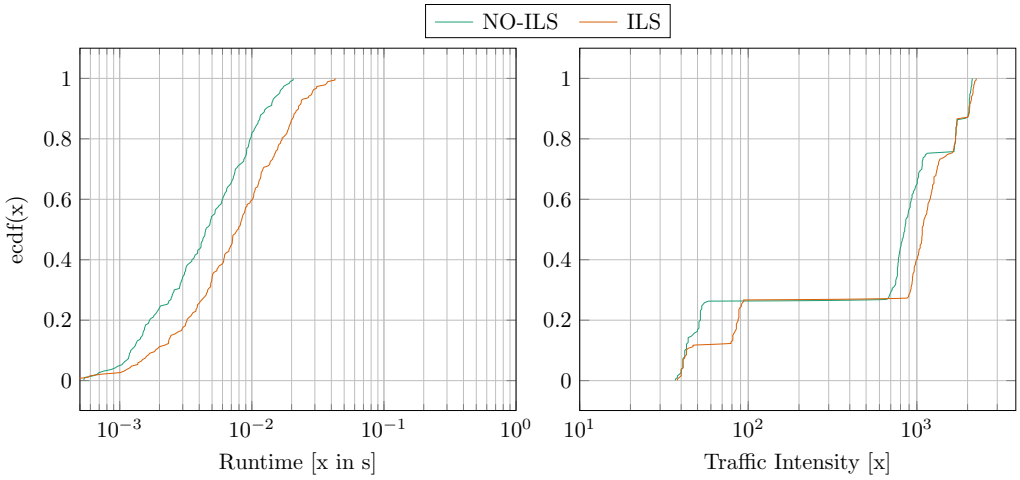
(e) Comparison of the impact of TRAA parameter set $TRAA_{nf}$ = 1000; $TRAA_{dv}$ = 0.001; $TRAA_{er}$ = 0.1



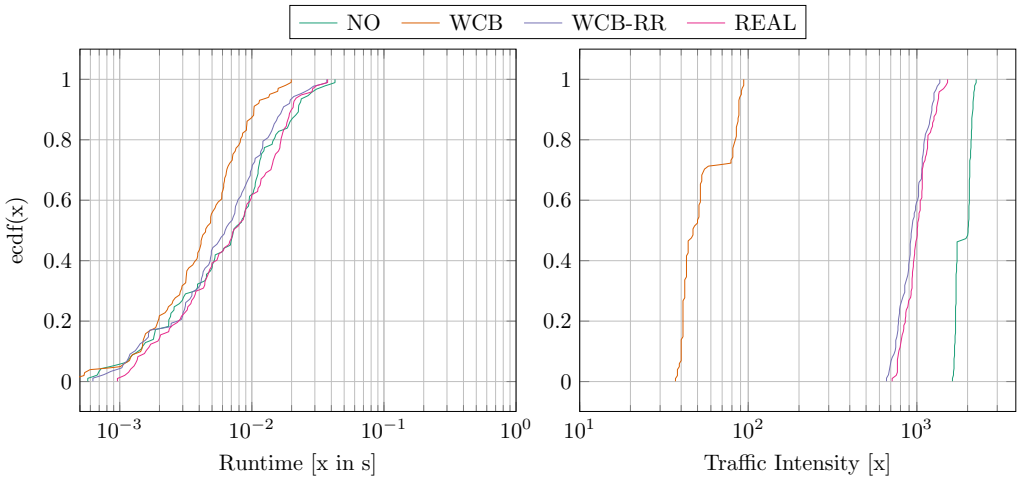
(f) Comparison of the Routing algorithm dimension



(g) Comparison of the Access Control dimension



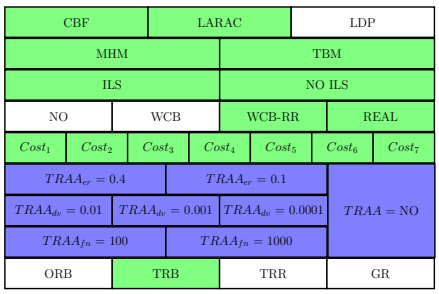
(h) Comparison of the Input Link Shaping dimension



(i) Comparison of the Burst Increase dimension

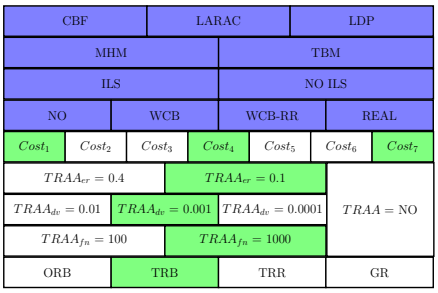
Figure 6.11.: Solution space reduction

mixture of Cost_4 and Cost_1.
Cost_1 provide the best runtime behavior of the overall data set because of its simplicity. The dynamic cost functions Cost_4 and Cost_7 show a similar worst case runtime behavior but they are always worse than Cost_1, but all three never exceed the 100 ms border.
On the other hand all three cost functions reach the same traffic intensity optimality level by performing different strategies. The good performance of Cost_1 confirms also that the least delay path search leads to a high maximum traffic intensity, since Cost_1 is an approximation of the least delay path search. The fact that Cost_4 adapts nicely to network state, shows the potential of this cost function. It is highly probable that the cost function also adapts to other utilization pattern. Cost_7 provide a good traffic intensity performance because Cost_1 and Cost_4 are the underlying concept.



Impact Resource Allocation (Fig. 6.11b-6.11e) We run the evaluation of the Resource Allocation algorithm on the same dataset used in the analysis of the cost functions. We focus at our analysis on the TRAA. This algorithm could be configured with three parameters. These parameter are $TRAA_{nf}$ which specifies after which number of embedded flows the resource allocation algorithm should be started, $TRAA_{dv}$ which limits the delay border movement and $TRAA_{er}$ determines the ratio of edges considered for optimization. In the following we analyze the influence of these parameters.
The first evaluation of $TRAA_{nf}$ (Figure 6.11b) shows that the resource allocation has only a limited influence on the maximum traffic intensity. Nevertheless, there are some small improvements in comparison not using (NO) the resource allocation algorithm. $TRAA_{nf}$ represents the number of processed requests after the TRAA execution is triggered. There is small improvement of the maximum traffic intensity by running the resource allocation every 1000 requests. This setting seems to improve even the runtime behavior. Hence, we chose this value.
The second evaluation of $TRAA_{dv}$ (Figure 6.11c) analyses the delay border movement limitation. The limit of the delay variation to a ratio of 0.001 leads to the highest improvement of the maximum traffic intensity while the influence on the runtime behavior is limited. Hence, we chose this value.
The third evaluation analyses the edge ratio $TRAA_{er}$ at Figure 6.11d. The value determines how many of the high and low utilized edges are considered for optimization. A ratio of 0.1 shows the best performance in terms of runtime and maximum traffic intensity. So we do chose this value.

As a reasonable setting we finally chose $TRAA_{nf}$ = 1000, $TRAA_{dv}$ = 0.001 and $TRAA_{er}$=0.1. We show the performance comparison of this parameter set compared to the case where the resource allocation algorithm is not running in Figure 6.11e. The identified setting outperforms the case where no resource allocation algorithm is used in slighting in runtime maximum traffic intensity. However, the influence of the resource allocation is limited in comparison to the performance impact of the routing algorithm, the Access Control, the burst increase or the cost function dimensions.



Revisit: High Level Sensitivity Analysis (Fig. 6.11f-6.11i) By revisiting the dimensions Access Control, Burst increase and ILS of the initial sensitivity analysis, by only considering the cost functions Cost $_1$, Cost $_4$,Cost $_7$ and the resource allocation parameter $TRAA_{nf}$ = 1000, $TRAA_{dv}$ = 0.001, $TRAA_{er}$=0.1 the following additional conclusions can be drawn.
The use of proper cost functions and resource allocation improves the performance of CBF and LARAC (see. Figure 6.11f). This improvement leads to the fact that there is only a small difference in the maximum traffic intensity performance, comparing the three routing algorithms. However, LARAC outperforms LDP in 70% of the cases. So we can conclude that LARAC is a good candidate for running the QoS framework.
The selection of the high performance cost functions leads to a clear performance picture for the access control (see. Figure 6.11g). The MHM and TBM provide now the expected runtime traffic intensity trade-off. At the cost of a higher runtime it is possible to improve the maximum traffic intensity behavior of the QoS framework, by using the TBM. It also shows that a proper cost function can eliminate the impact of the blocking problem of the TBM mentioned in section 4.6.2.
The behavior of ILS stays, from the performance trade-off perspective, the same as in the initial evaluation (see. Figure 6.11h).
WCB-RR and the REAL burst increase mode provide also a trade-off between runtime and maximum traffic intensity (see. Figure 6.11i). However, since the gain of the REAL mode in terms of maximum traffic intensity is limited, the expenses in extra runtime are not worth it. So the WCB-RR mode provides the best performance for the industrial use case.

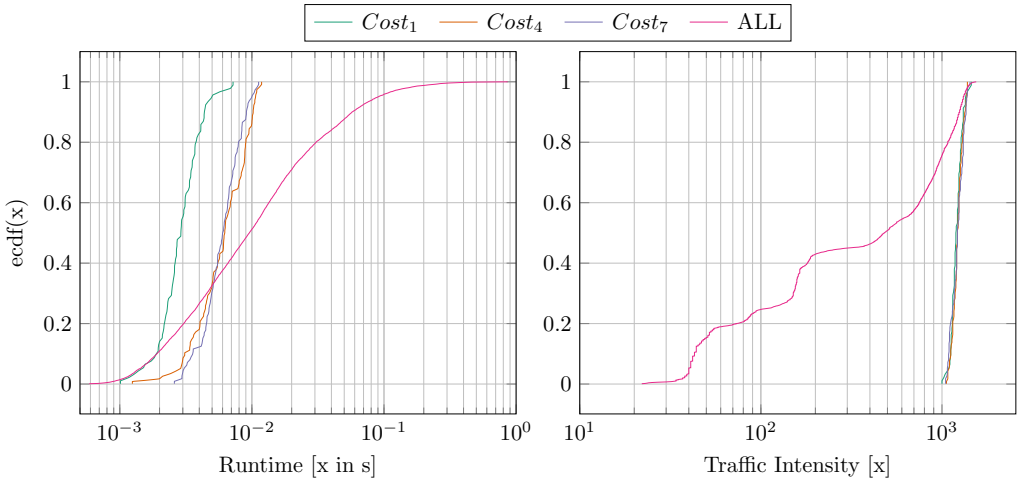
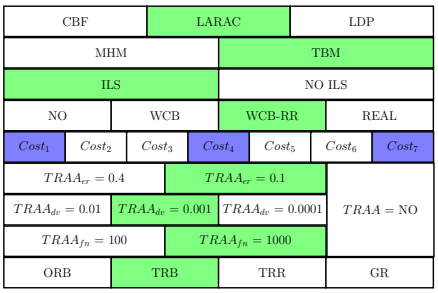


Figure 6.12.: Detailed analysis of the best configuration.

6.3.5. Analysis of the best configuration

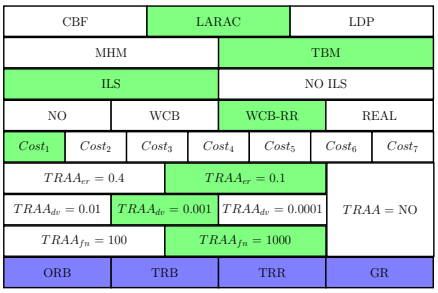


In Section 6.3.3 and 6.3.4 the best configuration for each dimension was found. The goal of this section is the evaluation the performance of the configuration which combines the settings previously found. LARAC is used for routing. The TBM Access Control mechanism is used combined with ILS. The burst increase is covered by the use of the WCB-RR scheme. The resource allocation algorithm is configured with the following parameter set: $TRAA_{nf}$ = 1000, $TRAA_{dv}$ = 0.001 and $TRAA_{er}$=0.1. We rerun the evaluation with this specific configuration because the initial evaluation provides only a limited amount of sampling points. This test shows if the conclusion drawn in section 6.3.3 and 6.3.4 is leading to a good overall setting and is not the result of statistical outliers. The second goal is to test the setting with different topologies. This test will show if the results can be generalized.

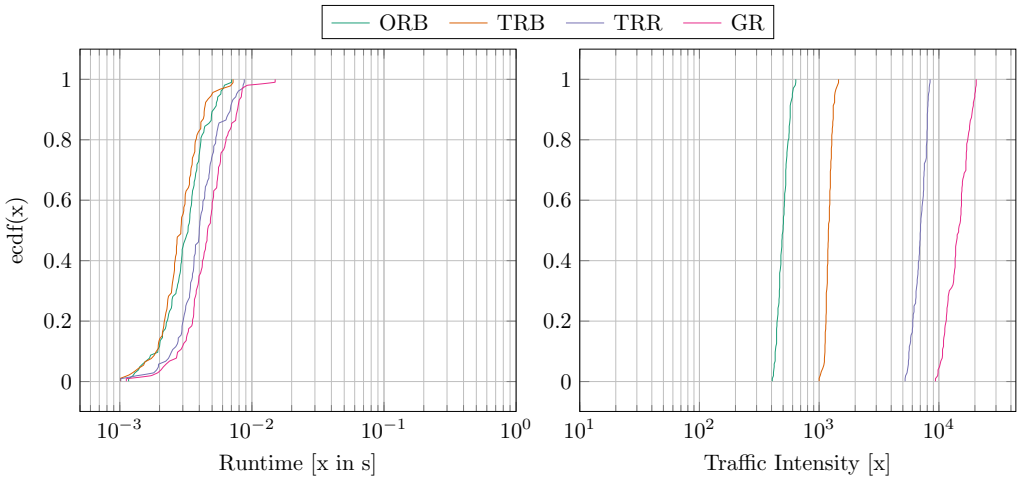
In figure 6.12 the ECDF of the three different settings of the cost function are compared to the data set used in section 6.3.3 and 6.3.4, excluding the data of the NO burst increase setting. All three cost functions provide the same performance in terms of maximum traffic intensity. This behavior confirms the results in our previous evaluation (Figure 6.11a). In addition to this, we notice that combined setting provides a better performance in terms of traffic intensity than 75% of the setting of the previous evaluation. Cost_1 outperforms Cost_2 and Cost_3 by a factor of 2-3 and compared to all settings Cost_1 is faster then 60% of this. These

results are promising, also in absolute numbers. It is possible to achieve an online embedding rate of 100 flows per second with a maximum traffic intensity of more then 1000 flows which meets the requirements of Problem 3: "Sufficient flow capacity". In addition is the performance of 100 flow embeddings per second more then sufficient for an industrial real-time system so Problem 2: "Online flow management" is also covered.

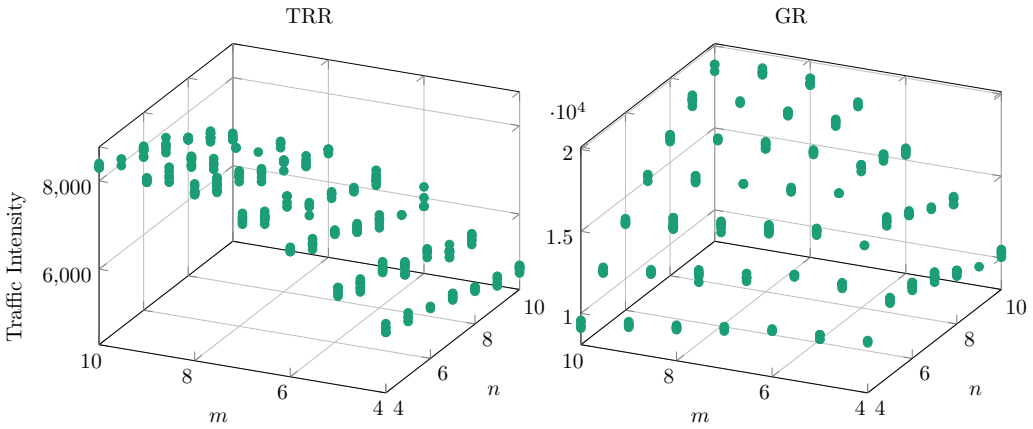
6.3.6. Analysis of the topology impact



In Figure 6.13a the impact of the topology is presented. Therefore, the TRB the ORB , TRR and GR topologies where simulated with cost function Cost_1. The TRR and GR topologies increase the runtime by the factor of one to two compared to the ORB and TRB topologies. The evaluation of the traffic intensity shows also that the usage of Cost_1 leads to an efficient usage of the bottleneck resources. TRB is able to carry the double amount of traffic compared to the ORB. In the less restricted topologies (TRR and GR) the traffic intensity can even reach 4000 - 20000 flows. The variance of the traffic intensity is caused by the different topology sizes (Fig. 6.13b). The maximum traffic intensity scales with the TRR topology dimension m which represents the number of branches in this topology. In contrast, if the number of nodes on a branch is increased, the maximum traffic intensity decreases. This indicates that the branch itself becomes a bottleneck. The GR provides a better traffic intensity if m or n are increased.
These results show that the identified setting provides the performance needed to handle Problem 3: "Sufficient flow capacity" and Problem 2: "Online flow management". There was no drastic performance degradation if the setting was applied for other topologies (ORB , TRR and GR). However, there is no proof that this algorithm selection is also the best choice for the ORB, TRR and GR topology. Therefore, the proposed evaluation method (Section 6.3.2) has to be performed on simulations which reflect the use cases of the QoS framework as accurately as possible. This will lead to the most accurate algorithm selection.



(a) Impact of the topology on runtime and traffic intensity.



(b) Impact of the topology size on the traffic intensity.

Figure 6.13.: Evaluation of the topology impact.

7. Conclusion and Outlook

In this chapter, the conclusions, observations and results of this thesis are discussed. In addition, the several directions for future work are outlined.

7.1. Conclusion

A complete implementation of a centralized deterministic Quality of Service control framework was presented in this Thesis. The implementation was done for the use case of industrial real-time communication. To provide this type of communication, a path with a deterministic end-to-end delay is needed. To enable fast online routing capabilities in packet switched networks a function split was introduced. This split breaks the global optimization problem into three independent optimization problems: the routing problem, cost function design and the resource allocation problem. These three problems work on basis of a Network Resource Model. In this Thesis, multiple deterministic Network Resource Models were presented.
To find a path with a deterministic end-to-end delay Constrained Shortest Path (CSP) routing algorithms are needed. Since there is a substantial amount of CSP algorithms available in the state of the art, but no quantitative comparison, an evaluation of 26 CSP algorithms was done in this thesis. Seven cost functions were introduced to optimize the global optimality of the control system.
The Function Split implies that the dynamic effects of adding and removing flows to and from the network are not effecting previous routing decisions. Therefore, the resource usage at each link has to be preallocated. The task of the resource allocation algorithm is to adapt this preallocation to the latest network state. To solve this task, the Tunable Resource Allocation Algorithm (TRAA) was introduced.
To evaluate the centralized deterministic Quality of Service control framework, a stand alone evaluation of all the components was not sufficient to determine the best system performance. A Monte-Carlo based simulation was used to create a dataset of system configurations. For this configuration the Maximum Traffic Intensity and Life cycle Runtime were calculated. The dataset was used to identify a configuration with a high performance in terms of maximum traffic intensity (more than thousand flows in average) and a suitable calculation time (around hundred of routings per second).

These results can be mapped back to the initial problem statements:

Problem 1 "Deterministic end-to-end delay guarantees": We solve this problem by using a deterministic network calculus based worst case analysis of the queuing delay. This approach provides conservative worst case bounds. We have shown by evaluation that the amount of deterministic real-time connections provided by this method is sufficient for the industrial use case.

Problem 2 "Online flow management": To provide an online industrial QoS framework the function split was introduced. We were able to improve the runtime performance by transforming the routing problem into a CSP problem.

Problem 3 "Sufficient flow capacity": The evaluation shows that 1000 real-time connections could be served to a Programmable Logic Controller (PLC).

Problem 4 "No topology limits": We tested the industrial QoS framework with four scalable topologies. There is no restriction by the design of the framework which prevents a topology layout. However, the topology has an impact on the performance of the framework.

Problem 5 "Commodity Hardware": The framework needs at least Openflow 1.0 and priority scheduling supported by the forwarding elements. A lot of commodity hardware fulfill this requirement.

Problem 6 "Cross traffic": The network resource model is cross traffic enabled. Therefore, cross traffic has to use a lower priority queue.

Problem 7 "Large topologies": The evaluation was performed on topologies of more than 100 forwarding nodes, which meets the initial requirements. However, since the worst case runtime was in the magnitude of ms also larger topologies could be supported.

These results shows that an implementation centralized deterministic Quality of Service control system for packet switched networks is possible, and by using the Function Split it already meets the industrial requirements. This provides the opportunity to use standard networking hardware within the industrial domain, which comes with lower costs in comparison to the specialized hardware used nowadays.

7.2. Future Work

The results of this Thesis is a basis for research in the four main topics. Namely the areas of Routing algorithms, Network Modeling, Cost Functions and Resource allocation have provide a variety of new challenges.
An extensive evaluation of unicast CSP algorithms was done in this thesis. However, there are many other routing problems which occur in real live scenarios:

- Resilient path routing where two link/node disjoint paths have to be found
- Multicast/Broadcast Routing where a one to many communication has to be established
- Multi-Constrained Shortest Path (MCSP) Routing where multiple constraints (e.g. Availability, Delay, ...) have to be satisfied.

The network resource modeling was focused on providing deterministic delay bounds to serve hard real-time requirements. In future work, stochastic delay bounds could be used to serve soft real-time requirements. In addition, both models should coexist which would allow to serve soft and hard real-time requirements at the same time. An additional direction could be the modeling of different communication links in the same routing graph. This would enable to control hybrid communication systems like wired and wireless communication efficiently.
In this Thesis, cost functions and reallocation algorithms are provided. Since the performance of this implementations highly dependents on the network state, machine learning approaches could be used to design an adaptive solution.

A. Appendix

A.1. Implementation

The framework presented in the thesis has been implemented in a modular way that allows to update any component such as network model or routing algorithm in a easy way. Further, the same software modules can be used for proof-of-concept implementation as well as for simulation-based evaluation.

A.1.1. System Overview

Fig. A.1 illustrates the main workflow of the QoS provisioning system which has been defined in section 2. The function split leads to two global workflow loops: resource allocation and routing.
The routing loop gets a connection request from the network (packet-in) or a request for a flow embedding via the northbound interface. The latest state of the network is used to search a path that fulfills the end-to-end delay requirement and provides enough data rate and buffer space. If a path is found this path will be embedded in the network. In addition, the resources consumed by the path will be fed into the queue-link model of the network. This leads to a new network state. A new connection request is not able to use the resources already consumed by a previous one. The routing task is designed as an online activity.
The resource allocation loop uses the current utilization of the network to adapt the resource allocation. As described in section 4 the delay calculation is done by preassigning resources to each queue. The goal of this loop is to optimize the resource allocation by using statistical information of the network state. This should increase the total number of flows or reduce the connection rejection rate. The resource allocation task can be implemented as a background activity.

A.1.2. Component Model

Basis for the software development is the concept of a component-based entity system [157]. An entity is a container object, which can carry one component of every defined type. The component is defined and extended by the user. It is used to carry application specific data in a global context. The system listens to

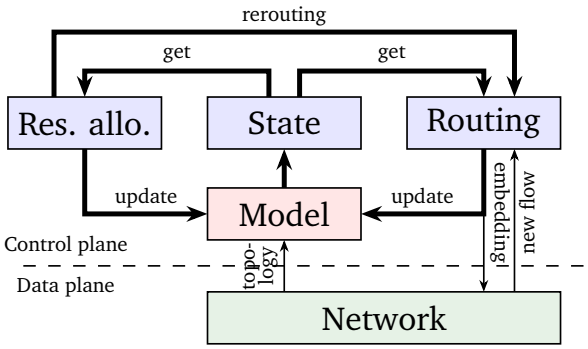


Figure A.1.: Overview over the two main system functions

component driven events. Systems can be implemented such that a method will be called automatically if a specific component is created, updated or destroyed. A basic feature for the system implementation is the queue-link model on which the QoS flow embedding, resource allocation and any other methods are carried out. Therefore, the present simple view of a communication network that is not taking different queues into account, has to be extended towards a queue-link model. For an Software Defined Networking (SDN) controller this is done as an extension to the topology module, which we call topology module extension below. From a software engineering point of view the basis for the topology module extension is the above described component-based entity system. Basic entities are extended with node and edge components to form a graph system. The graph system components are then further extended with queue, delay, rate and scheduler components to form the networking system implementing the queue-link model. On the networking system routing modules for the implementation of different routing algorithms and network modules for the implementation of different network resource models are attached.

A.1.3. Industrial QoS Framework Architecture

This subsection describes the functional view on the framework components implemented using the above described software architecture. Main part is the control architecture that represents the SDN controller. The switching hardware is not modified, i.e., standard Openflow capable switches are assumed.
The control architecture itself is based on a general controller architecture, e.g. such as the OpenDaylight controller platform. For the software component development of this project we introduced the functional module design shown in figure A.2. The most important module of this design is the topology extension module. It provides the basic component based entity system functionality. All modules that are developed for this project implement components and systems

to interact with each other.

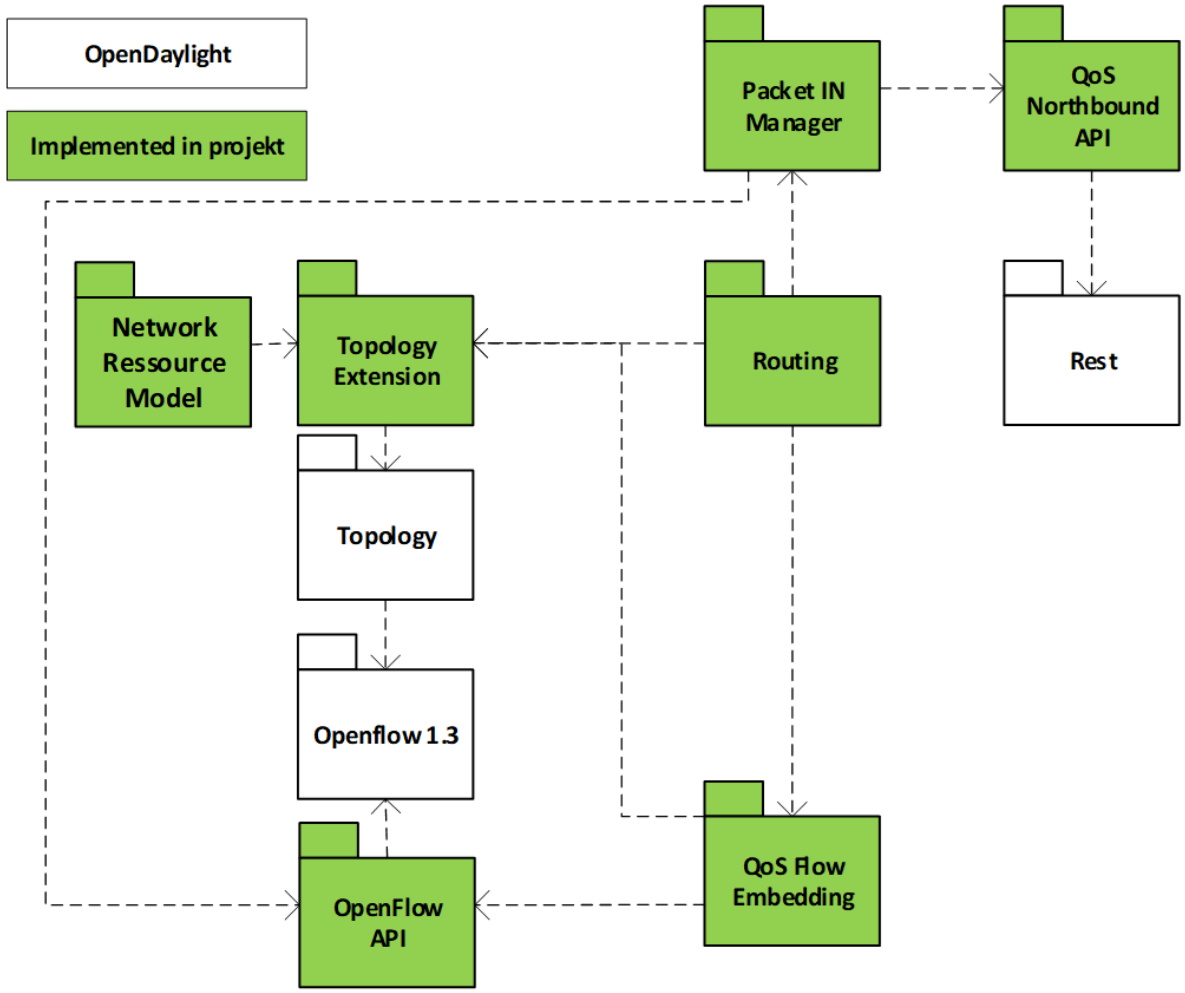


Figure A.2.: Overview of the developed module structure.

The module structure (Figure A.2) will be described in the following:

Topology Extension The topology extension is the main module of the software architecture; it implements the queue-link model of the network topology that is used to calculate the worst-case QoS guarantees needed for flow embedding etc. It is based on the Topology Module provided by any common SDN controller architecture. Two main types of modules are using it: (a) modules that work as a data provided to the queue-link model such as the Network Resource Model and (b) data consumer like the

routing module that use the data to calculate routes for flow embedding inside the network.

Routing The Routing module uses the information provided by the Network Resource Model to provide end-to-end latency guarantees. Therefore, routing requests are generated via the Packet IN Manager. This requests are solved by one CSP routing algorithm. The routing algorithm uses the latency and cost values provided by the network resource model. The final solution is pushed to the QoS Flow Embedding module.

Network Resource Model The Network Resource Model provides deterministic worst case latency borders for every forwarding element. It implements the modeling technique described in Section 4. In addition the implementation of the cost functions (Section 5.2) and the implementation of the resource allocation algorithm (Section 5.3) is part of this module.

Packet IN Manager The Packet IN Manager classifies and distributes the packet-in messages of the controller platform to the routing module. Therefore, service classes have to be configured via the QoS Northbound Application Programming Interface (API).

QoS Northbound API The QoS Northbound API is able to configure the Packet IN manager in a dynamic way. It is possible to set specific real-time flows through this interface based on REST. Therefore, the the flow matching and the QoS requirements have to be provided.

QoS Flow Embedding The QoS Flow Embedding Module takes care of deploying the end-to-end flows provided by the routing modules to the network. The second task is to store all the information of the embedded flows for future processing via the topology extension module.

OpenFlow API The OpenFlow API module is a abstraction layer which profiles easy access to the OpenFlow functionalities of the controller platform. It transforms i.g. routing responses (Paths) into a set of Openflow rules and deploys them in the network.

Abbreviations

kCSP k Constrained Shortest Path. 28, 34

kDCBF k Delay-Constrained Bellman-Ford. 35, 47, 87, 94, 97

kH_MCOP k Heuristic for Multi-Constrained Optimal Path. 40, 47, 87, 91, 93, 95, 97, 100–102

kLARAC k Lagrange Relaxation based Aggregate Cost. 38, 47, 87, 93, 95, 99, 101, 102

kMCP k Multi-Constrained Path. 28

kMCSP k Multi-Constrained Shortest Path. 28, 34, 35, 47, 87, 91, 92

kSP k Shortest Path. viii, 28, 31–33, 40, 95

API Application Programming Interface. 136

AQM Active Queue Management. vii, 8, 11

AUT Algorithm Under Test. 91

AVB Audio Video Bridging. 13

BF Bellman-Ford. viii, 31–35, 42, 47, 87, 94, 98

CBF Constrained Bellman-Ford. 34, 35, 43, 47, 87, 91–94, 98, 99, 102, 113, 118, 125

CD Constrained Dijkstra. 34

CDP Cost Delay Product. 45, 47, 87

CI Cost Inefficiency. 29, 38, 91, 92, 94–98, 100–103

CSP Constrained Shortest Path. viii, ix, 3–7, 19, 28–30, 33–36, 39, 41, 47, 49, 78, 85–89, 91, 93, 95, 97, 99, 101, 103, 111, 113, 129–131, 136

DCBF Delay-Constrained Bellman-Ford. 35, 47, 87, 93, 94, 98, 99, 101

DCCR Delay-Cost-Constrained Routing. 41, 47, 87, 96, 97

DCLC Delay-Constrained Least-Cost. 19, 22, 29–31, 34, 37, 40–43, 47, 49, 73, 75, 87

DCR Distributed delay Constrained Routing. 43, 44, 47, 87, 93, 96, 97, 100, 102, 103

DCUR Delay-Constrained Unicast Routing. x, 35, 42–44, 47, 87, 96, 97, 102, 103

DEB dual extended Bellman-Ford. 35, 47, 87, 101

DetServ Deterministic Services. viii, 3–6, 27, 49–52, 54, 56, 58–62, 64, 66–70, 72

DiffServ Differentiated Services. vii, 1, 8, 9, 14, 15

E_MCOP Exact Multi-Constrained Optimal Path. 41, 47, 87, 91, 92

E_MCP Exact Multi-Constrained Path. 41

EB extended Bellman-Ford. 35

ECDF Empirical Cumulative Density Functions. 116, 126

FB Fallback. 34, 35, 47, 87, 92, 94, 102

FIFO First-in, First-out. 9, 10

FQ Fair Queuing. 10

GPS Generalized Processor Sharing. 8, 10, 13

GR Grid Random. 89, 92, 93, 98, 101, 102, 116, 127

H_MCOP Heuristic for Multi-Constrained Optimal Path. 39–41, 47, 87, 91, 93, 95–97, 100–102

H_MCP Heuristic for Multi-Constrained Path. 40, 41, 95

IkSP Iterative k Shortest Path. 33, 34, 37, 38, 41, 47, 87, 92, 95

IAK Ishida Aman Kannari. 43, 44, 47, 87, 96, 97, 102, 103

ILP Integer Linear Programming. 33

ILS Input Link Shaping. viii, x, 50, 64–71, 112–114, 116, 119, 125, 126

IntServ Integrated Services. vii, 1, 8, 9, 14, 15

IP Internet Protocol. 75

LARAC Lagrange Relaxation based Aggregate Cost. x, 36–39, 42, 47, 87, 93–96, 99, 101–103, 113, 118, 125, 126

LARACGC Lagrange Relaxation based Aggregate Cost Gap Closing. 38, 42, 47, 87, 91–96, 99, 101

LC Least-Cost. viii, 32–40, 42–45, 47, 87, 90, 92–97, 102

LD Least-Delay. viii, 33–45, 47, 87, 91–98, 101–103

LDP Least Delay Path. 34, 47, 77, 87, 91–93, 95, 98, 99, 102, 113, 118, 120, 125

MCP Multi-Constrained Path. 28–30, 40, 41

MCSP Multi-Constrained Shortest Path. viii, 28, 29, 33–35, 39–41, 47, 87, 131

MH_MCOP Modified Heuristic for Multi-Constrained Optimal Path. 41, 47, 87, 95, 96, 101

MHM Multi-Hop Model. viii, x, 50, 56–60, 62, 66–68, 70, 79–82, 112, 119, 125

MIP Mixed Integer Program. ix, 6, 7, 18, 104, 106–108, 110, 111

NBI Northbound Interface. 20

NR_DCLC Nonlinear Relaxation Delay Constraind Least Cost. 40, 41, 47, 87, 95

ORB One Ring Bottleneck. 89, 101, 116, 127

PBO Partition-Based Ordering. 45, 47, 87

PCE Path Computation Element. 14

PLC Programmable Logic Controller. 2, 3, 89, 130

PQ Priority Queuing. 10, 11

QoS Quality of Service. iii, vii–xi, 1–18, 20, 22, 24, 25, 27–30, 32, 33, 42, 45–47, 49, 56, 74, 75, 85–88, 91, 93, 100, 102–104, 106, 107, 110–113, 115–117, 119, 121, 123, 125, 127, 130, 133–136

RDM Residual Delay Maximizing. 45, 47, 87

RL Rate Latency. 52, 53, 56, 63, 64, 66, 68, 143

RR Round Robin. 10

RSVP Resource reSerVation Protocol. 8

SkSP Static k Shortest Path. 33, 38, 40, 47, 87

SCRC Santos Coutinho-Rodrigues Current. 38, 41, 47, 87, 91, 92

SDN Software Defined Networking. vii, 8, 14, 15, 17, 20, 30, 45, 75, 85, 134, 135

SF_DCLC Selection Function Delay Constraind Least Cost. 45, 47, 87, 93, 97, 100, 102

SMS Sriram Manimaran Siva. 44, 45, 47, 87, 92

SP Shortest Path. viii, 28–32, 34–47, 87, 92, 94–97, 102

SSR+DCCR Search Space Reduction Delay-Cost-Constrained Routing. 42, 47, 87, 93, 96, 97, 100–103

TB Token Bucket. 52, 53, 58, 63, 66, 67, 141

TBM Threshold-Based Model. viii, x, 50, 56, 60–62, 67–70, 79, 80, 112, 119, 125, 126

TCP Transmission Control Protocol. 11

TRAA Tunable Resource Allocation Algorithm. ix, xi, 80, 82–84, 113, 115, 121–124, 129, 143, 144

TRB Two Ring Bottleneck. 89, 101, 102, 113, 116, 127

TRR Two Ring Random. 89, 101, 102, 116, 127

TSN Time Sensitive Networking. vii, xi, 7, 12–14

WFQ Weighted Fair Queuing. 10, 11

WRR Weighted Round Robin. 10, 11

Mathematical Notations

$\mathcal{A}$ set of resource allocation parameter. 71, 79–83, 110

α^{TB} Token Bucket (TB) service curve. 52, 53, 58, 63, 66, 67

A^B Buffer capacity alocated to an edge. 57–59, 67, 71, 79, 81, 82, 105

α arrival curve. 51, 52, 58, 66–68, 70

A^R Bit rate allocated to an edge. 57–59, 67, 71, 79–82, 110

A^{WCD} Worst Case Dealy alocated to an edge. 60–62, 68, 69, 71, 79

BP_{max} maximum blocking probability. 74

BP blocking probability. 74, 75

B buffer capacity. 11, 55–58, 61, 62, 67–69, 71

P_f path of a flow f. 18, 19, 50, 55, 56, 59, 61, 69, 71, 75, 76, 81, 103, 106, 142, 143

b burstiness. 8, 52–55, 59, 61, 63, 64, 69, 71, 105, 113

$\lambda_{\mathcal{F}}$ flow f arrival rate. 73, 74, 115

$h_{\mathcal{F}}$ flow f holding time. 73, 115

y traffic intensity. 73, 74, 115

Cost_1 cost function which optimizes the path to a small hop count. 77, 118, 120, 124–127

Cost_2 cost function which optimizes the path to a small hop count path and to a lowest priority. 77, 118, 126

Cost_3 cost function which optimizes the path to lowest priority with a linear function. 77, 118, 126

Cost_4 cost function which optimizes the path to a small hop count path and a high number of possible future flows for a input link for a q-link (m, u, v, q). 78, 118, 120, 124, 125

Cost_5 cost function which optimizes the path to a small hop count path and a high number of possible future flows for a physical link (u, v). 78, 118

Cost_6 cost function which optimizes the path to a small hop count path and a high number of possible future flows for a q-link (u, v, q). 78, 118

Cost_7 cost function which optimizes the path to a small hop count path and a high number of possible future flows for a physical link (u, v) and use a high priority first. 78, 118, 120, 124, 125

$\mathrm{Cost}(P_f)$ end to end cost of a path P_f. 19, 50, 75, 76, 103

Cost cost metric. 19, 50, 71, 75–78, 103, 118, 120, 124–127, 141, 142

$D(P_f)$ end to end delay of a path P_f. 18, 19, 50, 71, 75, 79, 80, 82, 83, 103, 106

$\mathcal{E}$ set of edges of a graph ($\mathcal{G}$). 17, 71, 142, 143

$\mathcal{E}^p$ set of edges of a graph ($\mathcal{G}$). 54, 56, 71, 80, 81

$\mathcal{E}^q_{(u,v)}$ Set of queue links (u, v, q) traversing physical link (u, v). 16, 54–57, 60–62, 69, 71, 80, 81, 83, 105, 106

$\mathcal{E}^q$ set of edges of a graph ($\mathcal{G}$). 54–59, 61, 64–67, 69, 71

v destination of an link. 71, 76, 143

m link leading to the link. 71, 76, 143

q queue of an link. 71, 76, 143

nf number of flows. 76, 78

u source of an link. 54, 55, 59, 61, 63, 64, 69, 71, 76, 143

$\mathcal{F}$ set of flows (f) embedded in the Graph $\mathcal{G}$. 17, 18, 50, 54–57, 71, 80–83, 103, 106

t deadline constraint. 18, 19, 50, 54, 56, 64, 71, 75, 79, 80, 83, 103, 106, 113

d destination. 18, 19, 50, 71, 75, 103, 106

f flow. 17–19, 50, 54–57, 59, 61, 63, 64, 69, 71, 74, 75, 80–83, 103, 105, 106, 113, 115, 141, 142

o origin. 18, 19, 50, 54, 55, 59, 61, 63, 64, 69, 71, 75, 103, 106

$\mathcal{G}$ graph consisting of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$). 17, 71, 142, 143

$\mathcal{G}^p$ graph consists of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$). 54, 71

$\mathcal{G}^q$ graph consists of a set of nodes ($\mathcal{N}$) and a set of edges ($\mathcal{E}$). 54, 71

$L(P_f)$ end to end loss probability of a path P_f. 18, 19, 50, 80, 103, 106

$\mathcal{N}$ set of nodes of a graph ($\mathcal{G}$). 17, 54, 71, 142, 143

α^* output arrival curve. 52, 53

$\mathcal{P}$ set of possible pathes. 18, 19, 50, 71, 75, 103, 106

(u, v) physical link from node u to node v. 16, 54–58, 60–62, 66–69, 71, 76, 78, 80–83, 105, 106, 110, 142

p priority. 54, 76–78

fes feasibility. 76, 78, 83

$\mathrm{p}_{\max}$ maximal priority. 76–78

(m, u, v, q) Queue link from node u to node v via queue q from physical edge m. 67, 69, 71, 76–78, 141

(u, v, q) Queue link from node u to node v via queue q. 54–71, 76–82, 105, 110, 120, 142

C link capacity. 53, 54

R service data rate. 11, 52, 54–58, 60, 62–64, 66–68, 71, 106

l packet length. 53–57, 61, 62, 67, 69

r data rate. 8, 52–55, 59, 61, 63, 64, 69, 71, 113

β^{RL} Rate Latency (RL) service curve. 52, 53, 56, 63, 64, 66, 68

β service curve. 52, 53, 66–68, 70

T service delay. 52–56, 60, 63, 64, 71

$TRAA_{dv}$ Tunable Resource Allocation Algorithm (TRAA) delay variation. 83, 121–126

$TRAA_{er}$ TRAA edge ratio. 83, 122, 124–126

$TRAA_{nf}$ TRAA number of flows. 83, 121, 122, 124–126

$\mathcal{U}$ set of resource utilisation parameter. 71

U^B Buffer capacity utilisation of an edge. 55–59, 61–63, 67, 69, 71, 79, 105

U^R Bit rate utilisation of an edge. 55–63, 67, 69, 71, 79

WCB Worst Case Burst. 55–58, 61, 62, 66–71

WCD Worst Case Delay. 55–63, 66–69, 71, 81, 82

Bibliography

Publications by the author

[1] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 70–76.

[2] J. W. Guck, M. Reisslein, and W. Kellerer, "Model-based control plane for fast routing in industrial QoS network," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, June 2015, pp. 65–66.

[3] ——, "Function split between delay-constrained routing and resource allocation for centrally managed QoS in industrial networks," *IEEE Transactions on Industrial Informatics (TII)*, vol. 12, no. 6, pp. 2050–2061, Dec 2016.

[4] 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.

[5] J. W. Guck, A. V. Bemten, and W. Kellerer, "DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1003–1017, Dec 2017.

[6] A. Van Bemten, J. W. Guck, C. Mas Machuca, and W. Kellerer, "Routing metrics depending on previously traversed edges: Taxonomy and solutions," in *2018 IEEE International Conference on Communications (ICC)*, 2018.

[7] J. W. Guck, W. Kellerer, V. Kulkarni, and J. Riedl, "Device and method for managing end-to-end connections of a network within a central network management entity," Patent, Feb. 2, 2017, WO Patent App. WO2017016610A1. [Online]. Available: http://google.com/patents/WO2017016610A1

[8] ——, "Device and method for managing end-to-end connections," Patent, Mar. 1, 2018, WO Patent App. WO2018036606A1. [Online]. Available: http://google.com/patents/WO2018036606A1

[9] A. Van Bemten, J. W. Guck, P. Vizarreta, C. Mas Machuca, and W. Kellerer, "LARAC-SN and Mole in the Hole: Enabling routing through service function chains," in *2018 IEEE Conference on Network Softwarization (NetSoft) (accepted)*, 2018.

[10] A. Van Bemten, J. W. Guck, C. Mas Machuca, and W. Kellerer, "Joint Dijkstra (JD): Search space reduction for expediting shortest path," in *2018 IFIP Networking Conference (IFIP Networking) (under review)*, 2018.

[11] S. Zoppi, A. Van Bemten, H. M. Gürsu, M. Vilgelm, J. Guck, and W. Kellerer, "Achieving hybrid wired/wireless industrial networks with wdetserv: Reliability-based scheduling for delay guarantees," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2307–2319, 2018.

[12] J. Folmer, D. Pantförder, J. W. Guck, A. Hosseini, and B. Vogel-Heuser, "Realisierung eines konzeptes zur diagnose ethernetbasierter echtzeitkommunikationssysteme," in *Informatik aktuell*. Springer Berlin Heidelberg, 2013, pp. 99–108.

[13] E. Grigoreva, J. W. Guck, and W. Kellerer, "Challenges and research directions in vehicular traffic modelling and uplink in-car scheduling," Technische Universitaet Muenchen, Tech. Rep., Mar 2016.

[14] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, "Towards adaptive state consistency in distributed SDN control plane," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

General publications

[15] J.-D. Decotignie, "Ethernet-based real-time and industrial communications," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1102–1117, 2005.

[16] P. Sharma, S. Banerjee, S. Tandel, R. Aguiar, R. Amorim, and D. Pinheiro, "Enhancing network management frameworks with SDN-like control," pp. 688–691, 2013.

[17] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, R. Figueiredo, D. Morris, M. Pickavet, and P. Demeester, "Implementing quality of service for the software defined networking enabled future internet," pp. 49–54, Sep. 2014.

[18] A. Akella and K. Xiong, "Quality of Service (QoS)-guaranteed network resource allocation via Software Defined Networking (SDN)," pp. 7–13, Aug. 2014.

[19] M. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "PolicyCop: An autonomic QoS policy enforcement framework for software defined networks," pp. 1–7, 2013.

[20] D. Adami, L. Donatini, S. Giordano, and M. Pagano, "A network control application enabling software-defined quality of service," pp. 6074–6079, 2015.

[21] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[22] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for SDN," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, 2014.

[23] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: Better internet routing based on SDN principles," in *Proc. ACM Workshop on Hot Topics in Networks*, 2012, pp. 55–60.

[24] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proc. ACM Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 13–18.

[25] X. Xiao and L. M. Ni, "Internet QoS: A big picture," *IEEE Network*, vol. 13, no. 2, pp. 8–18, 1999.

[26] ——, "The use of RSVP with IETF integrated services," Internet Requests for Comments, RFC Editor, RFC 2210, Sep. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2210.txt

[27] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*. Springer, 1996, pp. 153–181.

[28] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *Wireless Networks*, vol. 1, no. 1, pp. 61–81, 1995.

[29] R. Sivakumar, P. Sinha, and V. Bharghavan, "CEDAR: A core-extraction distributed ad hoc routing algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1454–1465, 1999.

[30] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proc. IEEE INFOCOM*, vol. 3, 1997, pp. 1405–1413.

[31] J. McQuillan, I. Richer, and E. Rosen, "The new routing algorithm for the ARPANET," *IEEE Transactions on Communications*, vol. 28, no. 5, pp. 711–719, 1980.

[32] J. Jaffe and F. Moss, "A responsive distributed routing algorithm for computer networks," *IEEE Transactions on Communications*, vol. 30, no. 7, pp. 1758–1762, 1982.

[33] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Transactions on Communications*, vol. 25, no. 1, pp. 73–85, 1977.

[34] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific Belmont, 1999.

[35] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[36] H.-S. Yang, M. Maier, M. Reisslein, and W. M. Carlyle, "A genetic algorithm-based methodology for optimizing multiservice convergence in a metro WDM network," *IEEE/OSA Journal of Lightwave Technology*, vol. 21, no. 5, p. 1114, 2003.

[37] K. Deb, *Multi-objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001, vol. 16.

[38] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[39] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, 7th ed. Pearson, 2017.

[40] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," in *Proc. IEEE European Workshop on Software Defined Networking*, 2013, pp. 81–86.

[41] D. Cavendish and M. Gerla, "Internet QoS routing using the Bellman-Ford algorithm," in *High Performance Networking*. Springer, 1998, pp. 627–646.

[42] L. Seno, F. Tramarin, and S. Vitturi, "Performance of industrial communication systems: Real application contexts," *IEEE Industrial Electronics Magazine*, vol. 6, no. 2, pp. 27–37, 2012.

[43] T. Skeie, S. Johannessen, and O. Holmeide, "Timeliness of real-time IP communication in switched industrial Ethernet networks," *IEEE Trans. on Industrial Informatics*, vol. 2, no. 1, pp. 25–39, 2006.

[44] P. Gaj, J. Jasperneite, and M. Felser, "Computer communication within industrial distributed environment—a survey," *IEEE Trans. Industrial Informatics*, vol. 9, no. 1, pp. 182–189, Feb. 2013.

[45] C. C. Skiscim and B. L. Golden, "Solving k-shortest and constrained shortest path problems efficiently," *Annals of Operations Research*, vol. 20, no. 1, pp. 249–282, 1989.

[46] C. Pornavalai, G. Chakraborty, and N. Shiratori, "QoS based routing algorithm in integrated services packet networks," *Journal of High Speed Networks*, vol. 7, no. 2, pp. 99–112, 1998.

[47] ——, "Routing with multiple QoS requirements for supporting multimedia applications," *Telecommunication Systems*, vol. 9, no. 3-4, pp. 357–373, 1998.

[48] S. Chen and K. Nahrstedt, "Distributed QoS routing with imprecise state information," in *Proc. IEEE Int. Conf. on Computer Communications and Networks*, 1998, pp. 614–621.

[49] ——, "Distributed quality-of-service routing in high-speed networks based on selective probing," in *Proc. IEEE Conference on Local Computer Networks (LCN)*, 1998, pp. 80–89.

[50] K. G. Shin and C.-C. Chou, "A distributed route-selection scheme for establishing real-time channels," in *High Performance Networking*. Springer, 1995, pp. 319–330.

[51] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Transactions on Networking*, vol. 6, no. 6, pp. 768–778, 1998.

[52] R. A. Guérin and A. Orda, "QoS routing in networks with inaccurate information: Theory and algorithms," *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 350–364, 1999.

[53] S. Chen and K. Nahrstedt, "Distributed quality-of-service routing in ad hoc networks," *IEEE Journal on Selected areas in Communications*, vol. 17, no. 8, pp. 1488–1505, 1999.

[54] L. Xiao, J. Wang, and M. Nahrstedt, "The enhanced ticket-based routing algorithm," in *Proc. IEEE Int. Conf. on Communications (ICC)*, vol. 4, 2002, pp. 2222–2226.

[55] A. Goel, K. G. Ramakrishnan, D. Kataria, and D. Logothetis, "Efficient computation of delay-sensitive routes from one source to all destinations," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 854–858.

[56] Y. P. Aneja, V. Aggarwal, and K. P. Nair, "Shortest chain subject to side constraints," *Networks*, vol. 13, no. 2, pp. 295–302, 1983.

[57] H. C. Joksch, "The shortest route problem with constraints," *Journal of Mathematical Analysis and Applications*, vol. 14, no. 2, pp. 191–197, 1966.

[58] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36–42, 1992.

[59] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 834–843.

[60] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, vol. 28, no. 5, pp. 213–219, 2001.

[61] F. Ergun, R. Sinha, and L. Zhang, "An improved FPTAS for restricted shortest path," *Information Processing Letters*, vol. 83, no. 5, pp. 287–291, 2002.

[62] A. Warburton, "Approximation of pareto optima in multiple-objective, shortest-path problems," *Operations Research*, vol. 35, no. 1, pp. 70–79, 1987.

[63] R. G. Garroppo, S. Giordano, and L. Tavanti, "A survey on multi-constrained optimal path computation: Exact and approximate algorithms," *Computer Networks*, vol. 54, no. 17, pp. 3081–3107, 2010.

[64] H. Agrawal, M. Grah, and M. Gregory, "Optimization of QoS routing," in *Proc. IEEE/ACIS Int. Conf. on Computer and Information Science (ICIS)*, 2007, pp. 598–603.

[65] D. H. Lorenz, A. Orda, D. Raz, and Y. Shavitt, "Efficient QoS partition and routing of unicast and multicast," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1336–1347, 2006.

[66] D. Raz and Y. Shavitt, "Optimal partition of QoS requirements with discrete cost functions," in *Proc. IEEE INFOCOM*, vol. 2, 2000, pp. 613–622.

[67] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Improving QoS routing performance under inaccurate link state information," in *Proc. International Teletraffic Congress*, 1999, pp. 7–11.

[68] Y. P. Aneja and K. Nair, "The constrained shortest path problem," *Naval Research Logistics Quarterly*, vol. 25, no. 3, pp. 549–555, 1978.

[69] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, no. 4, pp. 293–309, 1980.

[70] D. Blokh and G. Gutin, "An approximate algorithm for combinatorial optimization problems with two parameters," *Australasian Journal of Combinatorics*, vol. 14, pp. 157–164, 1996.

[71] A. Jüttner, B. Szviatovski, I. Mécs, and Z. Rajkó, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 859–868.

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

[73] A. Shimbel, "Structure in communication nets," in *Proceedings of the Symposium on Information Networks*, 1954, pp. 199–203.

[74] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, Apr. 1958.

[75] L. R. Ford Jr, "Network flow theory," DTIC Document, Tech. Rep., 1956.

[76] E. F. Moore, *The Shortest Path Through a Maze*. Bell Telephone System, 1959.

[77] A. Schrijver, "On the history of the shortest path problem," *Documenta Mathematica*, pp. 155–167, 2012.

[78] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[79] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quarterly of Applied Mathematics*, vol. 27, no. 4, pp. 526–530, Jan. 1970.

[80] M. J. Bannister and D. Eppstein, "Randomized speedup of the Bellman-Ford algorithm," in *Proc. of the SIAM Meeting on Analytic Algorithmics and Combinatorics*, 2012, pp. 41–47.

[81] L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," *Computers & Operations Research*, vol. 33, no. 11, pp. 3324–3343, 2006.

[82] E. Chow, "A graph search heuristic for shortest distance paths," Lawrence Livermore National Laboratory, Tech. Rep., 2005.

[83] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[84] D. Eppstein, "Finding the k shortest paths," *SIAM Journal on Computing*, vol. 28, no. 2, pp. 652–673, 1998.

[85] V. M. Jiménez and A. Marzal, "A lazy version of Eppstein's k shortest paths algorithm," in *Proc. Int. Workshop on Experimental and Efficient Algorithms*. Springer, 2003, pp. 179–191.

[86] H. Aljazzar and S. Leue, "K*: A directed on-the-fly algorithm for finding the k shortest paths," University of Konstanz, Germany, Tech. Rep., 2008.

[87] E. I. Chong, S. Maddila, and S. Morley, "On finding single-source single-destination k shortest paths," *J. Computing and Information, Special Issue ICCI*, vol. 95, pp. 40–47, 1995.

[88] Z. Jia and P. Varaiya, "Heuristic methods for delay-constrained least-cost routing problem using k-shortest-path algorithms," in *Proc. IEEE INFOCOM*, 2001, pp. 1–9.

[89] G. Liu and K. Ramakrishnan, "A*Prune: An algorithm for finding k shortest paths subject to multiple constraints," in *Proc. IEEE INFOCOM*, vol. 2, 2001, pp. 743–749.

[90] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial time approximation algorithms for multi-constrained QoS routing," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 656–669, 2008.

[91] G. Feng, C. Douligeris, K. Makki, and N. Pissinou, "Performance evaluation of delay-constrained least-cost QoS routing algorithms based on linear and nonlinear lagrange relaxation," in *Proc. IEEE Int. Conf. on Communications (ICC)*, vol. 4, 2002, pp. 2273–2278.

[92] H. F. Salama, D. S. Reeves, and Y. Viniotis, "A distributed algorithm for delay-constrained unicast routing," in *Proc. IEEE INFOCOM*, vol. 1, 1997, pp. 84–91.

[93] D. S. Reeves and H. F. Salama, "A distributed algorithm for delay-constrained unicast routing," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 239–250, 2000.

[94] Q. Sun and H. Langendörfer, "A new distributed routing algorithm for supporting delay-sensitive applications," *Computer Communications*, vol. 21, no. 6, pp. 572–578, 1998.

[95] R. Sriram, G. Manimaran, and C. S. R. Murthy, "Preferred link based delay-constrained least-cost routing in wide area networks," *Computer Communications*, vol. 21, no. 18, pp. 1655–1669, 1998.

[96] K. Ishida, K. Amano, and N. Kannari, "A delay-constrained least-cost path routing protocol and the synthesis method," in *Proc. IEEE Int. Conf. on Real-Time Computing Systems and Applications*, 1998, pp. 58–65.

[97] W. Liu, W. Lou, and Y. Fang, "An efficient quality of service routing algorithm for delay-sensitive applications," *Computer Networks*, vol. 47, no. 1, pp. 87–104, 2005.

[98] W. C. Lee, M. G. Hluchyi, and P. A. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Network*, vol. 9, no. 4, pp. 46–55, 1995.

[99] L. Guo and I. Matta, "Search space reduction in QoS routing," *Computer Networks*, vol. 41, no. 1, pp. 73–88, 2003.

[100] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, "An improved solution algorithm for the constrained shortest path problem," *Transportation Research Part B: Methodological*, vol. 41, no. 7, pp. 756–771, 2007.

[101] F. Xiang, L. Junzhou, W. Jieyi, and G. Guanqun, "QoS routing based on genetic algorithm," *Computer Communications*, vol. 22, no. 15, pp. 1392–1399, 1999.

[102] W. Zhengying, S. Bingxin, and Z. Erdun, "Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm," *Computer Communications*, vol. 24, no. 7, pp. 685–692, 2001.

[103] D. Karaboga and B. Basturk, "Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems," in *Proc. Int. Fuzzy Systems Assoc. World Congress*. Springer, 2007, pp. 789–798.

[104] C. C. Ribeiro and M. Minoux, "A heuristic approach to hard constrained shortest path problems," *Discrete Applied Mathematics*, vol. 10, no. 2, pp. 125–137, 1985.

[105] ——, "A new heuristics for finding the delay constrained least cost path," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 7, 2003, pp. 3711–3715.

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

[107] G. Feng, K. Makki, N. Pissinou, and C. Douligeris, "Heuristic and exact algorithms for QoS routing with multiple constraints," *IEICE Trans. on Commun.*, vol. 85, no. 12, pp. 2838–2850, 2002.

[108] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *Mathematical Programming*, vol. 73, no. 2, pp. 129–174, 1996.

[109] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[110] L. Wolsey and G. Nemhauser, *Integer and Combinatorial Optimization*, ser. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1999.

[111] J. Schmitt, P. Hurley, M. Hollick, and R. Steinmetz, "Per-flow guarantees under class-based priority queueing," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 7, 2003, pp. 4169–4174.

[112] J. Jasperneite and P. Neumann, "How to guarantee realtime behavior using Ethernet," in *Information Control Problems in Manufacturing 2004 (INCOM 2004): A Proceedings Volume from the 11th IFAC Symposium, Salvador, Brazil, 5-7 April 2004*, vol. 1. Gulf Professional Publishing, 2005.

[113] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," in *Telecommunications Forum Telfor (TELFOR), 2014 22nd*. IEEE, 2014, pp. 111–114.

[114] S. Gorlatch, T. Humernbrum, and F. Glinka, "Improving qos in real-time internet applications: from best-effort to software-defined networks," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*. IEEE, 2014, pp. 189–193.

[115] "Communication delivery time performance requirements for electric power substation automation," *IEEE Std 1646-2004*, pp. 1–24, 2005.

[116] Q. Duan, "Network-as-a-service in software-defined networks for end-to-end QoS provisioning," in *2014 23rd Wireless and Optical Communication Conference (WOCC)*. IEEE, 2014, pp. 1–5.

[117] W. Zhao, D. Olshefski, and H. G. Schulzrinne, "Internet quality of service: An overview," 2000.

[118] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001, vol. 2050.

[119] J. Jasperneite, J. Imtiaz, M. Schumacher, and K. Weber, "A proposal for a generic real-time Ethernet system," *IEEE Trans. Industrial Informatics*, vol. 5, no. 2, pp. 75–85, May 2009.

[120] A. L. King, S. Chen, and I. Lee, "The middleware assurance substrate: Enabling strong real-time guarantees in open systems with openflow," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*. IEEE, 2014, pp. 133–140.

[121] P. Gaj, J. Jasperneite, and M. Felser, "Computer communication within industrial distributed environment - A survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 182–189, 2013.

[122] M. Shen, L. Zhu, M. Wei, Q. Zhang, M. Wang, and F. Li, "Joint optimization of flow latency in routing and scheduling for software defined networks," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–8.

[123] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable qos control for network convergence." *INM/WREN*, vol. 10, no. 1, pp. 1–1, 2010.

[124] N. An, T. Ha, K.-J. Park, and H. Lim, "Dynamic priority-adjustment for real-time flows in software-defined networks," in *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International*. IEEE, 2016, pp. 144–149.

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

[126] O. S. Consortium *et al.*, "Openflow switch specification version 1.0.0," 2009.

[127] T. Mahmoodi, V. Kulkarni, W. Kellerer, P. Mangan, F. Zeiger, S. Spirou, I. Askoxylakis, X. Vilajosana, H. J. Einsiedler, and J. Quittek, "Virtuwind: virtual and programmable industrial network prototype deployed in operational wind park," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 9, pp. 1281–1288, 2016.

[128] I. Owens, A. Durresi *et al.*, "Video over software-defined networking (VSDN)," in *16th International Conference on Network-Based Information Systems (NBiS)*. IEEE, 2013, pp. 44–51.

[129] R. Widyono *et al.*, *The design and evaluation of routing algorithms for real-time channels*. International Computer Science Institute Berkeley, 1994.

[130] T. Sauter, "The three generations of field-level networks - evolution and compatibility issues," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3585–3595, 2010.

[131] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards qoe-driven multimedia service negotiation and path optimization with software defined networking," in *SoftCOM 2012, 20th International Conference on Software, Telecommunications and Computer Networks*, Sept 2012, pp. 1–5.

[132] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Signal & Information*

Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific. IEEE, 2012, pp. 1–8.

[133] E. Schweissguth, P. Danielis, C. Niemann, and D. Timmermann, "Application-aware industrial ethernet based on an sdn-supported tdma approach," in *Factory Communication Systems (WFCS), 2016 IEEE World Conference on*. IEEE, 2016, pp. 1–8.

[134] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 307–318.

[135] A. Van Bemten and W. Kellerer, "Network calculus: A comprehensive guide," 2016.

[136] J. Jasperneite, P. Neumann, M. Theis, and K. Watson, "Deterministic real-time communication with switched Ethernet," in *4th International Workshop on Factory Communication Systems*. IEEE, 2002, pp. 11–18.

[137] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[138] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture (rfc 1633)," *IETF*, 1994.

[139] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services (rfc 2475)," *IETF*, 1998.

[140] M. Welzl and M. Muhlhauser, "Scalability and quality of service: a trade-off?" *Communications Magazine, IEEE*, vol. 41, no. 6, pp. 32–36, 2003.

[141] S. J. Vaughan-Nichols, "Openflow: The next generation of the network?" *Computer*, vol. 44, no. 8, pp. 13–15, 2011.

[142] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over openflow networks: An optimization framework for qos routing," in *18th International Conference on Image Processing (ICIP)*. IEEE, 2011, pp. 2241–2244.

[143] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched ethernet," in *Proc. IEEE Euromicro Conf. on Real-Time Systems*, 2004, pp. 13–22.

[144] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electr. Mag.*, vol. 11, no. 1, pp. 17–27, Mar. 2017.

[145] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "A survey on smart grid potential applications and communication requirements," *IEEE Trans. on Industrial Informatics*, vol. 9, no. 1, pp. 28–42, Feb. 2013.

[146] B. W. Carabelli, R. Blind, F. Dürr, and K. Rothermel, "State-dependent priority scheduling for networked control systems," in *Proc. American Control Conference (ACC)*, May 2017, pp. 1–8.

[147] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with SDN network hypervisors: The cost of virtualization," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366–380, Sep. 2016.

[148] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, Jan. 2017.

[149] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325–346, First Qu. 2017.

[150] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks-the multiple node case," in *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE*, 1993, pp. 521–530 vol.2.

[151] ——, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, Jun 1993.

[152] C. Semeria, "Supporting differentiated service classes: queue scheduling disciplines," *Juniper networks*, pp. 11–14, 2001.

[153] G. Thiruchelvi and J. Raja, "A survey on active queue management mechanisms," *International Journal of Computer Science and Network Security*, vol. 8, no. 12, pp. 130–145, 2008.

[154] A. Bianco, J. M. Finochietto, G. Giarratana, F. Neri, and C. Piglione, "Measurement-based reconfiguration in optical ring metro networks," *IEEE/OSA J. Lightwave Techn.*, vol. 23, no. 10, pp. 3156–3166, 2005.

[155] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," *IEEE/ACM Trans. Netw.*, vol. 2, no. 2, pp. 122–136, 1994.

[156] R. Ramaswamy, N. Weng, and T. Wolf, "Characterizing network processing delay," in *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, vol. 3. IEEE, 2004, pp. 1629–1634.

[157] O. Wallentin, "Component-based entity systems: Modular object construction and high performance gameplay," 2014.

Cited websites

[158] The league of routing algorithms. [Online]. Available: http://www.lkn.ei.tum.de/lora

[159] (2017, 11) Time-sensitive networking task group. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[160] (2017, 11) Ieee 802.1. [Online]. Available: http://www.ieee802.org/1/

[161] (2017, 12) Java hotspot whitepaper. [Online]. Available: http://www.oracle.com/technetwork/java/whitepaper-135217.html