Network Slicing for Industry 4.0

Anders Ellersgaard Kalør

Master's Thesis
Networks and Distributed Systems

Aalborg University
June 8, 2017

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Frederik Bajers Vej 7
DK-9220 Aalborg Ø
http://www.es.aau.dk

Title:
Network Slicing for Industry 4.0

Theme:
Master's Thesis

Project period:
February 1–June 8, 2017

Project group:
NDS10-1021

Participant:
Anders Ellersgaard Kalør

Supervisors:
Petar Popovski*
Jimmy Jessen Nielsen*
René Guillame†
Andreas Müller†

Number of pages: 76

Date of completion:
June 8, 2017

Abstract:

Industry 4.0 introduces modern communication and information technologies to industrial manufacturing systems, and it is expected that applications will have end-to-end network requirements ranging from extremely low latency to high throughput. This thesis investigates network slicing as a technology facilitating the heterogeneous application requirements. We study methods for analyzing end-to-end characteristics in industrial networks, and propose schemes for slicing industrial communication technologies which provide different characteristics in terms of utilization, reliability and isolation. Furthermore, we define an abstract representation of industrial networks which decouples the management of industrial communication technologies from the construction network slices. Finally, we propose a method for constructing network slices from application requirements, which demonstrates how end-to-end analysis can be integrated into an algorithm. While there is still much work to be done before network slicing can be introduced in industrial networks, the work presented in this thesis provides insight into the challenges and possibilities involved in slicing industrial communication networks.

*Department of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7, 9220 Aalborg Ø, Denmark.
†Communication and Network Technology (CR/AEX1), Robert Bosch GmbH, Renningen, 70465 Stuttgart, Germany.

Preface

This master's thesis is written as part of the Networks and Distributed Systems program at Aalborg University and is the documentation of the work conducted by the author in the period from February 1 to June 8, 2017.

A significant part of the work has been carried out in the Communication and Network Technology section at Bosch research campus in Renningen, Germany, where I was fortunate to spend three months in an inspiring and innovative environment. I would like to thank René Guillame and Andreas Müller for their support and great supervision during my stay.

I would also like to thank Peter Popovski and Jimmy Jessen Nielsen for their guidance and supervision throughout the semester.

Anders Ellersgaard Kalør

Renningen, Germany
June 8, 2017

Contents

Abbreviations

EBB Exponentially Bounded Burstiness.

EBF Exponentially Bounded Fluctuations.

ETSI European Telecommunications Standards Institute.

MGF Moment Generating Function.

mMTC Massive Machine Type Communications.

NF Network Function.

NFV Network Function Virtualization.

NFV-MANO NFV Management and Orchestration.

PER Packet Error Rate.

QoS Quality of Service.

SDN Software Defined Networking.

SNC Stochastic Network Calculus.

URLLC Ultra-Reliable and Low Latency Communications.

VNE Virtual Network Embedding.

VNF Virtual Network Function.

1 Introduction

Since the introduction of mechanization in industry in the late 18th century, the manufacturing industry has undergone several transformations, most notably electrification (second industrial revolution) and digitalization (third industrial revolution). The term *Industry 4.0* was introduced in 2011 and refers to the fourth industrial revolution which introduces modern and emerging information and communication technologies, such as 5G wireless communications, cloud computing, and virtualization, to industrial manufacturing systems [1, 2]. This introduces several benefits such as high flexibility and traceability, and opens new business models for the manufacturing industry [2–4].

Although communication networks have been in part of manufacturing systems for several years, they have traditionally been used exclusively for control and supervision of the manufacturing process in a static configuration [5]. Industry 4.0 causes a transition towards a cloud architecture where the network not only provides the means for transportation of data between devices, but also provides a large elastic pool of configurable resources including communication, computation, storage, applications and services [6–8]. This brings a high degree of flexibility to the production lines as it allows them to change and scale without reconfiguring the control systems and modifying the physical infrastructure. For example, a personalized medicine manufacturing process may gather patient information directly from the patient database, and an optical inspection system may offload image analysis to the cloud.

Along with the introduction of cloud computing, it is envisioned that the number of interconnected physical devices increases drastically, and that the devices continuously will interact with the cloud in order to act intelligently and flexibly. This causes the communication networks to serve a very diverse set of users with a mixture of best-effort traffic, and applications which require end-to-end latency in the order of milliseconds, or even microseconds, and a reliability up to the order of $1 - 10^{-9}$ ("nine nines") [9]. Furthermore, as wireless communication technologies become more reliable, it is expected that they will, partially or completely, replace wired technologies which are currently dominating industrial networks [9]. This introduces a high degree of dynamics in the network due to varying channel conditions and moving devices. This in turn leads to varying data rates and reliability guarantees, as well as constantly changing end-to-end traffic paths. Therefore, dynamically adapting the network resources to the current conditions is required to maintain the strict Quality of Service (QoS) needed in industrial networks. Here, QoS refers to the properties of a service which may include, but are not limited to, latency, packet error rates, throughput and security properties.

The overall communication scenarios in Industry 4.0 are covered by the Massive Machine Type Communications (mMTC) and Ultra-Reliable and Low Latency Communications (URLLC) scenarios envisioned for 5G cellular systems. Hence, much attention has been drawn towards ideas and concepts investigated for 5G, and to enable 5G technologies for industrial systems. In this context, one promising enabling technology is network slicing, which addresses the problem of serving applications with diverse end-to-end QoS requirements in the same network [10–14]. Network slicing refers to the process of *slicing* the physical network into logical sub-networks which provide certain characteristics [15]. For instance, one network slice may offer very low latency information access by allocating storage and computation resources

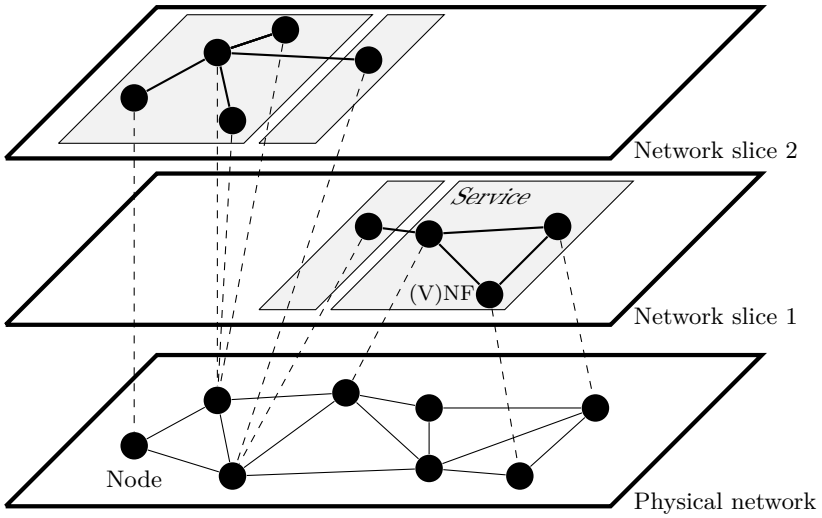


Figure 1.1: The network slicing concept.

for deploying a cache close to the user, and by reserving communication resources in the network. Similarly, another network slice in the same physical network may offer best-effort communication between sensors and a cloud service. Hence, a network slice consists of communication resources, a set of Network Functions (NFs), i.e. processing functions which are executed within a network, and resources to run these functions (Figure 1.1). NFs include traffic policers, firewalls and load balancers, but may also be extended to storage and servers, or control systems in the context of Industry 4.0.

Two technologies strongly facilitate the creation of management and of network slices; Software Defined Networking (SDN) an Network Function Virtualization (NFV). SDN allows for separation of the forwarding of messages and the specification of how to forward messages [16]. In the context of network slicing it facilitates the routing between NFs, as well as queue management. NFV refers to the technology that allows for virtualization of NFs, and is illustrated in Figure 1.2. The physical plane (also referred to as the substrate network) contains the physical components in the network, such as servers, storage, and communication technologies. The virtualization plane consists of virtual resources which are logical partitions or groupings of the underlying physical components. A virtual resource may be a virtual machine consisting of a certain amount of computational resources and memory, a portion of a physical storage medium, or a virtual network with a certain capacity. Finally, the service plane contains a number of Virtual Network Functions (VNFs), which are constructed from the virtual resources. The three planes are managed by a NFV Management and Orchestration (NFV-MANO) mechanism which is responsible for allocating resources based on application requirements and for configuring the physical and virtual resources that provide the network functions.

1.1 Network Slicing in 5G Cellular Networks

As network slicing has already been considered for 5G cellular networks, much work can be obtained from what has already been done in this context. In the current state of the 3GPP 5G standardization process, network slicing is a central part of the architec-

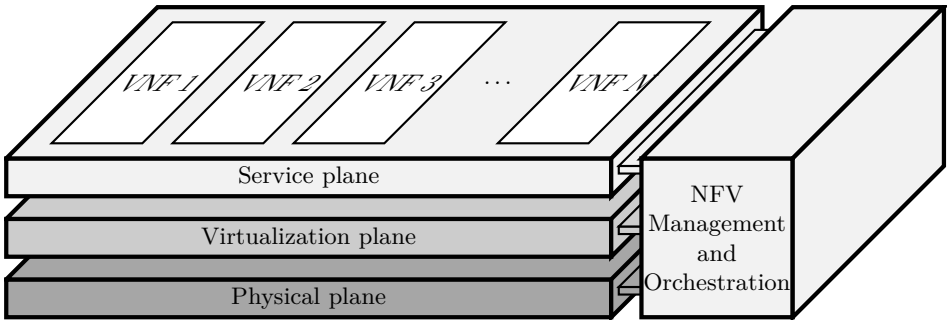


Figure 1.2: Three abstraction layers in Network Function Virtualization.

ture [10], where it provides the means for end-to-end QoS, as well as isolation between different services. For instance, a slice could be constructed to serve all flows which seek information in a specific database in the cloud, while another slice could be constructed to provide a non-critical high-throughput service such as video streaming. In order to do so, different network slices may be based on different technologies, so that a mMTC slice uses another access technology than the slice for e.g. smartphones. Furthermore, network slices can be composed with a minimum number features required to provide a certain service so that the user equipment does not have to support all features in order to use the network. This increases the range of devices that can possibly make use of 5G networks. Finally, network slicing can be used to evaluate new technology in a fully deployed network without influencing the service provided by other slices.

Network slicing also introduces new business models and partnerships between actors in the business ecosystem [11]. For instance, end-to-end slicing enables infrastructure-as-a-service, where operators, or connectivity providers, rent one or more slices of the physical network which they can provide services on top of. The individual network slices may be managed by the infrastructure provider, the connectivity provider, a content provider, etc. As an example, a content provider, e.g. a video on demand company or a content-delivery network provider, may optimize their own network slice for providing a good user experience.

Although network slicing is a central element in the 5G architecture, there are several open challenges which are unlikely to be considered in the standardization. This includes anything that is not related to architecture, communication protocols, or interfaces, such as which network slices to construct, and how to allocate resources in order to provide certain end-to-end characteristics. Nevertheless, there exists a vast of literature on the subject, and even reference implementations for realizing network slicing. However, the underlying networks and the end-to-end requirements in Industry 4.0 differ significantly from those considered in a traditional 5G cellular networks, even with the introduction of mMTC and URLLC.

1.2 Network Slicing for Industry 4.0

While much work done in the context of 5G apply to industrial networks as well, it is important to recognize the areas where the use-cases and requirements for industrial networks and Industry 4.0 differ from those currently considered for 5G.

Figure 1.3 illustrates an example of network slicing in an industrial setting. The

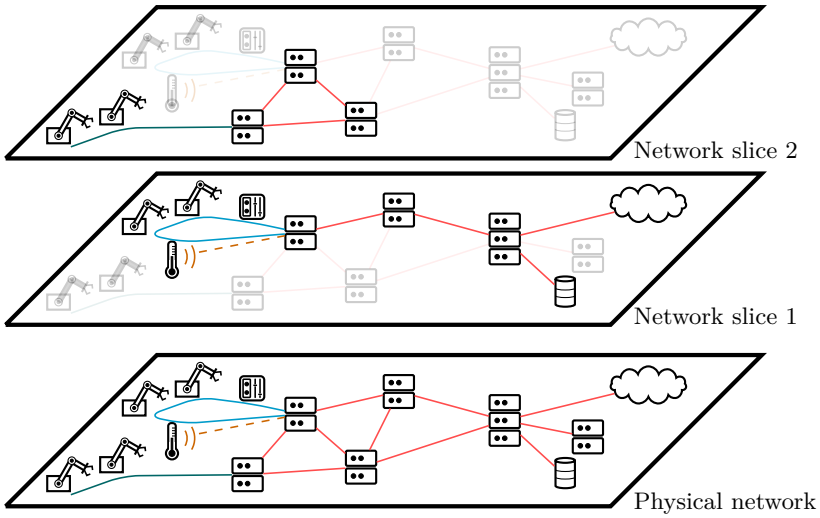


Figure 1.3: The network slicing concept in an industrial setting where a physical network is sliced into two network slices.

physical network consists of various physical components which are interconnected through various communication technologies (indicated by colors). The computation and storage resources are connected to the cloud through a high-capacity network while machines and instrumentation are interconnected by industrial communication technologies such as Industrial Ethernet. In *Network slice 1*, a collection of devices sharing a controller are connected to the cloud and to local storage. Furthermore, sufficient computational resources and link capacity are allocated to provide the desired service. In *Network slice 2*, computation resources are allocated for the control of two robotic arms. Notice that one of the physical servers is shared by both network slices. This may be the case if the virtual network function is used by both slices, or if both slices have a virtual network function which is running on same the physical server. Other use-cases of network slicing in Industry 4.0 include a scenario where a supplier of some component or a machine needs to do condition monitoring or update the firmware. It is desired that these operations do not influence the overall manufacturing process, and hence is isolated from the control network.

Splitting a physical network into network slices is a nontrivial and multifaceted task. From a technological perspective, the necessary steps to implement slicing of industrial networks include:

Abstraction/virtualization of industrial networks. Since industrial networks consist of multiple technologies, and hence constitute a heterogeneous network, defining a common abstract representation of the network characteristics is required to simplify the creation and use of network slices.

Realization of NFV. While virtualization of a wide range of network functions is already possible, functions that are specific to industrial networks are limited. This includes virtualization of industrial Ethernet switches, etc., where reliability requirements are very strict.

End-to-end QoS analysis. Due to the strict latency and reliability requirements in industrial networks, there is a need for analyzing end-to-end characteristics in heterogeneous networks at high percentiles of their distributions. However,

providing end-to-end guarantees in computer networks without significant resource overprovisioning is difficult due to traffic pattern uncertainties, queuing, and interaction between traffic flows. This is particularly true close to the edge of the network where the number of traffic sources is low. Furthermore, the strict requirements set by industrial applications mean that there is little margin for unexpected and rare events that may influence the service provided by the network, and hence heuristic provisioning based on experience, simulation or emulation is insufficient and time consuming.

Construction of methods for slicing networks. Network slicing as currently considered for 5G does not target industrial protocols, and hence there is a need to design slicing mechanisms for these protocols. Furthermore, the strict requirements of industrial applications need to be taken into account in this process.

Infrastructure realization. While network slicing may be implemented on top of existing deployed networks, the full potential requires an infrastructure built around network slicing. Considering the conservatism of the manufacturing industry, this is likely to take several decades. Therefore, methods for allowing network slicing within an existing infrastructure need to be considered.

While all of the above points need to be addressed to introduce network slicing in Industry 4.0, some of them are more critical than others. For instance, without being able to characterizing end-to-end QoS, it is very unlikely that network slicing will be accepted by the industry. On the other hand, NFV is less critical for an initial realization of network slicing, but increases the usefulness. Furthermore, some functionality depend on other elements in order to be realized. For instance, it is difficult to dynamically slice heterogeneous networks without having a common abstraction level for describing the networks. This thesis studies the task of providing end-to-end QoS in industrial networks as outlined by the following problem statement.

1.3 Problem Statement

Providing end-to-end guarantees in heterogeneous networks is a critical part of slicing industrial networks, and the requirements in industrial networks are much more strict than those that are typically considered in the context of network slicing and end-to-end QoS. The strict guarantees causes the impact of rare events to be significant, and hence traditional methods based on simulation and emulation of the systems are not suited for this use-case. Instead, a structured method for allocating slices that guarantee that the requirements are fulfilled is needed. Prerequisite for this is to analyze end-to-end properties in heterogeneous networks, and to define an abstraction layer that allows network slices to be constructed across heterogeneous network technologies. Furthermore, since industrial communication protocols are often based on statically allocated, deterministic resources, strategies for slicing these protocols need to be investigated. Finally, due to the high number of expected slices and the diversity of their requirements, potential ways of automating the construction of networks slices should be considered.

1.4 Organization of the Thesis

This thesis studies the problem defined above by investigating the characteristics of industrial networks that are important to the construction of network slices, and analyzes potential strategies for slicing industrial networks at the protocol level. Furthermore, it examines how this can be done in a way that provides sufficient end-to-end guarantees. In order to construct network slices in an automated fashion, an abstraction framework for describing industrial networks is defined and a heuristic based algorithm for constructing end-to-end network slices with strict guarantees to latency and reliability is proposed and evaluated.

The report is organized as follows; Chapter 2 provides an overview of the state-of-the-art in the field of network slicing, and reviews enabling technologies and notable proposed architectures. Chapter 3 introduces deterministic and stochastic network calculus, a potential tool for analyzing end-to-end delays in computer networks, and discusses possible uses of the calculus in the context of network slicing. A number of industrial communication protocols are examined in Chapter 4, and possible strategies for slicing them, and methods for analyzing end-to-end properties are discussed in Chapter 5. Chapter 6 introduces abstraction frameworks for describing physical industrial networks and network slice requirements in a generic way that can be used for allocating network slices. With basis in these frameworks, an algorithm for allocating communication resources for network slices is presented and evaluated in Chapter 7. Finally, the findings are summarized and discussed in Chapter 8, and Chapter 9 concludes the thesis.

2 Background and Related Work

Network slicing has recently been studied extensively in the context of 5G cellular networks where the need of being able to serve heterogeneous use-cases within the same network has given rise to interest in the topic. Furthermore, recent advances in virtualization and software defined networking have made the realization of network slicing accessible.

The most significant related concepts and enabling technologies for network slicing are illustrated in Figure 2.1. This chapter provides a brief overview of these subjects and review the current state-of-the-art. Furthermore, it examines some recent software initiatives that aim to realize network slicing.

2.1 Software Defined Networking

SDN separates the forwarding of packets in a network (the data plane) from the specification of how packets should be forwarded (the control plane) [16–19]. This may include configuring of queues, and, to some extend, traffic shapers. Furthermore, SDN introduces (centralized) programmability of the control plane so that the forwarding, queues, etc. can be configured using an API. This way, SDN is an enabling technology for network slicing since it allows a controller to manage routing between the VNFs in the network [20]. An SDN enabled switch may itself be a VNF, or a piece of dedicated hardware.

2.2 Hardware Virtualization

Virtualization refers to the mapping between abstract resources at the same abstraction level [21, 22]. For instance, virtual memory provides a mapping from a continuous memory address space to non-continuous memory segments. The virtual memory space is represented in the same way as the non-continuous memory segments, namely as addressable memory. Similarly, hardware virtualization refers to the process of mapping the abstract platform provided by the physical hardware to one or more (virtual) abstract hardware platforms that allow for running software.

Virtualization of hardware has gained popularity in cloud computing, where it enables the possibility to execute multiple virtual machines one the same physical machine while providing high isolation between the virtual machines. The virtual ma-

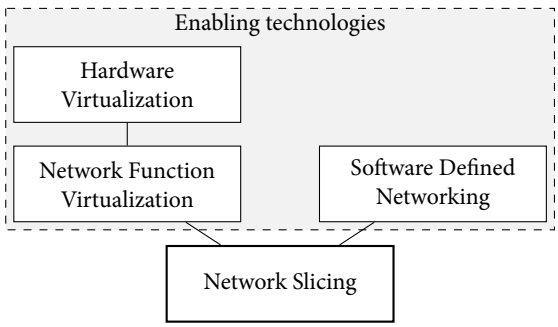
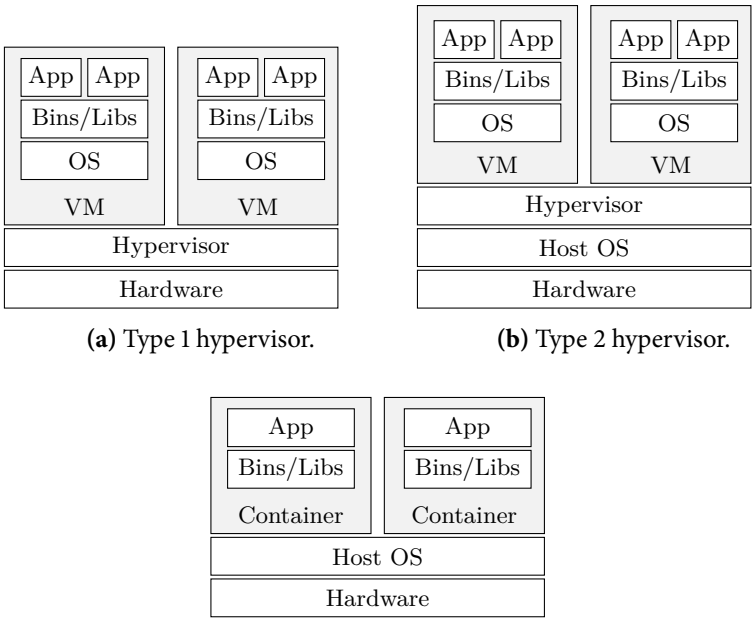


Figure 2.1: Enabling technologies for network slicing.



(a) Type 1 hypervisor.

(b) Type 2 hypervisor.

(c) Container.

Figure 2.2: Virtualization architectures.

chines are typically executed by a hypervisor, which schedules and maintains the virtual machines [22]. A hypervisor can either run directly on the hardware (type 1 hypervisor), or be executed in an operating system (type 2), as illustrated in Figures 2.2a and 2.2b. In both cases, the hypervisor provides an interface to the virtual machine equivalent to that provided by a physical machine. Type 1 hypervisors are typically more efficient than type 2 hypervisors, and can provide higher security and reliability. For these reasons, type 1 hypervisors are typically used in server and cloud environments, while type 2 hypervisors mainly are used in client systems.

An alternative to hypervisors is containers (Figure 2.2c), which allow virtual machines to be executed within a host operating system. In this setting, the virtual machines share the system libraries and other services provided by the host such as the network interfaces and the file system [23]. Therefore, containers are more lightweight constructs, but provides less isolation compared to hypervisors.

While typical VNFs do not have hard deadlines, the desire to bring virtualization to time-critical systems, such as industrial control systems adds several requirements to the method of virtualization [24–26]. Since traditional virtual machines are often constructed to provide a good average performance, much work has been focused on allowing virtualization of soft real-time systems, i.e. systems where missed deadlines lead to degraded service but not complete system failures as in hard real-time systems [26]. Soft real-time schedulers have been proposed for the Xen hypervisor [27–29] which significantly improve the number of met deadlines. For instance, the RT-Xen scheduler presented in [28], is able to meet all deadlines of a set of periodic tasks while maintaining a utilization of 78%, whereas the default Xen scheduler (credit) misses 10% deadlines at a utilization of 22%.

Current virtualization architectures are, however, not well suited for hard real-time systems [3, 25, 26]. In [25] it is argued that although there are good reasons for virtualizing embedded systems, it requires significant changes to the virtualization and

operating system technologies to fully achieve the benefits of virtualization. Specifically, it is highlighted that the hypervisors are black boxes from an application point of view, which limits the possibility to interact with the scheduler. Furthermore, since current hypervisors are designed to run an operating system, they provide a complex interface which puts demands on the complexity of the client software. This, in turn, makes it difficult to ensure ultra-high reliability of the client and limits the system predictability [3]. Several hypervisors for embedded systems have been proposed in literature, which are often very lightweight and aim to be deterministic [30, 31]. However, virtualization may introduce increased variance of memory access times, etc. due to multiplexing, which makes it more difficult to verify that deadlines can be met [32]. This is particularly true when the other guests running on the physical machine are unknown which is often the case in cloud environments.

2.2.1 Network Function Virtualization

NFV refers to the virtualization of functions, typically routing, traffic shaping or computation capabilities, which are executed in a network in order to provide a service. Although NFV and SDN are similar in the sense that they both introduce programmability of the control plane in a network, SDN targets the handling of packets while NFV decouples network functions from specialized hardware [33]. Furthermore, SDN does not necessarily imply virtualization while this is the main principle of NFV. NFV has been heavily studied in the context of 5G and future cellular networks where it is provisioned to introduce flexibility, increase reliability and scalability, and to be an enabler for network slicing and hence the support of heterogeneous user requirements [12–14].

Possible architectures of NFV are studied in [33] which also includes a discussion of challenges, related concepts such as Virtual Network Embedding (VNE), and an overview of standardization activities. The European Telecommunications Standards Institute (ETSI) defines a NFV architectural framework consisting of a NFV Infrastructure (NFVI), Virtualized Network Functions (VNFs), a Operations Support Systems and Business Support Systems plane (OSS/BSS), and a NFV-MANO plane as depicted in Figure 2.3 [34]. The OSS/BSS plane refers to an operator, which is responsible for creating and managing the network including VNFs. The individual VNFs are managed by an Element Management System (EMS) and interact with virtual resources in the NFVI. Lastly, the NFV-MANO plane is responsible for constructing and managing the network, and interfaces all planes in the architecture.

The ETSI further defines the concepts of VNF Forwarding Graphs (VNF-FG), describing the connectivity between VNFs, and a VNF Set, describing VNFs where connectivity does not matter. To this extend, a network service is a forwarding graph of network functions. It is emphasized that the forwarding graphs may be nested. For instance, a network service may use a service which can be independently described by another forwarding graph. In Figure 2.3 the VNF-FG is part of the Service, VNF and Infrastructure Description, and is considered to be a relatively static.

2.3 NFV-MANO Implementations

Several open source NFV-MANO implementations are currently being developed, most notably Open Source MANO [35, 36] and OPEN-O [37]. Open Source MANO

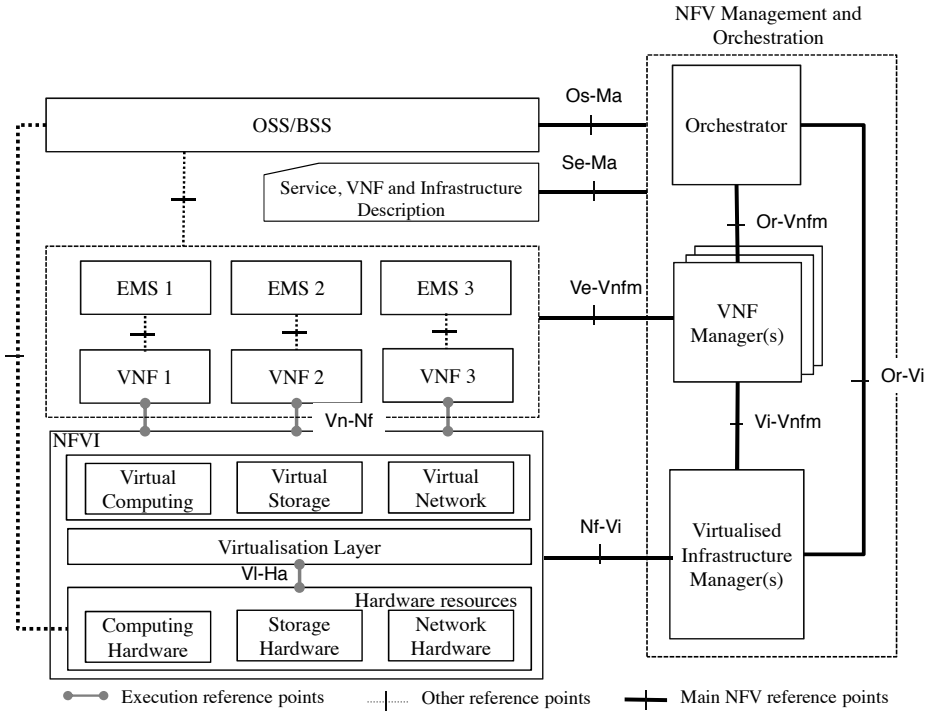


Figure 2.3: ETSI NFV architectural framework (from [34]).

aims to provide a production ready reference implementation of the ETSI NFV-MANO architecture, and is hosted by the ETSI and supported by several vendors. Furthermore, the project is used as a way to evaluate how the ETSI architecture works in practice, and hence provides feedback to the standardization process. The goal of OPEN-O is to provide orchestration of end-to-end services over any network, and while it is compliant with the ETSI NFV architecture, it targets a much wider range of networks. OPEN-O is hosted by The Linux Foundation and is also supported by several telecommunication vendors. Both projects are built upon several existing open source projects such as OpenStack [38] for virtual infrastructure management, OpenDaylight [39] and ONOS [40] for SDN control. While both Open Source MANO and OPEN-O are interesting in the context of realizing network slicing and provide tools for monitoring end-to-end network characteristics, they do not attempt to answer the question of how network slices should be created, or how many resources to allocate to individual network slices.

Several other projects aim to realize NFV, most notably OPNFV [41], which facilitates the development of open source projects which can be used for VNF. The ultimate goal of OPNFV is to construct an open reference platform for NFV.

3 Network Calculus

While many technologies facilitate the instantiation of network slices, the task of deciding which resources to allocate so that certain end-to-end properties are achieved is not well studied, in particular not in the context of industrial systems. While the strict latency requirements in industrial networks have traditionally been handled by deterministic master/slave communication protocols, introducing NFV and cloud computing is likely to introduce switched Ethernet and queuing based technologies in the network as well. Providing end-to-end QoS and analyzing end-to-end latency in queuing networks has been investigated for several decades, and several frameworks such as queuing theory and network calculus have been proposed. While queuing theory relies on Markov models and allows for probabilistic latency bounds, it is often not practical to use for analyzing networks with distributions that are not memoryless. Therefore, unless a memoryless distribution models the system well, queuing theory often introduces a high degree of uncertainty about the obtained results. Network calculus, on the other hand, allows for obtaining latency bounds in networks with a wide range of deterministic or stochastic arrivals. However, this comes at the cost of conservative bounds and results in low network utilization. Nevertheless, since safety is often a concern in industrial networks and conservative requirements are strongly favored over uncertainty, network calculus is more likely to provide a useful framework.

This chapter provides an overview of deterministic and stochastic network calculus, which allows for obtaining worst-case bounds in deterministic networks and stochastic bounds in stochastic networks, respectively. The content is based on the guide by Fidler & Rizk [42]. It first introduces deterministic network calculus and then extends it to a stochastic setting. Finally, it discusses the uses and limitations of the frameworks.

3.1 Deterministic Network Calculus

Deterministic network calculus considers deterministic arrival and service processes. These are described as discrete cumulative functions $A(t) \geq 0$ and $S(t) \geq 0$, respectively. $A(t)$ describes the number of bytes that have arrived in the interval $[0, t]$. Similarly, $S(t)$ is the service that has been offered by the system, also specified in bytes. For convenience we define $A(\tau, t) = A(t) - A(\tau)$ and $S(\tau, t) = S(t) - S(\tau)$. Affine arrival and service curves are the most commonly used functions and are written on the form [43]

$$A(t) = [\sigma_A t + \rho_A]_+, \tag{3.1}$$

where $[x]_+ = \max(x, 0)$. Such an arrival curve sends ρ_A bytes in a burst but no more than σ_A bytes/second in the long run. Hence, $A(t)$ represents a traffic flow which has passed through a leaky bucket with a token rate σ_A and token buffer of size ρ_A. Similarly for service curves we have

$$S(t) = [\sigma_S t - \rho_S]_+. \tag{3.2}$$

The affine curve is often used as an upper bound on linear and sub-linear arrival curves and as a lower bound for linear and super-linear service curves.

The cumulative departures of a system are denoted by $D(t)$. Let τ^* denote the be-

ginning of the last busy period. Then $D(\tau^*) = A(\tau^*)$. Since the system is busy in the interval $[\tau^*, t]$, the number of departures since time τ^* is exactly $S(\tau^*, t)$, from which it follows that

$$D(t) = A(\tau^*) + S(\tau^*, t). \tag{3.3}$$

Since τ^* is not generally known, one can obtain a bound on $D(t)$

$$D(t) \geq \min_{\tau \in [0,t]} \{A(\tau) + S(\tau, t)\}. \tag{3.4}$$

However, the number of departures in $[\tau, t]$ is bounded by the service, $D(t) \leq D(\tau) + S(\tau, t)$, and hence $D(t) \leq A(\tau) + S(\tau, t)$. It follows that there is equality in (3.4) and we obtain

$$D(t) = \min_{\tau \in [0,t]} \{A(\tau) + S(\tau, t)\}. \tag{3.5}$$

This operation is often denoted by the operator $\otimes$ so that $D(t) = A \otimes S(t)$. It is straightforward to show that the operator is associative and as a consequence allows the service rate of n tandem servers to be represented as $S(\tau, t) = S_1 \otimes S_2 \otimes \ldots \otimes S_n(\tau, t)$. In particular, by causality

$$S_1 \bigotimes S_2(\tau, t) = \min_{v \in [\tau,t]} \{S_1(\tau, v) + S_2(v, t)\}. \tag{3.6}$$

Note that if $A(t)$ and $S(t)$ provide upper and lower bounds on the arrival and service processes, respectively, then the departures are bounded by [44]

$$D(t) \leq \max_{\tau \geq 0} \{A(t + \tau) + S(\tau)\}. \tag{3.7}$$

For affine bounded $A(t)$ and $S(t)$ in (3.1) and (3.2), this yields

$$D(t) \leq \sigma_A t + \rho_A + \frac{\sigma_A \rho_S}{\sigma_S}, \tag{3.8}$$

which is again an affine bounded arrival process, but with the burst increased by $(\sigma_A \rho_S)/\sigma_S$.

The backlog, i.e. the number of bytes that is buffered or in transmission, is given by

$$B(t) = A(t) - D(t). \tag{3.9}$$

It follows that

$$\begin{aligned} B(t) &\leq \min_{\tau \in [0,t]} \{A(\tau) - A(\tau) - S(\tau, t)\} \\ &\leq \min_{\tau \in [0,t]} \{A(\tau, t) - S(\tau, t)\}. \end{aligned} \tag{3.10}$$

Assuming first-come-first-served scheduling, the waiting time is given by

$$W(t) = \min \{w \geq 0 : A(t) - D(t + w) \leq 0\}. \tag{3.11}$$

Insertion of (3.5) gives

$$W(t) = \min \left\{ w \geq 0 : A(t) - \min_{\tau \in [0,t]} \{A(\tau) + S(\tau, t + w)\} \leq 0 \right\} \tag{3.12}$$

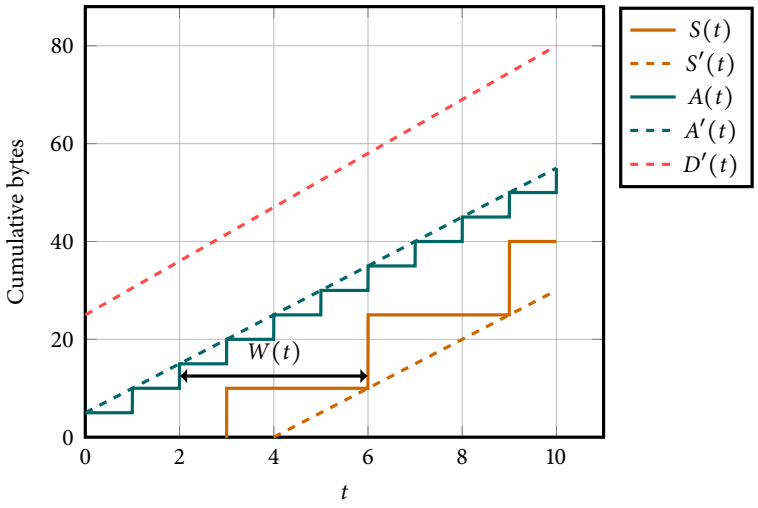


Figure 3.1: Graphical representation of the waiting time bound in deterministic network calculus.

from which we can form the bound

$$\begin{aligned} W(t) &= \min\left\{w \geq 0 : A(t) - \min_{\tau\in[0,t]}\{A(\tau) + S(\tau, t+w)\} \leq 0\right\} \\ &= \min\left\{w \geq 0 : A(t) + \max_{\tau\in[0,t]}\{-A(\tau) - S(\tau, t+w)\} \leq 0\right\} \\ &= \min\left\{w \geq 0 : \max_{\tau\in[0,t]}\{A(\tau, t) - S(\tau, t+w)\} \leq 0\right\}. \end{aligned} \tag{3.13}$$

For affine bounded arrival and service curves, the waiting time reduces to [43]

$$W(t) \leq \frac{\rho_S + \rho_A}{\sigma_S}. \tag{3.14}$$

The waiting time bound is illustrated in Figure 3.1 where arrivals and service are periodic. Concretely, $A(t) = 5\lfloor t \rfloor + 5$ and $S(t) = 15\lfloor t/3 \rfloor - 5$. $A'(t)$ and $S'(t)$ denote the affine bounds which may be used to simplify the analysis. $D'(t)$ denotes the corresponding departure bound of the server. Notice that the waiting time bound are higher when the affine bounded functions are used.

If two affine bounded arrival curves $A_1(t) = [\sigma_{A_1} t + \rho_{A_1}]_+$ and $A_2(t) = [\sigma_{A_2} t + \rho_{A_2}]_+$ arrive to the same affine bounded server $S(t)$, but $A_t(t)$ is prioritized higher than $A_2(t)$, then the service left for $A_2(t)$ is given as [43]

$$S_{lo}(t) \geq [(\sigma_S - \sigma_{A_1})t - \rho_S - \rho_A]_+, \tag{3.15}$$

while $A_1(t)$ is served by the entire service given by $S(t)$.

3.2 Stochastic Network Calculus

This section extends the deterministic network calculus described in previous section to include stochastic processes. In stochastic network calculus, the bounds are probabilistic. That is, they provide bounds on the percentiles of the distributions involved,

such as waiting times. It first introduces stochastic arrival and service curves, and then describe how waiting time bounds can be obtained.

3.2.1 Stochastic Arrival Curves

This section extends the deterministic arrival curves described in previous section to include stochastic processes. A natural way to extend the deterministic affine arrival curves in (3.1) is to provide a probabilistic model of the form

$$\Pr\left(A(\tau,t) > \rho_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right) \leq \epsilon_{\mathrm{A}}, \tag{3.16}$$

where $\rho_{\mathrm{A}} > 0$. For the purpose of studying stochastic network calculus, we consider the class of curves which have Exponentially Bounded Burstiness (EBB):

$$\Pr\left(A(\tau,t) > \rho_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right) \leq \alpha_{\mathrm{A}} e^{-\theta b_{\mathrm{A}}}, \tag{3.17}$$

where $\alpha_{\mathrm{A}} \geq 0$ and $\theta \geq 0$. This class of processes includes Poisson processes and Markovian on/off processes.

It is often more convenient to describe the distributions as their Moment Generating Functions (MGFs) due to the property that the MGF of a linear combination of independent random variables $X = \sum_i a_i X_i$ is given by the product of their moment generating functions $M_X(\theta) = \prod_i M_{X_i}(a_i\theta)$. The MGF of a random variable Y is defined as

$$M_Y(\theta) = E\left[e^{\theta Y}\right], \tag{3.18}$$

where $\theta \geq 0$. Similar to the affine arrival curve bound in (3.17), we define the class of processes for which their MGFs are bounded by an exponential affine function

$$E\left[e^{\theta A(\tau,t)}\right] \leq e^{\theta(\rho_{\mathrm{A}}(t-\tau)+\sigma_{\mathrm{A}})}, \tag{3.19}$$

where $\rho_{\mathrm{A}} > 0$ and $\sigma_{\mathrm{A}} \geq 0$ are functions of θ. It turns out that processes in this class also belong to EBB. This can be shown using the Chernoff bound:

$$\Pr\left(X > x\right) \leq e^{-\theta x} E\left[e^{\theta X}\right]. \tag{3.20}$$

Applying the Chernoff bound to $\Pr\left(A(\tau,t) > \rho_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right)$ one obtains

$$\begin{aligned}\Pr\left(A(\tau,t) > \rho_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right) &\leq e^{-\theta(\rho_{\mathrm{A}}(t-\tau)+b_{\mathrm{A}})} E\left[e^{\theta A(\tau,t)}\right] \\ &\leq e^{-\theta(\rho_{\mathrm{A}}(t-\tau)+b_{\mathrm{A}})} e^{\theta(\rho_{\mathrm{A}}(t-\tau)+\sigma_{\mathrm{A}})} \\ &= e^{\theta\sigma_{\mathrm{A}}} e^{-\theta b_{\mathrm{A}}}.\end{aligned} \tag{3.21}$$

Hence, processes satisfying (3.19) are EBB with parameters $\alpha_{\mathrm{A}} = e^{\theta\sigma_{\mathrm{A}}}$.

Unfortunately, the form (3.17) cannot be used to obtain bounds on the backlog using (3.10), since the τ that maximizes the expression is a random variable. Therefore, we use the following expression instead of (3.17):

$$\Pr\left(\exists A(\tau,t) > \rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right) \leq \alpha'_{\mathrm{A}} e^{-\theta b_{\mathrm{A}}}, \tag{3.22}$$

where the introduction of ρ'_{A} instead of ρ_{A} becomes clear shortly. To find an expression

for α'_{A} we first use the union bound to get

$$\Pr\left(\exists A(\tau,t) > \rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right) \leq \sum_{\tau=0}^{t} \Pr\left(A(\tau,t) > \rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right). \tag{3.23}$$

By defining $\rho'_{\mathrm{A}} = \rho_{\mathrm{A}} + \delta$ with $\delta > 0$ (which is still a valid bound) we have $\rho'_{\mathrm{A}}(t-\tau)+b = \rho_{\mathrm{A}}(t-\tau) + \delta(t-\tau) + b_{\mathrm{A}}$. By using (3.21) we get

$$\begin{aligned}\sum_{\tau=0}^{t} \Pr\left(A(\tau,t) > \rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}}\right) &\leq \sum_{\tau=0}^{t} e^{\theta\sigma_{\mathrm{A}}} e^{-\theta\delta(t-\tau)} e^{-\theta b_{\mathrm{A}}} \\ &= e^{\theta\sigma_{\mathrm{A}}} e^{-\theta b_{\mathrm{A}}} \sum_{\tau=0}^{t} e^{-\theta\delta(t-\tau)} \\ &\leq e^{\theta\sigma_{\mathrm{A}}} e^{-\theta b_{\mathrm{A}}} \sum_{\tau=0}^{t} e^{-\theta\delta t}. \end{aligned} \tag{3.24}$$

Letting $t \to \infty$ to obtain a steady-state bound the sum becomes a converging geometric series:

$$\begin{aligned} e^{\theta\sigma_{\mathrm{A}}} e^{-\theta b_{\mathrm{A}}} \sum_{\tau=0}^{t} e^{-\theta\delta t} &\leq e^{\theta\sigma_{\mathrm{A}}} e^{-\theta b_{\mathrm{A}}} \sum_{\tau=0}^{\infty} e^{-\theta\delta t} \\ &= \frac{e^{\theta\sigma_{\mathrm{A}}} e^{-\theta b_{\mathrm{A}}}}{1 - e^{-\theta\delta}}. \end{aligned} \tag{3.25}$$

Hence α'_{A} is

$$\alpha'_{\mathrm{A}} = \frac{e^{\theta\sigma_{\mathrm{A}}}}{1 - e^{-\theta\delta}}. \tag{3.26}$$

For convenience we denote the bound $\epsilon'_{\mathrm{A}} = \alpha'_{\mathrm{A}} e^{-\theta b_{\mathrm{A}}}$, i.e.

$$\epsilon'_{\mathrm{A}} = \frac{e^{\theta\sigma_{\mathrm{A}}}}{1 - e^{-\theta\delta}} e^{-\theta b_{\mathrm{A}}}. \tag{3.27}$$

3.2.2 Stochastic Service Curves

We may as well define Exponentially Bounded Fluctuations (EBF) service curves which follow the same idea as EBB:

$$\Pr\left(S(\tau,t) < \rho_{\mathrm{S}}(t-\tau) - b_{\mathrm{S}}\right) \leq \alpha_{\mathrm{S}} e^{-\theta b_{\mathrm{S}}}, \tag{3.28}$$

and similar define the MGF

$$E\left[e^{-\theta S(\tau,t)}\right] \leq e^{-\theta(\rho_{\mathrm{S}}(t-\tau)-\sigma_{\mathrm{S}})}. \tag{3.29}$$

From the Chernoff bound this is bounded by

$$\begin{aligned}\Pr\left(S(\tau,t) < \rho_{\mathrm{S}}(t-\tau) - b_{\mathrm{S}}\right) &\leq e^{\theta(\rho_{\mathrm{S}}(t-\tau)-b_{\mathrm{S}})} E\left[e^{-\theta S(\tau,t)}\right] \\ &\leq e^{\theta\rho_{\mathrm{S}}t - \theta\rho_{\mathrm{S}}\tau - \theta b_{\mathrm{S}}} e^{-\theta(\rho_{\mathrm{S}}(t-\tau)-\sigma_{\mathrm{S}})} \\ &= e^{\theta\rho_{\mathrm{S}}t - \theta\rho_{\mathrm{S}}\tau - \theta b_{\mathrm{S}} - \theta\rho_{\mathrm{S}}t + \theta\rho_{\mathrm{S}}\tau + \theta\sigma_{\mathrm{S}}} \\ &= e^{\theta\sigma_{\mathrm{S}}} e^{-\theta b_{\mathrm{S}}}. \end{aligned} \tag{3.30}$$

Following the same procedure as for arrival curves, but letting $\rho'_S = \rho_S - \delta$, $\delta > 0$ one obtains

$$\Pr\left(\exists S(\tau,t) < \rho'_S(t-\tau) - b_S\right) \leq \frac{e^{\theta\sigma_S} e^{-\theta b_S}}{1 - e^{-\theta\delta}}, \tag{3.31}$$

and hence

$$\alpha'_S = \frac{e^{\theta\sigma_S}}{1 - e^{-\theta\delta}}. \tag{3.32}$$

Finally, we let

$$\epsilon'_S = \frac{e^{\theta\sigma_S}}{1 - e^{-\theta\delta}} e^{-\theta b_S}. \tag{3.33}$$

The stability criterion of the system can be derived by the definitions of arrival and service processes. In the deterministic case, the system is stable when

$$\lim_{t\to\infty} \frac{A(t)}{S(t)} < 1. \tag{3.34}$$

In the stochastic case the criterion is equivalent but with the use of expectation

$$\lim_{t\to\infty} \frac{E[A(t)]}{E[S(t)]} < 1. \tag{3.35}$$

For EBB and EBF processes the expected value is obtained by setting $\theta = 1$ in the MGFs:

$$\begin{aligned} 1 &> \lim_{t\to\infty} \frac{\rho'_A t + b_A}{\rho'_S t + b_S} \\ &= \frac{\rho'_A}{\rho'_S}. \end{aligned} \tag{3.36}$$

Using $\rho'_A = \rho_A + \delta$ and $\rho'_S = \rho_S - \delta$ we further have

$$\begin{aligned} \rho_A + \delta &< \rho_S - \delta \\ \delta &< \frac{\rho_S - \rho_A}{2}, \end{aligned} \tag{3.37}$$

where $\delta > 0$, $\rho_S > 0$, and $\rho_A > 0$.

3.2.3 Backlog and Waiting Times

This section describes how to obtain bounds on the backlog and waiting time. For simplicity, only single server models are considered since any network can be reduced to a single server. This property is further described in Section 3.2.8. Consider arrival and service curve samples which satisfy the inequalities

$$A(\tau,t) \leq \rho'_A(t-\tau) + b_A, \tag{3.38a}$$

$$S(\tau,t) \geq \rho'_S(t-\tau) - b_S, \tag{3.38b}$$

where $t > 0$. The backlog follows from (3.10) as

$$\begin{aligned} B(t) &\leq \max_{\tau\in[0,t]} \{A(\tau, t - S(\tau, t)\} \\ &\leq \max_{\tau\in[0,t]} \left\{\rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}} - [\rho'_{\mathrm{S}}(t-\tau) - b_{\mathrm{S}}]_+\right\}. \end{aligned} \tag{3.39}$$

To ease the notation we define

$$b = \max_{\tau\in[0,t]} \left\{\rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}} - [\rho'_{\mathrm{S}}(t-\tau) - b_{\mathrm{S}}]_+\right\}. \tag{3.40}$$

b is finite in a stable system where $A(\tau, t) < S(\tau, t)$. In this case, b attains its maximum value at the maximum τ for which $\rho'(t-\tau) - b_{\mathrm{S}} \leq 0$. It follows that this is exactly when $\rho'_{\mathrm{S}}(t-\tau) - b_{\mathrm{S}} = 0$, i.e. $\tau = t - b_{\mathrm{S}}/\rho'_{\mathrm{S}}$. By substituting into (3.40) we obtain

$$\begin{aligned} b &= \rho'_{\mathrm{A}}\left(t - t + \frac{b_{\mathrm{S}}}{\rho'_{\mathrm{S}}}\right) + b_{\mathrm{A}} \\ &= b_{\mathrm{A}} + b_{\mathrm{S}}\frac{\rho'_{\mathrm{A}}}{\rho'_{\mathrm{S}}} \\ &= b_{\mathrm{A}} + b_{\mathrm{S}}\frac{\rho_{\mathrm{A}} + \delta}{\rho_{\mathrm{S}} - \delta}. \end{aligned} \tag{3.41}$$

Recall that in a stochastic setting, the conditions (3.38a) and (3.38b) may fail with probability bounded by ϵ'_{A} and ϵ'_{S}, respectively. By the union bound the backlog tail is bounded by

$$\Pr\left(B(t) > b\right) \leq \epsilon'_{\mathrm{A}} + \epsilon'_{\mathrm{S}}. \tag{3.42}$$

One may obtain bounds in the waiting time in a similar way. Recall that in the deterministic case the waiting time is defined as

$$W(t) = \min\left\{w \geq 0 : \max_{\tau\in[0,t]} \{A(\tau, t) - S(\tau, t+w)\} \leq 0\right\}. \tag{3.43}$$

Using conditions (3.38a) and (3.38b) we have

$$A(\tau, t) - S(\tau, t+w) \leq \rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}} - [\rho'_{\mathrm{S}}(t+w-\tau) - b_{\mathrm{S}}]_+, \tag{3.44}$$

which attains its minimum when $\tau = t$. This yields

$$\rho'_{\mathrm{A}}(t-\tau) + b_{\mathrm{A}} - [\rho'_{\mathrm{S}}(t+w-\tau) - b_{\mathrm{S}}]_+\Big|_{\tau=t} = b_{\mathrm{A}} - [\rho'_{\mathrm{S}}w - b_{\mathrm{S}}]_+. \tag{3.45}$$

Since b_{A} is non-negative, the values of $w \geq 0$ which satisfy that $b_{\mathrm{A}} - [\rho'_{\mathrm{S}}w - b_{\mathrm{S}}]_+ \leq 0$ are given by $\rho'_{\mathrm{S}}w - b_{\mathrm{S}} \leq b_{\mathrm{A}}$. It follows that

$$\begin{aligned} w &\leq \frac{b_{\mathrm{A}} + b_{\mathrm{S}}}{\rho'_{\mathrm{S}}} \\ &= \frac{b_{\mathrm{A}} + b_{\mathrm{S}}}{\rho_{\mathrm{S}} - \delta}. \end{aligned} \tag{3.46}$$

Following the same argument as for the backlog bound the tail bound on the waiting

time is

$$\Pr\left(W(t) > \frac{b_A + b_S}{\rho_S - \delta}\right) \leq \epsilon'_A + \epsilon'_S. \tag{3.47}$$

In some cases one may have a fixed $\epsilon' = \epsilon'_A + \epsilon'_S$ and seek the waiting time w which can be guaranteed with probability at least ϵ'. Letting $\epsilon'_A = \epsilon'_S = \epsilon'/2$ one obtains (considering ϵ_A)

$$\begin{aligned}
&\frac{\epsilon'}{2} = \frac{e^{\theta\sigma_A}}{1 - e^{-\theta\delta}} e^{-\theta b_A} \\
\Updownarrow \\
&\ln\left(\frac{\epsilon'}{2}\right) = \theta\sigma_A - \ln\left(1 - e^{-\theta\delta}\right) - \theta b_A \\
\Updownarrow \\
&\theta b_A = \theta\sigma_A - \ln\left(\frac{\epsilon'}{2}\right) - \ln\left(1 - e^{-\theta\delta}\right) \\
\Updownarrow \\
&b_A = \sigma_A - \frac{1}{\theta}\left(\ln\left(\frac{\epsilon'}{2}\right) + \ln\left(1 - e^{-\theta\delta}\right)\right).
\end{aligned} \tag{3.48}$$

The same procedure can be used for ϵ_S to obtain

$$b_S = \sigma_S - \frac{1}{\theta}\left(\ln\left(\frac{\epsilon'}{2}\right) + \ln\left(1 - e^{-\theta\delta}\right)\right). \tag{3.49}$$

3.2.4 Constant Rate Server

In reality, many servers provide a deterministic, constant service rate. These include for instance switched Ethernet where the service rate represents the serialization process. The constant rate server needs to be treated explicitly, since it has $\epsilon_S = 0$.

Let c denote the rate of the server and d the time offset until the service starts. This gives

$$S(\tau, t) = c(t - \tau) - d, \tag{3.50}$$

i.e. $\rho'_S = c$ and $b_S = d$. From (3.41) and (3.46) it follows that

$$b = b_A + d\frac{\rho'_A}{c}, \tag{3.51}$$

$$w = \frac{b_A + d}{c}. \tag{3.52}$$

Following the approach in [42] and setting $\delta = c - \rho_A$ and using that $\epsilon' = \epsilon_A$ one obtains

$$b_A = \sigma_A - \frac{1}{\theta}\left(\ln\left(\epsilon'\right) + \ln\left(1 - e^{-\theta(c-\rho_A)}\right)\right). \tag{3.53}$$

3.2.5 Arrival Processes

Several arrival processes in the EBF class can be derived. In this section we consider two widely used processes: Poisson processes and Markovian On/Off processes.

Arrivals with exponentially distributed inter-arrival times result in an arrival pro-

cess $N(t)$ which is Poisson distributed with mean inter-arrival time λ^{-1}

$$N(t) \sim \text{Poisson}(\lambda t). \tag{3.54}$$

The MGF for a Poisson random variable is [45]

$$M_{N(t)}(\theta, t) = e^{\lambda t(e^{\theta}-1)}. \tag{3.55}$$

Assuming constant packet size s we have the arrival curve

$$A(t) = sN(t) \tag{3.56}$$

with MGF

$$M_{\text{A}}(\theta, t) = M_{N(t)}(s\theta, t). \tag{3.57}$$

It follows that

$$\rho_{\text{A}} = \frac{\lambda(e^{\theta s}-1)}{\theta}, \tag{3.58}$$

$$\sigma_{\text{A}} = 0, \tag{3.59}$$

for $\theta > 0$.

For packet sizes which follow an exponential distribution with mean s the arrival process is

$$A(t) = \sum_{k=1}^{N(t)} X_k, \tag{3.60}$$

where $X_k \sim \exp(s^{-1})$. The MGF for X_k is [42]

$$M_X(\theta) = \frac{1}{1-\theta s}. \tag{3.61}$$

Since $A(t)$ is a sum of random variables the MGF of $A(t)$ is the product of the MGFs of X_k,

$$\begin{aligned} M_A(\theta, t) &= E\left[M_X(\theta)^{N(t)}\right] \\ &= E\left[e^{\ln\left(M_X(\theta)^{N(t)}\right)}\right] \\ &= E\left[e^{\ln(M_X(\theta))N(t)}\right]. \end{aligned} \tag{3.62}$$

By the definition of a MGF we have that $M_{\text{A}}(\theta, t) = M_{N(t)}\left(\ln\left(M_X\left(\theta\right)\right), t\right)$. Inserting into (3.55) gives

$$\begin{aligned} M_A(\theta, t) &= e^{\lambda t\left(e^{\ln\left(M_X(\theta)\right)}-1\right)} \\ &= e^{\lambda t(M_X(\theta)-1)} \\ &= e^{\lambda t\left(\frac{1}{1-\theta s}-1\right)} \\ &= e^{\frac{\theta s \lambda t}{1-\theta s}}. \end{aligned} \tag{3.63}$$

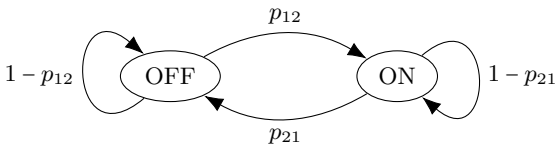


Figure 3.2: Markovian On/Off process.

It follows that

$$\rho_A = \frac{s\lambda}{1 - \theta s}, \tag{3.64}$$

$$\sigma_{\mathrm{A}} = 0, \tag{3.65}$$

for $0 \geq \theta < 1/s$.

Although Poisson traffic are often used for analysis of networks, many applications tend to send data in bursts. One traffic model which captures bursty arrivals is the (discrete-time) Markovian On/Off process depicted in Figure 3.2. A Markovian On/Off process contains an On and an Off state. In the On state (state 1), the process generates traffic with a constant rate r, while no traffic is generated in the Off state (state 2). Transitions from the On state to the Off state, and from the Off to the On state, occur with probability p_{12} and p_{21}, respectively. The mean arrival rate is given by $p_{\mathrm{on}} r$ where

$$p_{\mathrm{on}} = \frac{p_{12}}{p_{12} + p_{21}}. \tag{3.66}$$

Processes with the same mean arrival rate can have various degrees of burstiness. The burstiness can be characterized by the quantity $T = p_{12}^{-1} + p_{21}^{-1}$ which is low when the system often changes state and high when the time between state transitions is high. Realizations of Markov On/Off processes with equal mean rates and various values of T are shown in Figure 3.3.

The Markovian On/Off process belongs to the class of EBB arrival curves with parameters $\sigma_{\mathrm{A}} = 0$ and

$$\rho_{\mathrm{A}} = \frac{1}{\theta} \ln \left(\frac{p_{11} + p_{22} e^{\theta r} + \sqrt{\left(p_{11} + p_{22} e^{\theta r}\right)^2 - 4\left(p_{11} + p_{22} - 1\right) e^{\theta r}}}{2} \right), \tag{3.67}$$

where $p_{11} = 1 - p_{12}$ and $p_{22} = 1 - p_{21}$ [42].

3.2.6 Aggregate Flows

To simplify the analysis of a network, it is often desirable to combine multiple flows into a single flow. Stochastic network calculus provides a way to aggregate flows even when the flows have different arrival curves. Consider the aggregate arrival curve of M flows given by

$$A_{\mathrm{agg}}(\tau, t) = \sum_{i=1}^{M} A_i(\tau, t) \tag{3.68}$$

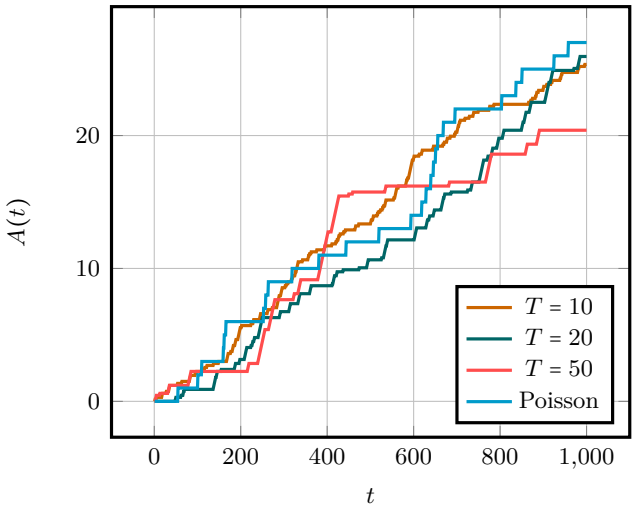


Figure 3.3: Realization of Markovian On/Off processes with fixed rate $r = 0.15$ and $p_{\text{on}} = 1/600$ for various burstiness parameters T.

which has moment generating function

$$E\left[e^{\theta A_{\text{agg}}(\tau,t)}\right] = E\left[e^{\theta\left(\sum_{i=1}^{M} A_i(\tau,t)\right)}\right] = E\left[\prod_{i=1}^{M} e^{\theta A_i(\tau,t)}\right]. \tag{3.69}$$

Assuming independence between the flows and using the affine MGF bound we have

$$\begin{aligned} E\left[e^{\theta A_{\text{agg}}(\tau,t)}\right] &= \prod_{i=1}^{M} E\left[e^{\theta A_i(\tau,t)}\right] \\ &\leq \prod_{i=1}^{M} e^{\theta(\rho_{\text{A}_i}(t-\tau)+\sigma_{\text{A}_i})} \\ &= e^{\theta\left(\sum_{i=1}^{M}(\rho_{\text{A}_i}(t-\tau)+\sigma_{\text{A}_i})\right)}. \end{aligned} \tag{3.70}$$

It follows that $A_{\text{agg}}(\tau, t)$ also belongs to the group of EBB arrivals with parameters

$$\rho_{\text{agg}} = \sum_{i=1}^{M} \rho_{\text{A}_i}, \tag{3.71a}$$

$$\sigma_{\text{agg}} = \sum_{i=1}^{M} \sigma_{\text{A}_i}. \tag{3.71b}$$

3.2.7 Scheduling

Multiple flows arriving to the same server are scheduled according to some discipline, e.g. first-come-first-served, round-robin, or a priority scheduling. In stochastic network calculus, a worst-case scheduling discipline is considered by assuming that all other flows are scheduled before the flow of interest. We consider two flows arriving to the server: cross flow and through flow. We are interested in obtaining the backlog and waiting time experienced by the through flow. In case more than two flows arrive

to the server we obtain the cross flow as the aggregate flow of cross traffic. Let $A_{\text{cr}}(\tau, t)$ denote the cross flow and $A_{\text{th}}(\tau, t)$ denote the through flow so that the cumulative arrivals are given by $A(t) = A_{\text{cr}}(\tau, t) + A_{\text{th}}(\tau, t)$ and the cumulative departures are $D(t) = D_{\text{cr}}(\tau, t) + D_{\text{th}}(\tau, t)$.

Let τ^* denote the beginning of the last busy period and consider a $t \geq \tau^*$. During the time from τ^* to t, at most $S(\tau^*, t)$ bytes can depart from the server. Furthermore, since the queue is empty at time τ^* the number of departured bytes is equal to the number of arrivals, i.e. $D(\tau^*) = A(\tau^*)$. It follows that

$$\begin{aligned} & D(t) = A(\tau^*) + S(\tau^*, t) \\ \Updownarrow \quad & \\ & D_{\text{cr}}(t) + D_{\text{th}}(t) = A_{\text{cr}}(\tau^*) + A_{\text{th}}(\tau) + S(\tau^*, t) \\ \Updownarrow \quad & \\ & D_{\text{th}}(t) = A_{\text{th}}(\tau^*) + S(\tau^*, t) - (D_{\text{cr}}(t) - A_{\text{cr}}(\tau^*)). \end{aligned} \tag{3.72}$$

Since $D_{\text{cr}}(t) \leq A_{\text{cr}}(t)$ we have the lower bound

$$\begin{aligned} & D_{\text{th}}(t) \geq A_{\text{th}}(\tau^*) + S(\tau^*, t) - (A_{\text{cr}}(t) - A_{\text{cr}}(\tau^*)) \\ \Updownarrow \quad & \\ & D_{\text{th}}(t) \geq A_{\text{th}}(\tau^*) + [S(\tau^*, t) - A_{\text{cr}}(\tau^*, t)]_+ \\ \Updownarrow \quad & \\ & D_{\text{th}}(t) \geq \min_{\tau \in [0,t]} \{A_{\text{th}}(\tau) + [S(\tau, t) - A_{\text{cr}}(\tau, t)]_+\}. \end{aligned} \tag{3.73}$$

Notice that this expression does not depend on how $A_{\text{cr}}(\tau, t)$ and $A_{\text{th}}(\tau, t)$ are scheduled, but rather provides a minimum of the number of departures.

From (3.5) it follows that the quantity $[S(\tau, t) - A_{\text{cr}}(\tau, t)]_+$ is a service curve. Define the quantity as the leftover service for $A_{\text{th}}(t)$ denoted $S_{\text{lo}} = [S(\tau, t) - A_{\text{cr}}(\tau, t)]_+$. We then have

$$\begin{aligned} E\left[e^{-\theta S_{\text{lo}}(\tau,t)}\right] &= E\left[e^{-\theta[S(\tau,t)-A_{\text{cr}}(\tau,t)]_+}\right] \\ &\leq E\left[e^{-\theta S(\tau,t)+\theta A_{\text{cr}}(\tau,t)}\right]. \end{aligned} \tag{3.74}$$

Assuming independence between $S(\tau, t)$ and $A_{\text{cr}}(\tau, t)$ we have $E\left[e^{-\theta S(\tau,t)+\theta A_{\text{cr}}(\tau,t)}\right] = E\left[e^{-\theta S(\tau,t)}\right] E\left[e^{\theta A_{\text{cr}}(\tau,t)}\right]$. For affine server MGF envelope with ρ_{S} and σ_{S} the leftover service also has affine MGF envelope given by

$$\begin{aligned} E\left[e^{-\theta S_{\text{lo}}(\tau,t)}\right] &\leq e^{-\theta(\rho_{\text{S}}(t-\tau)-\sigma_{\text{S}})} e^{\theta(\rho_{\text{A}_{\text{cr}}}(t-\tau)+\sigma_{\text{A}_{\text{cr}}})} \\ &= e^{-\theta((\rho_{\text{S}}-\rho_{\text{A}_{\text{cr}}})(t-\tau)-(\sigma_{\text{S}}+\sigma_{\text{A}_{\text{cr}}}))}. \end{aligned} \tag{3.75}$$

That is, the leftover service $S_{\text{lo}}(t)$ has affine MGF envelope parameterized by

$$\rho_{\text{lo}} = \rho_{\text{S}} - \rho_{\text{A}_{\text{cr}}}, \tag{3.76a}$$

$$\sigma_{\text{lo}} = \sigma_{\text{S}} + \sigma_{\text{A}_{\text{cr}}}. \tag{3.76b}$$

Note that if $S(t)$ is a constant rate server and A_{cr} and A_{th} are independent, then $S_{\text{lo}}(t)$

also has $\epsilon' = 0$ [46] with waiting time bound given by

$$b = \frac{\sigma_A + \sigma_{lo} - \frac{1}{\theta}\left(\ln\left(\epsilon'\right) + \ln\left(1 - e^{-\theta(\rho_{lo}-\rho_A)}\right)\right)}{\rho_{lo}}. \tag{3.77}$$

3.2.8 Equivalent Servers

A network of multiple servers can be reduced to a single server which is equivalent to the entire network from the view of a certain flow. This allows the above derived methods to be used for analyzing entire networks and not only single server systems.

From deterministic network calculus we have that a system of two servers $S_1(t)$ and $S_2(t)$ may be represented as $S(\tau, t) = S_1 \otimes S_2(\tau, t)$. It follows that the MGF of $S(t)$ is given by

$$\begin{aligned} E\left[e^{-\theta S(\tau,t)}\right] &= E\left[e^{-\theta(S_1 \otimes S_2)(\tau,t)}\right] \\ &= E\left[e^{-\theta(\min_{v\in[\tau,t]}\{S_1(\tau,v)+S_2(v,t)\})}\right]. \end{aligned} \tag{3.78}$$

By replacing $\min_{v\in[\tau,t]}\{\cdot\}$ with the sum $\sum_{v\in[\tau,t]}\{\cdot\}$ and assuming independence between the servers we obtain an upper bound

$$E\left[e^{-\theta S(\tau,t)}\right] \leq \sum_{v=\tau}^{t} E\left[e^{-\theta S_1(\tau,v)}\right] E\left[e^{-\theta S_2(v,t)}\right]. \tag{3.79}$$

Assuming that the service processes are stationary and hence only depends on the time difference, (3.79) is equivalent to the convolution between the MGFs

$$\begin{aligned} E\left[e^{-\theta S(\tau,t)}\right] &\leq \sum_{v=0}^{t-\tau} E\left[e^{-\theta S_1(v)}\right] E\left[e^{-\theta S_2(t-\tau-v)}\right] \\ &= \sum_{v=0}^{t-\tau} M_{S_1}(-\theta, v) M_{S_2}(-\theta, t-\tau-v) \\ &= M_{S_1} * M_{S_2}(-\theta, t-\tau), \end{aligned} \tag{3.80}$$

where we have used the shorthand notation $M_S(\theta, t) = E\left[e^{\theta S(t)}\right]$. It follows by recursion that the MGF of n tandem servers is

$$M_S(-\theta, t) \leq M_{S_1} * M_{S_2} \ldots M_{S_3}(-\theta, t). \tag{3.81}$$

For services which obey affine MGF envelopes is can be shown that the tandem servers can be reduced to a single server with $\rho_S = \min_i \rho_{S_i}$ and $\sigma_S = \sum_i \sigma_{S_i}$ [42]. This result is consistent the common intuition about bottlenecks in computer networks.

3.3 Uses and Limitations of Network Calculus

Network calculus provides a framework for analyzing waiting times in networks. Deterministic network calculus is intuitive to use, but is limited to deterministic arrival and service processes. Furthermore, it is based on worst-case assumptions which may in many cases be too strict.

Stochastic network calculus extends deterministic network calculus to include stochas-

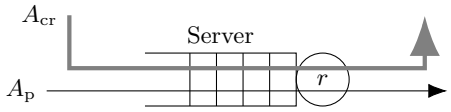


Figure 3.4: Example queue consisting of a single server, 10 Markovian On/Off cross flows (A_{cr}), and 1 Poisson flow of interest (A_p).

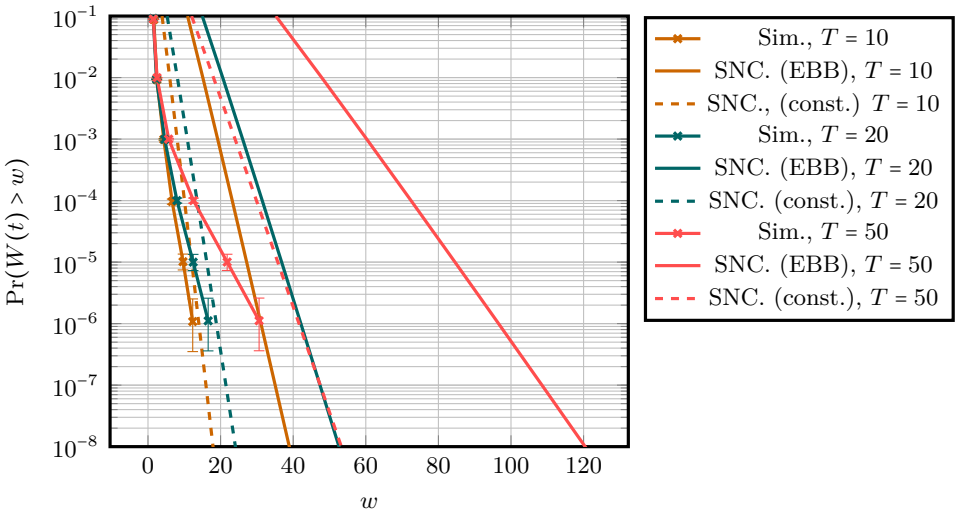


Figure 3.5: Waiting time bounds and simulation in a system with Poisson arrivals and Markov On/Off cross traffic.

tic arrival and/or service processes. A natural question that arises is how good the bounds obtained by the framework are, and how the framework compares to other frameworks such as queuing theory and deterministic network calculus. To answer this, we consider two examples. First, we consider a network equivalent to the one considered in [42, Section III A] which illustrates the usefulness of stochastic network calculus. We consider a queue depicted in Figure 3.4 consisting of a single constant rate server with rate $c = 1$. We are interested in the waiting time percentile defined by $\Pr(W(t) > w) < \epsilon$ experienced by a Poisson flow A_p with $\lambda = 0.25$ and packets of size 1. Besides the Poisson flow, 10 independent Markov On/Off flows are also sharing the server (indicated as a single flow A_{cr}). Each On/Off flow sends packets of size $r = 0.15$ in the On state and has a mean rate of $p_{on}r = 0.025$. The free parameters θ and δ are optimized using Ipopt [47] to provide the best bound.

The bounds are calculated by first calculating an aggregate flow of the On/Off flows, and then calculating the leftover service from the server. This is then used as the server process for the Poisson flow.

Simulation results and network calculus bounds are shown in Figure 3.5 for various burstiness values T of the On/Off flows. The vertical bars on the simulation points are 95% confidence intervals. We obtain bounds using stochastic network calculus by two methods: (1) treating leftover service as a regular EBB server which has nonzero violation probability ϵ_{lo}, and (2) by using the result from [46] shown in (3.77) stating that $\epsilon_{lo} = 0$. The former method is equivalent to the method used in [42] and provides equivalent results. However, the latter method provides much better bounds on the waiting time since it makes use of the fact that the service is deterministic.

It is worth noting that if the above scenario was to be analyzed using queuing the-

ory, it would require a Markov chain with states for each On/Off flow. Furthermore, capturing the fact that the On/Off processes generate a constant rate in the On state requires the use of a discrete Markov chain which leads to a very complex transition matrix. Likewise, analyzing the scenario using deterministic network calculus would lead to high overprovisioning since the arrival processes, in theory, may produce an arbitrary high amount of traffic. Hence, in order to obtain useful results from deterministic network calculus, one would have to introduce traffic shapers to the network which may not be feasible or desired in practice.

However, stochastic network calculus also has its limitations. For instance, it does not handle periodic arrivals very well since such an arrival process does not fit well into the category of affine MGF envelopes. To show this, we consider the same queue as before, but this time letting A_{p} be a periodic arrival process which generates 1 byte with period 1, and A_{cr} be a Poisson arrival process which generates 1 byte with rate $\lambda = 0.5$. Both arrival processes are served by the server with a constant rate $r = 2$. To model the periodic arrival process using stochastic network calculus, we use the fact that the MGF of a source generating r bytes with period τ and uniform phase is given by [48, 49]

$$e^{\theta\left(r\left\lfloor\frac{t}{\tau}\right\rfloor+\frac{1}{\theta}\ln\left(1+\left(\frac{t}{\tau}-\left\lfloor\frac{t}{\tau}\right\rfloor\right)\left(e^{\theta r}-1\right)\right)\right)}, \tag{3.82}$$

which is upper bounded by the affine MGF

$$e^{\theta\left(\frac{r}{\tau}t+r\right)}, \tag{3.83}$$

i.e. $\rho_{\mathrm{A}} = r/\tau$ and $\sigma = r$. However, the difference between the process described by this bound and the periodic process is that arrivals following this approximation are more likely to arrive to an idle server since they do not arrive in bulks. This reduces the overall waiting time experienced by the arrivals as illustrated in Figure 3.6 which shows the bounds obtained from stochastic network calculus (SNC) along with bounds obtained by simulation (sim). While the affine approximation provides an upper bound for the small percentiles it does not capture the tail of the waiting time distribution well. The reason is that while the affine approximation provides a bound on all moments, it is not a bound on the shape of the distribution. The specific scenario could possibly be better characterized using queuing theory by modelling the system as a discrete-time Markov chain indexed by the periodicity of the arrivals. Furthermore, this would result in an exact representation of the waiting time distribution. However, a queuing theoretic approach would be limited to the case where the cross traffic is Poisson distributed, and to a single queue since the departure distribution may be difficult to characterize. Finally, we could attempt to model the scenario using deterministic network calculus. Since both the periodic arrival process and the server are deterministic, only the Poisson flow causes difficulties similar to those in the previous scenario.

To summarize, deterministic and stochastic network calculus provides a framework for obtaining bounds on waiting times in networks with deterministic of stochastic arrivals and servers. Deterministic network calculus supports any bounded arrival and server processes, but are affine bounded processes simplifies the calculations. Stochastic network calculus supports a wide range of stochastic arrival processes which may not be possible or straightforward to model using e.g. queuing theory or deterministic network calculus. Furthermore, it allows for combining different arrival processes in the same network, which can be particularly difficult in queuing theory. However, it cannot be applied for arrival or service processes which are deterministic and bursty,

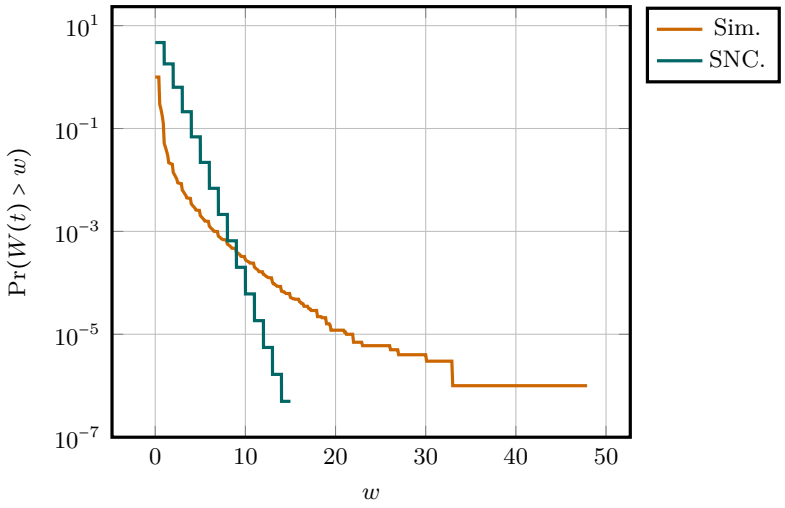


Figure 3.6: Comparison between waiting time bounds obtained from stochastic network calculus using an affine MGF envelope approximation, and a simulated exact system.

such as periodic arrivals. Such processes are better modelled using queuing theory and deterministic network calculus. However, one class of networks which remains difficult to analyze is networks consisting of a mixture of stochastic and deterministic processes, since stochastic processes are difficult to model deterministically, and deterministic processes may be troublesome to model stochastically.

Both deterministic and stochastic network calculus can be used to analyze end-to-end properties of network slices in Industry 4.0. While deterministic network calculus targets worst-case analysis and hence is suitable for applications with very strict requirements, stochastic network calculus allows for relaxing this requirement in scenarios where worst-case analysis is too strict. However, the fact that stochastic network calculus is limited to analysis of stochastic arrival processes constraints the types of networks in which it can be applied. Specifically, since industrial communication technologies often rely on determinism, stochastic network calculus may be inapplicable to analyzing networks where these protocols are predominant. On the other hand, deterministic network calculus may be unsuitable for analyzing cloud networks with many flows and a high degree of multiplexing. Since both types of networks are expected in Industry 4.0, both frameworks are likely to be useful in different parts of the network.

4 Industrial Communication Technologies

Industry 4.0 deployments are likely to consist of both industrial communication technologies and packet switched networks. Since the operating principles of the communication technologies are important in order to analyze end-to-end network slice properties, understanding the characteristics of industrial networks is required. Furthermore, due to the network complexity in Industry 4.0 and the conservatism of the manufacturing industry, it is likely that the transition to a complete Industry 4.0 deployment will take several decades. Therefore, the protocols that are currently in use in industrial networks are likely to be part of an Industry 4.0 network as well, and hence need to be supported in the construction of network slices. This chapter provides an overview of some widely used communication technologies used in industrial systems.

As depicted in Figure 4.1, an industrial communication network typically follows the hierarchical structure [50]. In the top of the hierarchy is the plant level which provides a plant-wide network which may also be connected to an external infrastructure such as the Internet. The plant level typically includes general purpose hardware and cloud computing resources which can be used by the devices at the cell level, or even by components at the device level. The cell level contains control units and devices which control master-slave networks, and the device level includes devices such as actuators, input/output devices, sensors, etc. The communication technologies on the plant level typically consist of high capacity links which provide a best-effort service. However, it may also contain links that allow for strict queuing disciplines and hence provide a certain guarantees. The cell level may contain both deterministic and non-deterministic links. For instance, the communication between controllers may be deterministic while the connectivity to the plant level may be based on switched Ethernet or similar technologies. Finally, the communication between the cell level and the device level is likely to be deterministic since the applications at this level may require very low latency (sub-millisecond) and very high reliability, which is typically realized using pre-allocated cyclic master-slave protocols.

The remainder of this chapter examines the overall operating principles of two widely used industrial communication technologies which mainly operate in the cell and device levels of the hierarchy. Furthermore, it briefly discusses an ongoing work in the field of deterministic and predictable networks which may be applicable at the plant

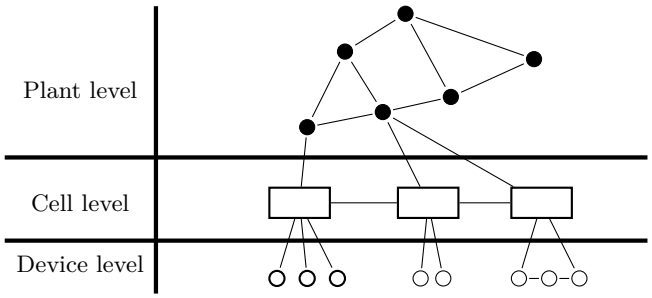


Figure 4.1: Hierarchical architecture of industrial network.

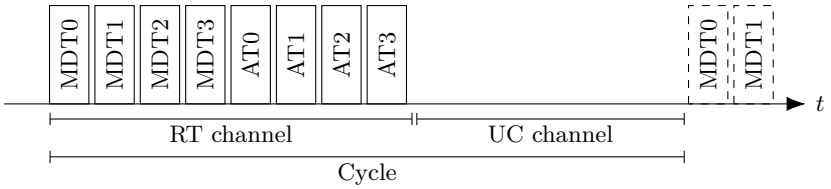


Figure 4.2: Cycle structure of Sercos III.

level. Although numerous industrial communication technologies exist, the considered protocols are based on some common operating principles which are shared by most protocols used for low latency and high reliability industrial networks.

4.1 Sercos III

Sercos III [51] is an 100 Mbit/s Ethernet based industrial communication protocol which is designed for device level applications such as communication between a controller and actuators, etc. following a master-slave operation scheme. Sercos III supports ring topologies, and is typically deployed in a double ring to provide redundancy. The slave devices in the ring use cut-through connectors so that the physical Ethernet layer forms a bus between the slaves and the master.

Communication in Sercos III occurs in cycles which consist of a Real-Time (RT) channel followed by a Unified Communication (UC) channel separated in time (Figure 4.2). The RT channel carries Sercos III telegrams which are formatted as the regular IEEE 802.3 frames with EtherType `0x88CD`. The channel consists of 1–4 Master Data Telegrams (MDTs) which contain data from the master to the slaves, and 1–4 Acknowledge Telegrams (ATs) which contain data from slaves to the master, or between slaves. Both MDTs and ATs are broadcasted by the master, but ATs are filled by the slaves. The MDT and AT Ethernet frames contain a regular Ethernet header and frame check sequence trailer as illustrated in Figure 4.3. The frame payload is a 6 bytes MDT or AT header followed by service channels and real-time data. The service channels take up 6 bytes per slave, and are used for Sercos III specific communication which is not forwarded to the application layer. In case multiple MDT or AT frames are used, the service channels may be split across all frames. Following the service channels are the real-time data divided into connections which follow the publish/subscribe pattern. Each connection occupies 2 bytes for a header followed by a variable number of bytes for application data. Hence, given the Ethernet payload size of 1500 bytes, $N_{\text{conn}}^{\text{MDT}}$ MDT connections and $N_{\text{conn}}^{\text{AT}}$ AT connections, the following inequalities must be satisfied

$$\sum_{n=1}^{N_{\text{conn}}^{\text{MDT}}} 2 + s_n^{\text{MDT}} \leq (1500 - 6) N_{\text{frames}}^{\text{MDT}} - 6N_{\text{slaves}}, \tag{4.1}$$

$$\sum_{n=1}^{N_{\text{conn}}^{\text{AT}}} 2 + s_n^{\text{AT}} \leq (1500 - 6) N_{\text{frames}}^{\text{AT}} - 6N_{\text{slaves}}, \tag{4.2}$$

where s_j^{MDT} and s_j^{AT} are the amount of application data used by connection j (bytes) for MDT and AT, respectively, $N_{\text{frames}}^{\text{MDT}}$ and $N_{\text{frames}}^{\text{AT}}$ are the number of MDT/AT frames, and N_{slaves} is the number of slaves.

The UC channel is used for non-Sercos Ethernet frames, such as regular IP packets,

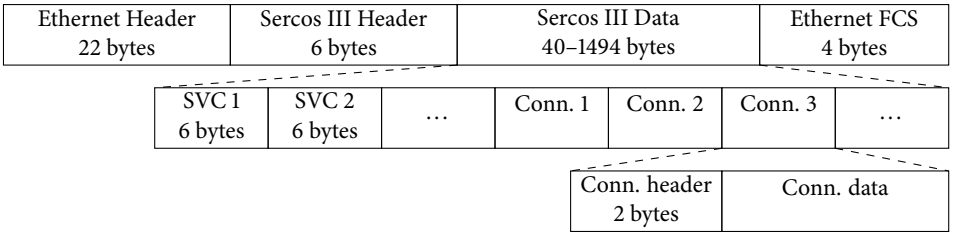


Figure 4.3: Sercos III frame structure.

and may be transmitted by both master and slave devices. The UC channel is handled as best-effort, and packets may be stored by the Sercos devices before being forwarded. The duration of the UC channel is determined by the cycle time and the size of the RT channel. The cycle time can either be 31.25 µs, 62.50 µs, 125 µs, or multiples of 250 µs up to 65 ms.

4.2 EtherCAT

EtherCAT (Ethernet for Control Automation Technology) [52] is, like Sercos III, based on 100 Mbit/s Ethernet and introduces several modifications to regular Ethernet. EtherCAT defines two protocols: EtherCAT Device Protocol (EDP) and EtherCAT Automation Protocol (EAP). EDP is used for master/slave communication and mainly targets hard real-time applications such as closed-loop control systems. EAP both supports cyclic transmissions (soft real-time) and acyclic transmissions, and is used for plant-wide communication, e.g. between controllers and for human machine interfaces. As already mentioned we limit the focus to hard real-time protocols and hence only consider EDP.

The operation of EDP is similar to that of Sercos III. Ethernet frames are generated by the master device while slaves are allowed to fill the payload. Although communication with EDP does not have to be periodic, it is typically used this way [52]. However, contrary to Sercos III, EDP does not allow regular Ethernet packets to be delivered to the slaves. Each Ethernet frame contains one or more datagrams which contain a header that includes addressing, and specifies whether slaves are allowed to read and/or write the datagram. Addressing in EDP can either be direct or logical. In direct addressing, the slave address is specified in the datagram whereas in logical addressing, a piece of data is assigned an address in a logical memory space. By using the abstraction of a logical memory space, the EDP protocol can be considered a distributed memory space where multiple users can to read or write. This way, the master can address multiple slaves in one datagram.

EDP can also be used for acyclic communication using a special mailbox datagram. A mailbox allows for tunneling other protocols through EtherCAT, including CAN and regular Ethernet frames. While the mailbox mechanism allows for acyclic communication, slots for mailboxes must be allocated in the cycle. Packets which are tunneled through mailboxes may be queued and are not guaranteed to be transmitted in the following cycle.

EDP frames consist of the regular Ethernet header, followed by a 2 bytes EtherCAT header and the datagrams (Figure 4.4). Each datagram consists of a datagram header (10 bytes), a configurable, but fixed, number of data bytes and a 2 bytes work-

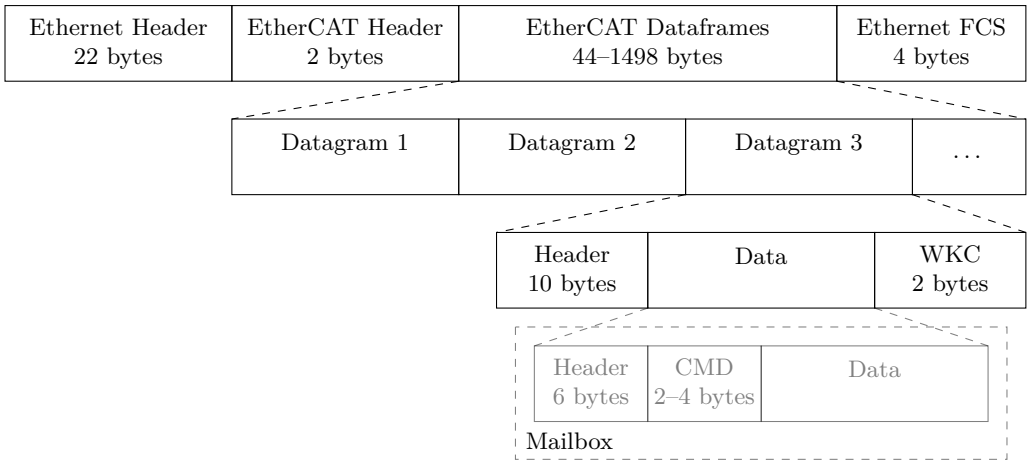


Figure 4.4: EDP frame structure. The mailbox frame is only used when other protocols are tunneled through EtherCAT.

ing counter (used to verify success of read/write operations). In case a mailbox is used, the datagram data field is divided into a 6 bytes mailbox header, 2-4 bytes command field (depending on the type of mailbox), and a variable amount of mailbox data. The cycle time is determined by the master device, which is also allowed to use various periods for various datagrams of traffic. Therefore, two consecutive cycles do not necessarily address the same devices.

Although EDP devices can be connected to form a variety of topologies, including line, ring and star networks, the wiring between physical Ethernet ports in an EtherCAT device causes the devices always to be connected in a line topology. Therefore, slaves can only communicate to slaves located after the transmitting device on the line. If a slave needs to send data to a slave located before the transmitting device, the data must be forwarded by the master device. In both cases, the transmission must be initialized by the master device which has to allocate the datagram in the frame.

4.3 Deterministic and Time-Sensitive Networking (DetNet/TSN)

Due to an increasing demand on very low latency and high reliability in Ethernet based networks, several initiatives have been started to enable this. These technologies are very interesting and relevant as enablers for connecting industrial devices to the cloud and enabling real-time cloud computing. Two notable initiatives in this field are the IETF Deterministic Networking (DetNet) [53] which is a general architecture for deterministic network technologies, and the IEEE 802.1 Time-Sensitive Networking (TSN) [54], which brings several deterministic elements to switched Ethernet.

The IETF DetNet aims at providing assured end-to-end latency by reserving resources such as bandwidth and buffer space for specific flows. Furthermore, DetNet introduces synchronized packet forwarding and static end-to-end routes possibly with redundant paths. While IETF DetNet mainly focuses on the nodes in a network and the overall control plane, the focus of 802.1 TSN is to provide a physical layer protcol for realizing deterministic and/or very low latency communication using Ethernet. 802.1 TSN consists of several standards which are not necessarily all required for enabling

	Sercos III	EtherCAT	DetNet/TSN
Technology	100 Mbit Ethernet	100 Mbit Ethernet	Mega/Gigabit Ethernet
Overall structure	Cyclic master/slave	Cyclic master/slave	Switched
Cyclic structure	Deterministic followed by non-deterministic	Deterministic and non-deterministic in same frame	-
Cycle time	Constant	Variable (controlled by master device)	-
Topology	Line, ring	Line, ring, tree, star (realized as line)	Tree, star
Master/slave method	Master-to-slave in one frame, slave-to-slave/slave-to-master in another frame	Master-to-slave and slave-to-master in same frame	-
Handling of Non-RT	Transmitted like regular Ethernet in allocated space	Tunneled	Transmitted in allocated space

Table 4.1: Comparison of the three industrial communication protocols.

deterministic and low latency communication. Current standards include preemption of Ethernet frames to prevent high-priority packets to be blocked by low-priority packets, as well as time-aware scheduling of queues. Although both EITF DetNet and 802.1 TSN are in the early phases of development, they allow for a great simplification in the analysis of end-to-end guarantees since elements, such as head-of-line blocking, can be safely ignored. Furthermore, it motivates for a priority queuing based scheme in the plant-wide network.

4.4 Comparison

A comparison of the main characteristics of the three communication technologies is given in Table 4.1.

5 Slicing Industrial Networks

This chapter considers the task of slicing industrial networks from a low-layer perspective, namely how the communication resources provided by the industrial protocols can be divided into multiple slices. Although network slices may be constructed from physically isolated media, it is generally desired to share a medium between multiple slices in order to utilize the resources efficiently. However, this introduces multiplexing between network slices, and the multiplexing scheme strongly influences the guarantees that can be provided to the application, as well as the level of isolation between the slices. First, different multiplexing schemes that can be applied on top of cyclic industrial protocols are considered, with focus on Sercos III and EtherCAT presented in the previous chapter. Then, two example scenarios are defined which are used to analyze the impact of different multiplexing schemes on the latency and reliability, and to investigate how end-to-end properties can be obtained.

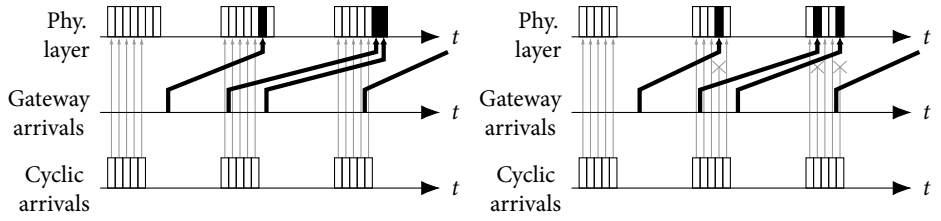
5.1 Multiplexing in Industrial Communication Technologies

A trivial way to allocate network slices over industrial protocols is to statically allocate a number of resources in every cycle for each slice. However, although this provides complete isolation between the slices, it is likely to result in very low utilization of the links since some network slices may not transmit data very often. Furthermore, since the resource allocation typically cannot be reconfigured without restarting the system, allocating resources to each slice significantly limits the scalability of the slices. Instead, one may look into multiplexing the resources between slices. Since resources in industrial protocols typically are not multiplexed, in particularly not for real-time traffic, there is a need for constructing an access layer above the protocol which controls the multiplexing of resources. In this section, various approaches for multiplexing resources in industrial communication protocols are discussed based on the overall structures of Sercos III and EtherCAT, but with more general applications.

Although both Sercos III and EtherCAT already provide resource multiplexing in the UC channel and by the use of mailboxes, these functionalities are not well suited for real-time communication since nodes may store frames for several cycles before forwarding them. Instead, one must use the reserved real-time resources if low latency is a strict requirement, and possibly implement multiplexing schemes on top of these.

One approach is to introduce a slave gateway which has a certain number of allocated resources that can be multiplexed by the gateway (Figure 5.1a). For instance, a wireless transceiver could act as a slave device in the industrial network and at the same time as a gateway for a number of wireless devices. This approach provides high flexibility since the multiplexing decision is moved to the application layer, but introduces a forwarding delay which may violate the strictest requirements. Furthermore, the number of resources which can be allocated may be limited, e.g. as is the case for the number of MDT/AT frames in Sercos III. Depending on the stochastic properties of the arrivals and the application requirements, this scheme may also result in resources being unused in most cycles.

Another approach is to allow multiple slaves to use the same resources. In case a slave writes to a resource which has already been written to, the existing data in the resource is overwritten (replaced) by the new data (Figure 5.1b). This may result



(a) Multiplexing of reserved resources between gateway arrivals.

(b) Multiplexing by overwriting random cyclic traffic.

Figure 5.1: Two methods of cyclic resource multiplexing.

in service degradation of the application that has its data overwritten during periods with bursty arrivals, but causes a low overhead since no resources are reserved for the gateway arrivals. However, the approach poses some topological challenges. Due to the double ring topology in Sercos III, the master node is likely to receive different frames from each ring. In that case the master has to choose which frame to use. In EtherCAT, the line topology causes the order of the nodes to define how nodes are prioritized, i.e. nodes closer to the end of the line are less likely to have its data overwritten.

Lastly, a combination of the two above methods is also possible, where multiple slaves share the same resources following a contention-based statistical multiplexing scheme. This approach does not require the use of a gateway, and provides flexibility and allows for relatively high utilization of the resources since they are multiplexed by many applications.

5.2 Single-Cell Network

In order to analyze the impact of multiplexing between different network slices we consider the network depicted in Figure 5.2 representing a cell in a factory producing personalized medicine. The network comprises a master controller, a pipetting machine mounted on a robotic arm that dispenses drug substances, and a weighing scale which measures the actual amount of dispensed drug. Furthermore, the network contains a wireless transceiver which receives measurements from sensors located around the cell, and is connected to a HID device which allows an operator to supervise the manufacturing process. Besides the pipetting machine and the scale, the wireless transceiver and the HID device act as Sercos slaves. The individual sensors are not Sercos slaves but are connected to the network through the wireless transceiver. The corresponding traffic requirements are listed in Table 5.1, where M2S is master-to-slave and S2M is slave-to-master. The master device contains a closed-loop controller of the pipetting machine which requires periodic communication from the master to the pipetting machine, and from the pipetting machine to the master. We assume that the pipetting machine has six degrees of freedom (moving parts and pipettes) and that each of these requires an exchange of 128 bytes. The scale requires a transmission of 256 bytes every 10 milliseconds. A number of wireless sensors are placed around the cell to provide general data about the process, the environment, etc. We assume that the sensor traffic is event based (e.g. triggered by changes in the environment) so that the inter-arrival times are well modelled by an exponential distribution. Each sensor

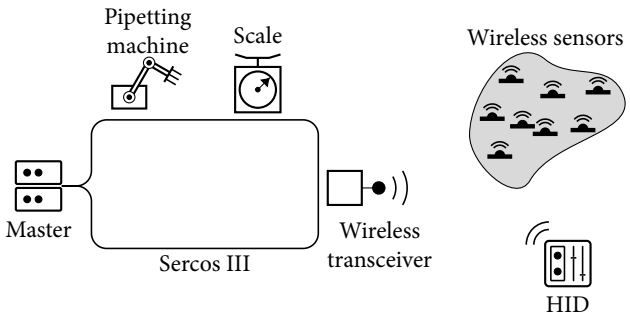


Figure 5.2: The scenario considered in the case study.

Application	Direction	Type	Period	Size	Cycle time	PER
Master control	M2S	Periodic	1 ms	$6 \cdot 128$ B[1]	1 ms	10^{-6}
Master control	S2M	Periodic	1 ms	$6 \cdot 128$ B[1]	1 ms	10^{-6}
Scale	S2M	Periodic	10 ms	256 B[1]	10 ms	10^{-9}
Wireless sensor	S2M	Poisson	0.1 s–1 hour	32 B[1]	1 ms	10^{-9}
HID stream	M2S	Periodic	20 ms	20 kB[2]	10 ms	10^{-2}

Table 5.1: Traffic requirements in the scenario considered in the case study. Cycle times and PERs are maximum values.

reading occupies 128 bytes and we consider mean reporting rates of 0.1 second up to 1 hour. Since the sensor reportings are rare and may be power constrained, the required Packet Error Rate (PER) is very low, 10^{-9}. Furthermore, since a reading may trigger changes in the system, the latency requirement is also very strict. The HID device receives a video stream with frames periodically transmitted by the master device. Due to compression, etc., the size of each frame follows an exponential distribution. A good user experience requires a relatively low latency (10 ms) but the PER is not critical (10^{-2}). Throughout the analysis we assume that the PER of the Sercos III link is $R_{\mathrm{L}} = 1 - 10^{-11}$.

For the purpose of analyzing the influence of multiplexing between network slices on the ability to satisfy the requirements, two network slice allocations which are based on different approaches are considered. We study the case where individual network slices are allocated to each requirement, but allow for resource multiplexing between the slices. In the first approach, a number of shared resources are allocated in each cycle for sporadic traffic. In the second approach, no resources are allocated for the sporadic traffic, but it is allowed to overwrite the cyclic traffic at the cost of reduced reliability of the cyclic traffic. In both cases we compare the utilization to the probability that the requirements are fulfilled.

5.2.1 Conservative Slicing

We first consider a conservative network slice allocation where resources are reserved for all flows except the HID stream which is served in the UC channel. A cycle time of 1 ms is assumed since this is the most strict cycle time requirement given by the appli-

[1]Deterministic.
[2]Exponentially distributed.

cations. Whether the application requirements are satisfied depends on the random factors in the system. Since the wireless sensor reportings arrive according to a Poisson process, the number of instantaneous reportings influences whether they can be delivered within 1 ms. In particular, if more reportings arrive than there are allocated resources, then only some of them can be transferred within the cycle time of 1 ms. On the other hand, allocating many resources leads to a low utilization due to infrequent transmissions. Therefore, this network slicing scheme poses a trade-off between the utilization and the rate of sensor reportings for which the the reliability requirements can be satisfied. The total amount of payload data in the MDT frames is

$$L'_{\mathrm{MDT}} = \underbrace{9 \cdot 6}_{\text{SVC}} + \underbrace{6(128+2)}_{\text{Master control}}. \tag{5.1}$$

Similarly, the number of payload bytes in the AT frames is given by

$$L'_{\mathrm{AT}} = \underbrace{9 \cdot 6}_{\text{SVC}} + \underbrace{N_{\mathrm{sensor\_conn}}(128+2)}_{\text{Wireless sensors}} + \underbrace{256+2}_{\text{Scale}} + \underbrace{6(128+2)}_{\text{Master control}}, \tag{5.2}$$

where $N_{\mathrm{sensor\_conn}}$ is the number of Sercos III connections allocated for wireless sensor traffic. To simplify the analysis, we assume that Sercos III connections can be perfectly packed in the Ethernet frames. Under this assumption, the total amount of MDT and AT bytes on the wire, including 26 + 6 bytes of overhead per Ethernet frame, is given by

$$L_{\mathrm{MDT}} = (26+6)\left\lceil \frac{L'_{\mathrm{MDT}}}{1494} \right\rceil + L'_{\mathrm{MDT}}, \tag{5.3}$$

$$L_{\mathrm{AT}} = (26+6)\left\lceil \frac{L'_{\mathrm{AT}}}{1494} \right\rceil + L'_{\mathrm{AT}}, \tag{5.4}$$

where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x. Because Sercos III is based on 100 Mbit/s Ethernet, the number of bytes that can be transferred during one cycle (1 ms) is 12.5 kB. Ignoring inter-frame gaps, the amount of data available in the UC channel is

$$L_{\mathrm{UC}} = 12500 - L_{\mathrm{MDT}} - L_{\mathrm{AT}}. \tag{5.5}$$

The number of connections to reserve for wireless sensor traffic, $N_{\mathrm{sensor\_conn}}$, depends on the number of sensors and the reliability requirement. Assuming sensor reportings are triggered independently, the aggregate number of reportings is also Poisson distributed with rate $\lambda_{\mathrm{A}} = N_{\mathrm{sensors}}\lambda_{\mathrm{s}}$ where N_{sensors} is the number of sensors and λ_{s} is the rate of individual sensor reportings relative to the cycle time. The reliability of a reporting is given by the complementary of the probability that a slot is available for transmission and the Sercos transmission succeeds:

$$R_{\mathrm{sensor}} = 1 - \left(1 - \Pr(\text{slot unavailable})\right)\left(1 - 10^{-11}\right). \tag{5.6}$$

To satisfy the reliability constraint, we must have $R_{\mathrm{sensor}} \geq 1 - 10^{-9}$. The probability that a slot is unavailable is equivalent to the probability that more reportings are generated than the number of reserved connections, i.e. $\Pr(A(t) > N_{\mathrm{sensor\_conn}})$ where $A(t)$ is the number of reports generated between cycles $t-1$ and t. It follows from the Poisson

distribution that

$$\Pr(A(t) > N_{\text{sensor\_conn}}) = 1 - e^{-\lambda_A} \sum_{i=0}^{N_{\text{sensor\_conn}}} \frac{\lambda_A^i}{i!}. \tag{5.7}$$

Suppose that the minimum $N_{\text{sensor\_conn}}$ that causes the requirement to be satisfied is allocated. In most cycles, the resources will be unused since the rate of sensor transmissions is very low compared to the cycle time. This is illustrated in Figure 5.3a for four different reporting rates. Even with a very high number of sensors or a high transmission rate the utilization is very low. In industrial systems, where the resources per cycle are very limited, this may have a significant impact on the scalability of a system.

The impact of the overhead can also be illustrated by considering the HID stream. As the number of connections required to serve the wireless sensors increases, the number of resources in the UC channel decreases, and so do the probability of satisfying the latency requirement of the HID stream. The HID stream generates frames periodically of exponentially distributed size and has a latency requirement of at most of 10 ms (10 cycles). Since it is often desired to transmit only the most recent frame, we assume that frames do not queue. Instead, we seek the probability that a frame requires more than 10 cycles to be transmitted, or equivalently, that the size of a frame, including Ethernet overhead, is greater than $10L_{\text{UC}}$ bytes. Assuming that L_{UC} is evenly divisible by the length of an Ethernet frame, the required bytes to transmit are given by

$$L = 26\left\lceil \frac{L'}{1500} \right\rceil + L', \tag{5.8}$$

where L' is the number of application bytes to transfer and L is the required bytes on the wire including overhead. We may obtain a simple bound on L by using $\lceil x \rceil \leq x + 1$:

$$L \leq L'\left(\frac{26}{1500} + 1\right) + 26. \tag{5.9}$$

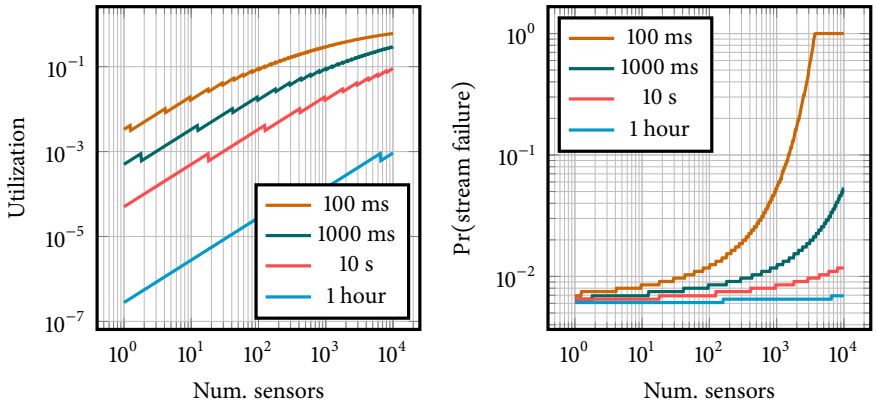
Since the exponential distribution is closed under scaling, the probability that L exceeds $10L_{\text{UC}}$ is less than to the probability that L' exceeds $(1500/1526)(10L_{\text{UC}} - 26)$. The failure probability is hence given by

$$\Pr(L > 10L_{\text{UC}}) \leq e^{-\left(\frac{1500}{1526}\right)(10L_{\text{UC}}-26)\lambda_F}, \tag{5.10}$$

where $1/\lambda_F = 20000$ is the mean frame size. The failure probability is illustrated in Figure 5.3b. As the number of sensors increases, the probability of satisfying the latency requirement of the HID stream decreases. The lower bound on the stream failure probability represents the case where no resources are allocated for the wireless sensors. Taking the low utilization into account, it is desirable to search for a better multiplexing scheme that allows the latency requirement to be satisfied for a higher number of sensors.

5.2.2 Slice Overwriting

Motivated by the results in the previous section, we now consider an allocation of network slices which has significantly higher utilization. In this scheme, network slices are allowed to overwrite resources allocated to other network slices at the cost of re-



(a) Sensor connection utilization vs. number of sensors.

(b) UE stream latency violation probability vs. number of sensors.

Figure 5.3: Utilization and HID stream failure probability in scenario 1.

ducing the reliability of the overwritten slice. Specifically, the wireless sensors can overwrite the connections of the master control network slice. Since the master control only requires a reliability of $1 - 10^{-6}$, it can tolerate more errors than the Sercos III technology introduces. This can exploited to transmit sensor data without having to reserve resources in every cycle. We assume that a wireless sensor reporting overwrites a random (uniformly distributed) connection in the master control network slice. Since the access to the Sercos III link is performed by the wireless receiver, we assume that no collisions occur, i.e. two sensors do not attempt to overwrite the same connection.

Consider a connection C_k in the master control slice and let $A(t)$ denote the number of sensor reportings to be transmitted in cycle t. The probability that connection C_k is overwritten by a sensor reporting in cycle t, denoted $C_k(t)$ is given by

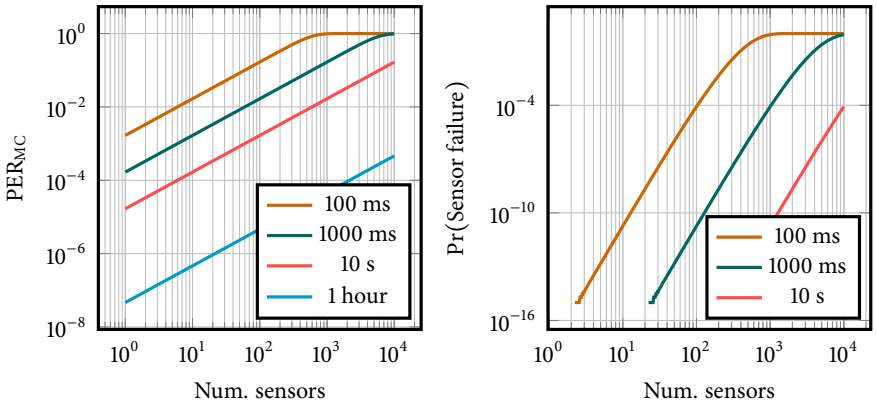
$$\Pr(C_k(t)) = \Pr(C_k(t) \mid A(t)) \Pr(A(t)). \tag{5.11}$$

As the overwritten connections are chosen according to a uniform distribution, $\Pr(C_k(t) \mid A(t))$ is the probability that C_k belongs to a random subset of size $A(t)$ of the total N connections:

$$\Pr(C_k(t) \mid A(t)) = \begin{cases} \frac{A(t)}{N} & A(t) \leq N, \\ 1 & \text{otherwise.} \end{cases} \tag{5.12}$$

Marginalizing over the Poisson distributed $A(t)$ we have

$$\begin{aligned} \Pr\left(C_k(t)\right) &= \sum_{n=0}^{\infty} \Pr\left(C_k(t) \mid A(t)\right) \Pr\left(A(t)\right) \\ &= \sum_{n=0}^{N} \Pr(C_k(t) \mid A(t)) \Pr\left(A(t)\right) + \sum_{n=N+1}^{\infty} \Pr\left(A(t)\right) \\ &= \sum_{n=0}^{N} \frac{\lambda_{\mathrm{A}}^n e^{-\lambda_{\mathrm{A}}} n}{n! N} + \sum_{n=N+1}^{\infty} \frac{\lambda_{\mathrm{s}}^n e^{-\lambda_{\mathrm{A}}}}{n!} \\ &= \sum_{n=0}^{N} \frac{\lambda_{\mathrm{s}}^n e^{-\lambda_{\mathrm{A}}} n}{n! N} + \sum_{n=0}^{\infty} \frac{\lambda_{\mathrm{s}}^n e^{-\lambda_{\mathrm{A}}}}{n!} - \sum_{n=0}^{N} \frac{\lambda_{\mathrm{s}}^n e^{-\lambda_{\mathrm{A}}}}{n!} \end{aligned}$$



(a) Master control transmission error vs. number of sensors.

(b) Sensor reporting error vs. number of sensors. The error probabilities for reportings with 1 hour intervals are outside the range of the plot.

Figure 5.4: Transmission failure probabilities in case of slice overwriting.

$$
\begin{aligned}
&= \sum_{n=0}^{N} \frac{\lambda_s^n e^{-\lambda_A}}{n!} \left(\frac{n}{N} - 1 \right) + e^{-\lambda_A} \left(\sum_{n=0}^{\infty} \frac{\lambda_s^n}{n!} \right) \\
&= \sum_{n=0}^{N} \frac{\lambda_s^n e^{-\lambda_A}}{n!} \left(\frac{n}{N} - 1 \right) + e^{-\lambda_A} e^{\lambda_A} \\
&= \sum_{n=0}^{N} \frac{\lambda_s^n e^{-\lambda_A}}{n!} \left(\frac{n}{N} - 1 \right) + 1. \qquad (5.13)
\end{aligned}
$$

Since the master control transmission using connection C_k is successful if the connection is not overwritten and the Sercos frame is not lost, the new PER of the master control slice is then given by

$$
\begin{aligned}
\mathrm{PER}_{\mathrm{MC}} &= 1 - (1 - \Pr(C_k(t)))\, R_{\mathrm{L}} \\
&= 1 + R_{\mathrm{L}} \sum_{n=0}^{N} \frac{\lambda_A^n e^{-\lambda_A}}{n!} \left(\frac{n}{N} - 1 \right). \qquad (5.14)
\end{aligned}
$$

The number of master control connections in each cycle is limited to N, and therefore there is a risk that more sensor reportings arrive than there are connections. This is equivalent to (5.7) in the conservative slicing approach with $N_{\mathrm{sensor\_conn}} = N$.

Using the values $N = 6$ and $R_{\mathrm{L}} = 1 - 10^{-11}$ as defined above, the master control transmission error and the sensor reporting error is shown in Figures 5.4a and 5.4b. In these results, it should be taken into account that the number of resources left for the UC channel is independent of the number of sensors, and hence the HID stream requirements are always satisfied. Furthermore, no additional overhead is introduced and the utilization in the RT channel remains high. The allocation, however, does for most cases not satisfy the reliability requirements for $\mathrm{PER}_{\mathrm{MC}}$. Therefore, it may be necessary to reserve some resources for alarms while allowing the master control connections to be overwritten when needed, and hence reach a compromise between reliability and utilization.

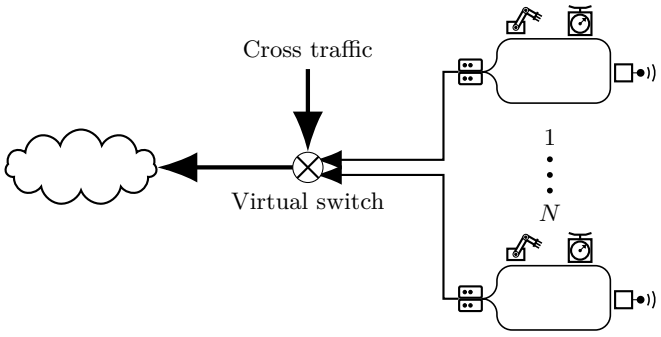


Figure 5.5: The scenario with connectivity between N cells and the cloud.

5.3 Cloud Connectivity

The network described in the previous section is now extended to include N identical cell networks which are all connected to the cloud through a single 1 Gbit Ethernet link (Figure 5.5). Furthermore, this link is shared with other applications which collectively generate cross traffic consisting of packets of 1500 bytes with exponentially distributed inter-arrival times. Suppose that the sensor reportings from the wireless sensors and the weighing scale need to be sent to the cloud for further processing and storage. We assume that the end-to-end delay for the wireless reportings must be below 5 ms and the scale measurements below 20 ms, and that the constraints must be satisfied with reliability at least $1 - 10^{-6}$. Both classes of traffic arrive to the master device at specific time instances according to the cycle time of the cell networks (i.e. in bursts). However, the number of alarm reportings in each cycle is random, and scale measurements only arrive every ten cycles. For simplicity, we ignore the propagation delay since it is constant and trivial to include, and we assume that buffers are sufficiently large to prevent packet loss. Nevertheless, it is important to notice that the propagation delay may be significant in larger networks due to the low latency requirements.

Before analyzing end-to-end guarantees in this particular scenario, we state some general observations which may provide some insight into the overall constraints in the system. Consider a traffic flow $\mathcal{F}$ from a source to a destination which has an end-to-end delay $D = \sum_{i \in \mathcal{F}} D_i$ where D_i is the delay introduced by link i including queuing. Suppose we have a requirement to the end-to-end delay stating that $Pr(D > D') \leq \epsilon$ for some D' and ϵ. Assuming that the delays along the path D_i are independent, the probability density function of D is given the convolution of the distribution functions for the individual links

$$p_D(t) = p_{D_1} * p_{D_2} * \ldots * p_{D_N}(t). \tag{5.15}$$

We are unlikely to obtain a closed-form expression of the density function $p_D(t)$ unless we constraint $p_{D_i}(t)$ to have a certain form. However, this is likely to lead to large inaccuracies in the model, and it may even be difficult to determine whether the model is conservative or optimistic compared to the actual delay distribution. Furthermore, even $p_{D_i}(t)$ may not be straightforward to obtain without strong assumptions on the system since it depends on the traffic characteristics in the link. Instead, an apparent approach is to use analyze the problem using Stochastic Network Calculus (SNC). However, as described in Section 3.3 it is not trivial to model the periodic traffic and service in the Sercos link. Alternatively, we could model the system using

deterministic network calculus. However, in that case we cannot model probabilistic multiplexing and hence we need to allocate resources to each wireless sensor which results in a very low utilization of the Sercos link. This in not feasible in practice since the resources in the Sercos link are very limited.

We may instead model each sub-network independently. In general, we assume that sub-network i (comprising one or more links) guarantees a certain delay D_i' with reliability R_i, i.e. $\Pr(D_i > D_i') \leq 1 - R_i$. Under the assumption of independent link delays, the end-to-end requirement is fulfilled if $\sum_{i\in\mathcal{F}} D_i' \leq D'$ and $1 - \prod_{i\in\mathcal{F}} R_i \leq \epsilon$. Furthermore, we may state that this is a conservative requirement since $1 - \prod_{i\in\mathcal{F}} R_i \leq \epsilon$ is a sufficient but not a necessary condition for the total delay D to be below D' with the desired reliability. For instance, we may have two links for which we know certain delay bounds, say, $\Pr(D_1 > 10) \leq 10^{-9}$ and $\Pr(D_2 > 10) \leq 10^{-9}$, an end-to-end requirement of, say, $\Pr(D > 50) \leq 10^{-9}$ *may* still be satisfied even though the product of link reliabilities is less than required. However, we cannot guarantee this based on the knowledge of the system. From the observation above it follows that a necessary condition for satisfying the requirement (assuming no redundant paths) is that the reliability of each link is at least $1 - \epsilon$. While this may seem trivial, it provides a fundamental limit to the reliability that can be provided in a system without parallel paths.

In the scenario under consideration we divide the network into two sub-networks: the Sercos links and the switched Ethernet links, subsequently referred to as cell and backend network, respectively. To proceed the analysis, we assume that we have a cell network allocation which satisfies the requirements listed in Table 5.1. For the scale traffic which requires an end-to-end guarantee of $\Pr(D > 20\text{ ms}) \leq 10^{-6}$, it follows that the requirement to the backend network is

$$\begin{aligned}\Pr(D > 20 - 10\text{ ms}) &\leq 1 - \frac{1-10^{-6}}{1-10^{-9}} \\ &= 9.99 \cdot 10^{-7}.\end{aligned} \tag{5.16}$$

Similarly, for the alarm traffic we have the backend requirement

$$\Pr(D > 5 - 1\text{ ms}) \leq 9.99 \cdot 10^{-7}. \tag{5.17}$$

To provide guarantees in the backend network we may either use stochastic network calculus or deterministic network calculus. Since the arrivals in this scenario are generated periodically following the cycle times of the Sercos links, the arrival process is not well modelled by stochastic network calculus. Instead, the arrivals from the cell network may be modelled using deterministic network calculus. However, we may not immediately be able to model the Poisson cross traffic. To cope with this, we may either be able obtain some bound on the arrival process, or we may require a higher priority to the deterministic traffic so that the Poisson flow does not influence the real-time traffic. Depending on the application, providing an upper bound may not be straightforward in practice, at least not without some degree of uncertainty or a high overprovisioning factor. Alternatively, an upper bound may be enforced by applying traffic shapers, e.g. a token bucket, to the arrival process. In this scenario, since we do not have any requirements for the Poisson flow, we consider the first method and assume that the traffic from the cells is queued with higher priority than the Poisson flow. Specifically, we consider a scenario where the switch has three prioritized

queues. Furthermore, to simplify the analysis, we assume that DetNet/TSN is used so that there is support for packet preemption in the switch. We further assume that the preemption is instantaneous and that it does not introduce additional overhead.

We model the periodic arrivals from each cell as affine functions which provide upper bounds on the arrivals. Since the scale measurements generate 256 bytes every 10 ms, the arrival rate including 26 bytes Ethernet header is bounded by [3]

$$\begin{aligned} A_{\mathrm{sc}}(t) &= 282\left\lfloor \frac{t}{10} \right\rfloor + 282 \\ &\leq 282\frac{t}{10} + 282. \end{aligned} \tag{5.18}$$

We can obtain a similar bound on the arrival process of the wireless alarms. Although the alarms are generated following a Poisson distribution, but the cell network limits the effective arrival rate. Therefore, we consider the worst-case arrival and assume that N slots are available for the alarms in each cycle so that at most $128N$ bytes arrive. Assuming that each alarm is encapsulated in its own Ethernet frame it follows that the arrival process is bounded by

$$A_{\mathrm{al}}(t) \leq 154Nt + 154N. \tag{5.19}$$

The service curve is defined as a constant rate server which can serve 1 Gbit/s (125 kB/ms):

$$S(t) = 125 \cdot 10^3 t. \tag{5.20}$$

We initially assume that the arrivals from scale measurements and from alarms share the same queue in the switch (highest priority). The aggregate arrival to the switch is given by

$$A(t) \leq c\left(\frac{282}{10} + 154N\right)t + c(282 + 154N), \tag{5.21}$$

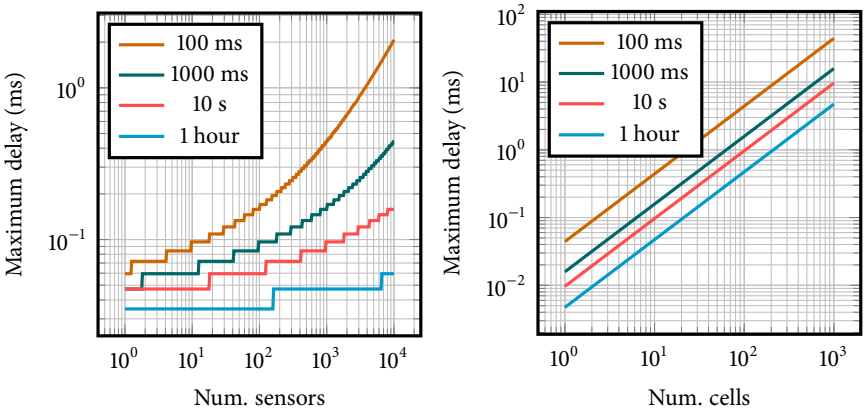
where c is the number of cells. Since these arrivals are higher prioritized than the cross traffic, we can effectively ignore the impact of the cross traffic and may readily obtain a bound on the delay using deterministic network calculus as

$$W(t) \leq \frac{c(282 + 154N)}{125 \cdot 10^3}. \tag{5.22}$$

This is illustrated in Figure 5.6. Figure 5.6a shows the worst-case delay from the master device to the cloud in a scenario with $c = 10$ cells, while Figure 5.6b shows the scenario with 1000 sensors but various numbers of cells. In both cases, it is clear that a significant number of sensors and a high number of cells can be served by the same link given that the traffic has highest priority.

We now consider the case where the traffic from the wireless alarms is prioritized higher than the scale measurements, while the Poisson cross traffic is still served by

[3] Depending on the network infrastructure, additional overhead may be needed, e.g. due to IP. However, to simplify the scenario we only consider Ethernet.



(a) Increasing rate of alarms.

(b) Increasing number of cells each with 1000 sensors.

Figure 5.6: Maximum delay from cell master to cloud.

the lowest priority queue. The waiting time bound on the wireless alarms are given by

$$W_{\text{al}}(t) \leq \frac{c154N}{125 \cdot 10^3}. \tag{5.23}$$

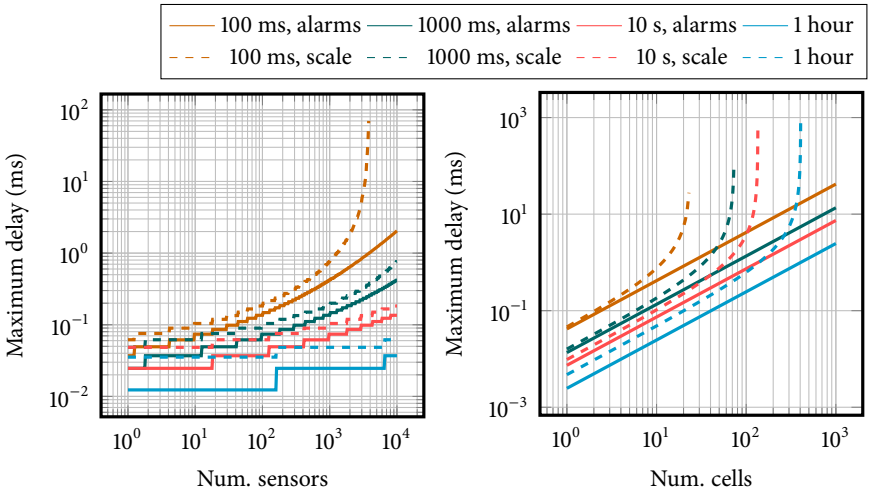
The resulting leftover service available for the scale measurements is

$$S_{\text{lo}}(t) \geq (125 \cdot 10^3 - c154N)t - c154N, \tag{5.24}$$

which yields the following waiting time bound

$$W_{\text{sc}}(t) \leq \frac{c(282 + 154N)}{125 \cdot 10^3 - c154N}. \tag{5.25}$$

This is illustrated in Figure 5.7 where the impact of the alarm traffic on the scale traffic is evident. When the number of alarm arrivals reaches a certain point, the cell traffic increases exponentially due to the fact that alarm traffic occupies almost the entire service. However, the alarm traffic is completely isolated from the scale traffic, which ensures that its end-to-end characteristics can be determined independently of the scale traffic. Furthermore, the maximum delay experienced by the alarm traffic is lower than in the case with a single shared queue since the alarm traffic is served before the scale traffic.



(a) Increasing rate of alarms.

(b) Increasing number of cells each with 1000 sensors.

Figure 5.7: Maximum delay from cell master to cloud in the case where alarms are prioritized higher than scale traffic.

6 Abstract Network Representation

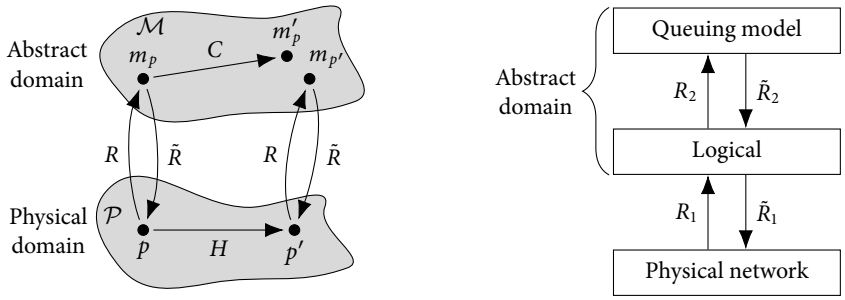
While the analysis in the previous chapter shows how end-to-end properties can be obtained in an industrial network consisting of Sercos III and Ethernet, it is desired to decouple the analysis from the physical technologies in order to support heterogeneous networks. This chapter defines an abstract description of physical networks which is used in the process of constructing network slices, or specifically, to allocate network resources to provide a certain functionality with end-to-end guarantees. Furthermore, a description of corresponding network slice requirements is defined. The purpose of the descriptions is to hide the physical hardware and technologies from the NFV-MANO mechanism so that it, to a certain extent, is technology independent. However, before introducing the abstract descriptions, we formally define what we mean by an abstract description, as well as which operations that can be done in the abstract domain, based on the work presented in [21, 55, 56].

We are interested in a certain abstract *representation* of some physical object, in this case a computer network. This representation may be a graph, a queuing model, etc. Depending on the abstract representation, we may be able to do operations in the abstract domain. For instance, we could add an edge to the graph or increase the service rate of a queuing model. Such an operation results in a new representation in the abstract domain, namely a new graph or a new queuing model. Often, operations in the abstract domain represent an operation in the physical domain. Adding an edge to the graph may represent the operation of adding a new physical link between two nodes, and increasing the service rate may represent an increase in the clock rate of a server. If the abstract representation provides a good description of the physical object, then the operation in the abstract domain should be closely related to the corresponding operation in the physical domain.

To describe this relation formally, we consider the model depicted in Figure 6.1a where an object that exists in the physical domain $\mathcal{P}$ is denoted by $p \in \mathcal{P}$, and the abstract domain $\mathcal{M}$ comprises of all abstract objects $m \in \mathcal{M}$. Furthermore, we let m_p denote the abstract representation of the physical object p. An operation in the abstract domain is represented by the mapping $C : \mathcal{M} \to \mathcal{M}$ and an operation in the physical domain by $H : \mathcal{P} \to \mathcal{P}$. Finally, the representation relation $R : \mathcal{P} \to \mathcal{M}$ denotes the mapping from a physical object to an abstract representation.

If an operation in the abstract domain $m_p \to m'_p$ describes the corresponding physical operation $p \to p'$ accurately, then the abstract representation of the new physical object should be "close" to the abstract representation resulting from the operation. The abstract evolution is said to *commute* if $\|m'_p - R(p')\| \leq \epsilon$ for some norm $\|\cdot\|$. If a set of objects and evolutions commute under the same representation, then m_p is said to be a *faithful representation* of p for the evolutions $C(m_p)$ and $H(p)$. Having a faithful representation allows us to "trust" the abstract representation.

The inverse representation relation, e.g. the mapping from an abstract representation to a physical domain, is called instantiation and denoted by $\tilde{R} : \mathcal{M} \to \mathcal{P}$. If such a mapping exists, then an object in the abstract domain may instantiate itself in the physical domain. For instance, a queuing model of a network may be instantiated at the physical level by applying a specific configuration to the physical objects. However, while a mapping from the physical domain to the abstract domain always exists, instantiation is not always possible. For example, we may not be able to connect two nodes, even though adding an edge to a graph in the abstract domain is a simple



(a) The definition of abstract representations considered in this chapter.

(b) A physical object can have multiple abstract representations.

Figure 6.1: Graphical representation of the an abstraction.

operation.

A physical object can have multiple representations in the abstract domain which allow for different operations. As illustrated in Figure 6.1b, a network may have a logical representation which is used for configuring the physical nodes, and a queuing model which allows for performing mathematical operations. In such a system it may not be possible to instantiate a physical object directly from the queuing model, but instead the queuing model must first be mapped to the logical representation which allows instantiation in the physical domain.

To complete the definition of an abstraction, we define the relation between abstraction and virtualization. In this context we refer to "real resources" and "virtual resources", and say that virtualization is the mapping between real and virtual resources. Note that real resources are not physical resources, but refer to representations that represent physical resources. On the contrary, virtual resources do not represent physical resources, and so, a mapping from physical resources to virtual resources may not exist. A requirement for the mapping to be a virtualization is that the virtual resources are represented in the same way as the real resources. In other words, the virtualization mapping must be between objects at the same representation level. Specifically, let $\mathcal{R} \subseteq \mathcal{M}'$ and $\mathcal{V} \subseteq \mathcal{M}'$ denote the real and virtual resources, respectively, at some representation layer $\mathcal{M}' \subseteq \mathcal{M}$. We define a virtualization as a mapping $f : \mathcal{V} \to \mathcal{R} \cup \{t\}$, where t is a special symbol indicating that a real resource for the corresponding virtual resource does not exist. In a computer network, a graph representation may be virtualized into a new graph, where nodes are connected in a different way.

6.1 Physical Network Representation

Based on the abstraction framework defined above, an abstract representation of industrial networks which can be used by the NFV-MANO mechanism is defined. However, since the NFV-MANO mechanism has many functions, it is unlikely that a single representation is suitable for all functions. Several frameworks for describing physical networks exist, notably the set of attributes defined by the Metro Ethernet Forum (MEF) [57], which is aimed at describing the end-to-end Ethernet service provided by a network service provider. This includes the mean and percentiles of the frame delay distribution, frame loss probabilities, data rate, etc. However, while the MEF Ethernet services attributes provide a unified way of specifying the service provided by an

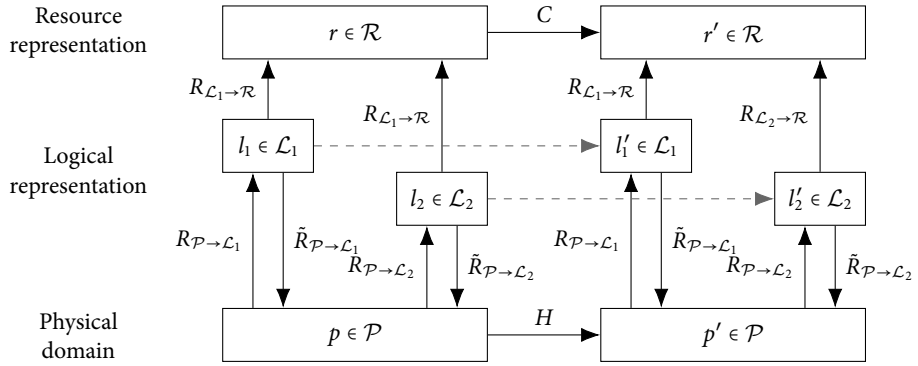


Figure 6.2: The abstraction model considered for the physical network.

Ethernet service provider, its limitation to regular Ethernet makes it inapplicable for industrial Ethernet based protocols. Furthermore, due to the many parameters of industrial Ethernet protocols, such as cycle times, resource allocation, etc., an industrial Ethernet link cannot be considered as a black box which provides a certain service, but the individual resources need to be described.

As stated earlier, we seek a representation that can be used by the NFV-MANO mechanism to allocate end-to-end network resources required to provide a given functionality. Furthermore, the focus is on determining which resources to allocate, but not how to instantiate an allocation. Therefore, the representation presented in this section has the simple purpose of describing the resources that the network makes available for the NFV-MANO. For convenience, we refer to this representation as the resource representation.

The overall abstraction hierarchy is illustrated in Figure 6.2, where the resource representation is the highest abstraction layer, and provides a technology independent representation of resources in the network. Below the resource representation is the logical representation where configuration of the physical components such as queues, hypervisors, etc. is done. Hence, the logical representation is technology and vendor specific. While the the NFV-MANO in principle could operate with the logical representation, this would be inconvenient since the logical layer consists of many different representations. Formally specifying the logical representation is outside the scope of this report, but its existence and the fact that a logical representation can be used for instantiation in the physical domain is important.

Since the resource representation describes a network, it is natural to represent it as a graph. Moreover, as a physical link may provide several services, such as a real-time service and a non-real-time service, or multiple queue priorities, the network forms a directed multigraph where each edge represents a certain service provided by a link. We denote the graph $G = (V, E, b)$ where V is the set of vertices (nodes) and E is the set of edges (links). The edges map to source and target vertices through $b : E \to \{(u, v) : u, v \in V\}$. An example of this representation is shown in Figure 6.3 where the physical network in Figure 6.3a is represented as a multigraph in Figure 6.3b. The Industrial Ethernet within each cell is represented in the resource representation as a complete graph interconnecting all the nodes that it connects. The edge line type indicates that the edges have different characteristics. For instance, the link in the cell provides real-time and non-real-time services, and the connection between the switch node and the cloud node has three queues with different priorities. Note that

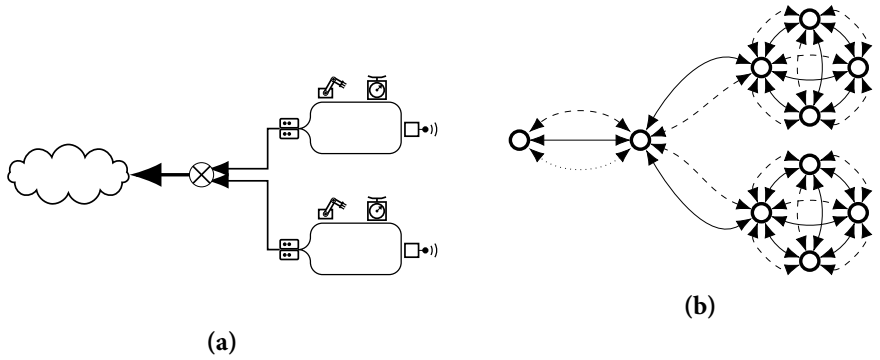


Figure 6.3: The relation between physical (a) and resource (b) network representations.

we for simplicity have used bidirectional edges in this example, although the directed multigraph G contains unidirectional edges.

In the following sections we define the characteristics of the edges and vertices in the multigraph, respectively, where we refer to the edges as resource links and the vertices as resource nodes.

6.2 Resource Nodes

The resource nodes describe the servers, switches, etc., in the network which provide resources in the form of processing, storage, or hardware dependent functionality such as a base station for wireless connectivity. Although there are many parameters that are important to determine whether a given functionality can be deployed on a node, we only consider a subset consisting of most important for the allocation of network slices. We define a resource node $N_j \in V$ as the set of characteristics $N_j = (H_j, C_j, R_j)$, where H_j denotes hardware capabilities, C_j is the node capacity, and R_j is the node failure rate. In the following sections we define these characteristics in turn.

6.2.1 Hardware Capabilities

Some nodes may contain hardware that enables certain functionality. This includes wireless transceivers, the capability of acting as master in a master-slave network/bus, etc. We do not define the specific hardware capabilities since these are very application dependent and may be treated equivalently as binary constraints by the NFV-MANO mechanism. We assume that a node may either support a given functionality or not, and hence it can be represented as a binary symbol, so that $H_j = \{H_j^{(1)}, H_j^{(2)}, \ldots, H_j^{(K)}\}$ where

$$H_j^{(k)} = \begin{cases} 1 & \text{node } j \text{ has hardware capability } k, \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

6.2.2 Node Capacities

A node has a certain capacity with respect to processing, memory, network buffer sizes, memory bandwidth, storage etc. Although all these capacities are important to take into account in a deployment, only processing and memory resources are considered in this representation since they are deemed most important.

The processing capacity of a node can be specified from several parameters, includ-

ing instructions per second, clock frequency, number of CPUs. However, the actual performance depends on several other parameters such as architecture, instruction set, CPU cache size, etc. For this reason, processing resources in cloud computing are often specified in terms of virtual CPUs, where a virtual CPU has clearly defined properties. We adapt this approach to our abstraction model, and specify the node processing capacity as the number of virtual CPUs available on a node. We assume that a the requirements in terms of virtual CPUs are known for a given VNF. Such requirements may be determined by benchmark testing VNFs on machines that provide equivalent virtual CPUs. Since a physical node may be shared among multiple virtual machines the actual processing resources are influenced by the load of the other machines. We assume that this impact is handled by proper scheduling between the virtual machine and by reasonable provisioning of resources.

In addition to processing resources, a node capacity is also defined by the available memory. Contrary to the CPUs, we assume that memory is reserved to an application and hence not shared among several virtual machines.

Formally, we specify the node capacity as $C_j = (C_j^{\text{cpu}}, C_j^{\text{mem}})$ where $C_j^{\text{cpu}} \in \mathbb{N}_{\geq 0}$ is the number of virtual CPUs and $C_j^{\text{mem}} \in \mathbb{N}_{\geq 0}$ is the total node memory in bytes.

6.2.3 Node Reliability

A node may fail due to aging hardware, memory or data corruptions, software bugs, etc. In reliability engineering systems are often described in terms of reliability and availability. Reliability refers to the probability that the system does *not* fail within a certain time period, and availability is the probability that a system is functioning at a certain time instant [58]. For hardware components, reliability is often assumed to follow a bathtub shaped reliability curve where the risk of failure is high in the beginning (the so-called burn-in period), then decreases and finally increases again (due to wear-out). This characteristic does not hold for software, which is more likely to follow a constant failure probability, with increased failure rates in the time following an update [58].

Here, we jointly consider hardware and software failures, and we refer to any of them as node failures. Notice that node failures do not include failures of virtual network functions but only failures related to the service provided by nodes. Furthermore, we assume that a node is tested sufficiently so that we can ignore the burn-in period and that the failure rate is constant, i.e. the reliability function follows an exponential distribution with constant rate R_j. Lastly, following the arguments in [59, 60] we assume that nodes fail independently and that failures do not propagate through the network.

6.3 Resource Links

Links in the resource representation describe the characteristics and resources associated with physical links. Since cyclic/deterministic and acyclic protocols differ fundamentally, we distinguish between the two types in the resource representation. Cyclic protocols provide a cyclic structure with reserved resources and a fixed data rate, and hence also a guaranteed worst-case delay. In contrast, acyclic protocols, such as switched Ethernet, are often multiplexed and introduce queuing and possibly random access mechanisms which cause stochastic delays. Therefore, it is much more difficult to provide strong guarantees with regard to end-to-end delays when acyclic

protocols are used since they depend on all traffic flows sharing the link. Furthermore, describing cyclic protocols stochastically is likely to lead to unnecessary overprovisioning of resources in order to satisfy the application requirements. Distinguishing between cyclic and acyclic protocols in the resource representation allows us to utilize both the cyclic and acyclic resources efficiently. From a formal point of view, we divide the set of edges E into two disjoint sets $E = E^{\mathrm{c}} \cup E^{\mathrm{ac}}$ where E^{c} denote the cyclic links and E^{ac} are the acyclic links.

However, cyclic and acyclic links share some common properties. Since a physical link may be modelled as several resource links, for instance if the link technology contains multiple classes of traffic or represents a bus, several resource links may be multiplexed and subject to aggregate constraints. To describe this, a number of disjoint sets $\mathcal{M}_j \subseteq E$ are defined, each containing links that are multiplexed. That is, if two links share the same resources, then they are contained in the same set. We assume that all links are contained in some multiplexing set (possibly with cardinality one), i.e

$$\bigcup_j \mathcal{M}_j = E. \tag{6.2}$$

Cyclic links contained in the same multiplexing set share the same resources in each cycle while acyclic links share the same data rate. This is elaborated in the following sections.

6.3.1 Cyclic Resource Links

The resources provided by a cyclic link may be described by the cycle time and the amount of resources available to applications within each cycle. Since reconfiguring these parameters often requires a restart of the system, it is assumed that resources are fixed. However, in a deployment intended for network slicing, some resources may be reserved specifically for network slices, and available for the NFV-MANO.

The NFV-MANO mechanism can decide whether the available resources should be reserved for specific network slices, or multiplexed between network slices, e.g. using the multiplexing schemes described in Section 5.1.

Communication over a link may fail, e.g. due to a momentary high level of noise, interference, synchronization problems, etc. While some communication protocols may attempt to increase link reliability by implementing retransmission strategies, we define the reliability of a deterministic resource link as the probability that a transmission fails, independently of whether retransmission strategies are implemented. This is motivated by the fact that some applications cannot tolerate high transmission failure rates, and that the network slicing process should be independent of the transport layer protocols. To simplify the representation, it is assumed that transmission failures are independent so that a single probability is sufficient to characterize the reliability.

The last link characteristic that is included in the description is the link delay, which is defined to be the sum of the propagation delay and serialization delay. Since queuing depends on the traffic allocated to the link, this quantity is not included in the description.

Formally, the characteristics of a cyclic resource link $L_j^{\mathrm{c}} \in E^{\mathrm{c}}$ is defined by the ordered set $L_j^{\mathrm{c}} = (c_j^{\mathrm{c}}, s_j^{\mathrm{c}}, r_j, l_j)$ where c_j^{c} denotes the cycle time (in seconds), s_j^{c} is the number of resources (in bytes) available in each cycle, r_j is the transmission reliability, and l_j is the delay in the link (in seconds). To allow for multiplexing across cyclic re-

source links, all resource links belonging to the same physical link must be contained in the same multiplexing set $\mathcal{M}_i$.

6.3.2 Acyclic Resource Links

Acyclic resource links do, contrary to cyclic resource links, not provide a deterministic cyclic structure. Instead, data transmitted over an acyclic link are queued, and transmitted when the medium is free. In this representation, acyclic resource links characterized by the data rate along with a link delay and a reliability. Similar to cyclic traffic, several acyclic resource links may share the same physical link where the individual links are scheduled access to the shared medium. Although one can imagine several types of schedulers, we limit the representation to priority based schedulers. The priority of edge j is defined by the attribute ϵ_j so that link j is served before link k if and only if $\epsilon_j^{\text{ac}} < \epsilon_k^{\text{ac}}$ and link L_j^{ac} and L_k^{ac} are contained in the same multiplexing set $\mathcal{M}_i$.

Formally, we describe an acyclic resource link $L_j^{\text{ac}} \in E^{\text{ac}}$ by $L_j^{\text{ac}} = (s_j^{\text{ac}}, \epsilon_j^{\text{ac}}, r_j, l_j)$ where s_j^{ac} is the data rate of the link (bytes/sec.), ϵ_j^{ac} is the queue priority, and r_j and l_j are the transmission reliability and link delay defined as for cyclic links.

6.4 Application Request Representation

Besides a description of the physical network, the NFV-MANO mechanism also needs a description of application requirements in order to construct network slices. This section introduces a representation similar to the resource representation of a physical network, but for describing requirements to network slices. The ETSI has a specification for a detailed high level framework for describing complete virtual networks in terms of VNFs and virtual links [61]. The framework allows for specifying requirements to the underlying physical hardware, including QoS, latency, etc., and is intended as an application level input to the NFV-MANO mechanism. However, its high detail level makes it complex and inconvenient to include in an algorithm.

A network slice consists of communication resources, a set of NFs, and resources to run these functions. Hence, an application request should contain sufficient information for the NFV-MANO to identify where the desired NFs can be instantiated, as well as the amount of communication resources to allocate. In the ETSI NFV representation [61], application requirements are represented as forwarding graphs, i.e. a graph consisting of interconnected NFs. Each node in the graph describes a NF along with its requirements with respect to processing, memory, etc., while edges describe the traffic characteristics and requirements to latency, reliability, etc. This approach is followed in the description presented here, adopted to comply with the resource representation. Specifically, an application request is a directed forwarding (multi-)graph $\widehat{F} = (\widehat{V}, \widehat{E}, \widehat{b})$ with vertices $\widehat{V}$, edges $\widehat{E}$ and $\widehat{b} : \widehat{E} \to \{(u, v) : u, v \in \widehat{V}\}$. The hat notation is used to distinguish application requests from the notation used in the resource representation.

6.4.1 Application Nodes

Following the description of representation nodes, a node requirement $\widehat{N}_j \in \widehat{V}$ is described by $\widehat{N}_j = (\widehat{H}_j, \widehat{C}_j, \widehat{R}_j)$. $\widehat{H}_j = \{\widehat{H}_j^{(1)}, \widehat{H}_j^{(2)}, \ldots, \widehat{H}_j^{(K)}\}$ are hardware requirements

so that $\widehat{H}_j^{(i)} = 1$ if hardware capability i is required. $\widehat{C}_j = (\widehat{C}_j^{\text{cpu}}, \widehat{C}_j^{\text{mem}})$ is the required node capacity, and $\widehat{R}_j$ is the required maximum failure rate.

6.4.2 Application Links

Application links describe the connectivity between application nodes. Industrial applications often transmit packets with a fixed size and rate, while other applications may provide a more sporadic pattern, and essentially produce an unlimited amount of traffic, e.g. in case of wireless sensors which may only be limited by the capacity of the wireless channel. Therefore, we divide application links into links which are rate limited and links where the traffic arrival bounds are stochastic.

Rate limited applications are assumed to have a cumulative arrival function that is bounded by an affine function $A(t) \leq \widehat{\sigma}_j t + \widehat{\rho}_j$. A link requirement is then described by $\widehat{L}_j^{\text{a}} = (\widehat{\sigma}_j, \widehat{\rho}_j, \widehat{r}_j, \widehat{l}_j)$, where $\widehat{r}_j$ and $\widehat{l}_j$ are the end-to-end reliability and delay requirements, respectively.

Stochastically bounded traffic arrivals are described by $\widehat{L}_j^{\text{s}} = (\widehat{W}_j, \widehat{r}_j, \widehat{l}_j, \widehat{s}_j)$, where $\widehat{W}_j(x, \tau)$ is a cumulative distribution function (CDF) of the number of packets generated during a period of τ seconds, and $\widehat{W}_j^{-1}(x, \tau)$ is the corresponding inverse CDF. It is assumed that a packet has a fixed (maximum) size of $\widehat{s}_j$ bytes.

6.5 Summary of Notation

In this section we restate the notation introduced above for clarity. The resource representation is defined by the directed multigraph $G = (V, E^{\text{c}}, E^{\text{ac}}, b)$ where V is the set of resource nodes, E^{c} and E^{c} are the disjoint sets of cyclic and acyclic resource links, respectively, and $b : E^{\text{c}} \cup E^{\text{ac}} \to \{(u, v) : u, v \in V\}$. Each resource node $N_j \in V$ is described by $N_j = (H_j, C_j, R_j)$. A cyclic link $L_j^{\text{c}} \in E^{\text{c}}$ is described by $L_j^{\text{c}} = (c_j^{\text{c}}, s_j^{\text{c}}, r_j, l_j)$ while an acyclic link $L_j^{\text{ac}} \in E^{\text{ac}}$ is described by $L_j^{\text{ac}} = (s_j^{\text{ac}}, \epsilon_j^{\text{ac}}, r_j, l_j)$. The elements and their corresponding types are listed in Table 6.1.

Application requests are, as listed in Table 6.2, described by a forwarding graph $\widehat{F} = (\widehat{V}, \widehat{E}^{\text{a}}, \widehat{E}^{\text{s}}, \widehat{b})$ with application nodes $\widehat{V}$ and affine edges with affine and stochastically bounded arrivals $\widehat{E}^{\text{a}}$ and $\widehat{E}^{\text{s}}$, respectively. An application node $\widehat{N}_j \in \widehat{V}$ is described by $\widehat{N}_j = (\widehat{H}_j, \widehat{C}_j, \widehat{R}_j)$ where each set element is a requirement corresponding to the resource node representation. A link $\widehat{E}^{\text{a}}$ with affine bounded arrivals are described by $\widehat{L}_j^{\text{a}} = (\widehat{\sigma}_j, \widehat{\rho}_j, \widehat{r}_j, \widehat{l}_j)$ and a link $\widehat{E}^{\text{s}}$ with stochastically bounded arrivals by $\widehat{L}_j^{\text{s}} = (\widehat{W}_j, \widehat{r}_j, \widehat{l}_j, \widehat{s}_j)$.

Parameter	Definition	Description
	Resource Graph	
G	$G = (V, E^{\mathrm{c}}, E^{\mathrm{ac}}, b),\ E^{\mathrm{c}} \cap E^{\mathrm{ac}} = \varnothing$	Resource graph.
V	$V = \{N_1, N_2, \ldots, N_{\lvert V\rvert}\}$	Set of resource nodes.
E^{c}	$E^{\mathrm{c}} = \{L_1^{\mathrm{c}}, L_2^{\mathrm{c}}, \ldots, L_{\lvert E^{\mathrm{c}}\rvert}^{\mathrm{c}}\}$	Set of cyclic resource links.
E^{ac}	$E^{\mathrm{ac}} = \{L_1^{\mathrm{ac}}, L_2^{\mathrm{ac}}, \ldots, L_{\lvert E^{\mathrm{ac}}\rvert}^{\mathrm{ac}}\}$	Set of acyclic resource links.
b	$b : E^{\mathrm{c}} \cup E^{\mathrm{ac}} \to \{(u, v) : u, v \in V\}$	Mapping from links to source and destination nodes.
	Resource Nodes	
N_j	$N_j = (H_j, C_j, R_j),\ j = 1, \ldots, \lvert V\rvert$	Resource node.
H_j	$H_j = \{H_j^{(1)}, H_j^{(2)}, \ldots, H_j^{(K)}\}$	Node hardware capabilities.
$H_j^{(i)}$	$H_j^{(i)} \in \{0, 1\}$	Indication of whether node j has hardware capability i.
C_j	$C_j = (C_j^{\mathrm{cpu}}, C_j^{\mathrm{mem}})$	Node capacity.
C_j^{cpu}	$C_j^{\mathrm{cpu}} \in \mathbb{N}_{\geq 0}$	Virtual CPUs.
C_j^{mem}	$C_j^{\mathrm{mem}} \in \mathbb{N}_{\geq 0}$	Memory (bytes).
R_j	$R_j \in [0, 1]$	Failure rate (exponential dist.).
	Resource Links	
$\mathcal{M}_j$	$\bigcup_j \mathcal{M}_j = E,\ j = 1, \ldots, J$	Multiplexing set.
L_j^{c}	$L_j^{\mathrm{c}} = (c_j^{\mathrm{c}}, s_j^{\mathrm{c}}, r_j, l_j),\ j = 1, \ldots, \lvert E^{\mathrm{c}}\rvert$	Cyclic resource link description.
c_j^{c}	$c_j^{\mathrm{c}} \in \mathbb{R}_{\geq 0}$	Cycle time (seconds).
s_j^{c}	$s_j^{\mathrm{c}} \in \mathbb{N}_{\geq 0}$	Resources per cycle (bytes).
r_j	$r_j \in [0, 1]$	Transmission reliability.
l_j	$l_j \in \mathbb{R}_{\geq 0}$	Transmission delay (seconds).
L_j^{ac}	$L_j^{\mathrm{ac}} = (s_j^{\mathrm{ac}}, \epsilon_j^{\mathrm{ac}}, r_j, l_j),\ j = 1, \ldots, \lvert E^{\mathrm{ac}}\rvert$	Acyclic resource link description.
s_j^{ac}	$s_j^{\mathrm{ac}} \in \mathbb{N}_{\geq 0}$	Data rate (bytes/sec.).
ϵ_j^{ac}	$\epsilon_j^{\mathrm{ac}} \in \mathbb{N}_{\geq 0}$	Queue priority.

Table 6.1: Notation used in the resource representation of a physical network.

Parameter	Definition	Description
	Forwarding Graph	
$\widehat{F}$	$\widehat{F} = (\widehat{V}, \widehat{E}^{\mathrm{a}}, \widehat{E}^{\mathrm{s}}, \widehat{b}),\ \widehat{E}^{\mathrm{a}} \cap \widehat{E}^{\mathrm{s}} = \varnothing$	Forwarding graph.
$\widehat{V}$	$\widehat{V} = \{\widehat{N}_1, \widehat{N}_2, \ldots, \widehat{N}_{\lvert\widehat{V}\rvert}\}$	Set of application nodes.
$\widehat{E}^{\mathrm{a}}$	$\widehat{E}^{\mathrm{a}} = \{\widehat{L}^{\mathrm{a}}_1, \widehat{L}^{\mathrm{a}}_2, \ldots, \widehat{L}^{\mathrm{a}}_{\lvert\widehat{E}^{\mathrm{a}}\rvert}\}$	Set of affine bounded application links.
$\widehat{E}^{\mathrm{s}}$	$\widehat{E}^{\mathrm{s}} = \{\widehat{L}^{\mathrm{s}}_1, \widehat{L}^{\mathrm{s}}_2, \ldots, \widehat{L}^{\mathrm{s}}_{\lvert\widehat{E}^{\mathrm{s}}\rvert}\}$	Set of stochastic bounded application links.
$\widehat{b}$	$\widehat{b} : \widehat{E}^{\mathrm{a}} \cup \widehat{E}^{\mathrm{s}} \to \{(u, v) : u, v \in \widehat{V}\}$	Mapping from links to source and destination nodes.
	Application Nodes	
$\widehat{N}_j$	$\widehat{N}_j = (\widehat{H}_j, \widehat{C}_j, \widehat{R}_j),\ j = 1, \ldots, \lvert\widehat{V}\rvert$	Application node.
$\widehat{H}_j$	$\widehat{H}_j = \{\widehat{H}^{(1)}_j, \widehat{H}^{(2)}_j, \ldots, \widehat{H}^{(K)}_j\}$	Node hardware requirements.
$\widehat{H}^{(i)}_j$	$\widehat{H}^{(i)}_j \in \{0, 1\}$	Indication of whether node j requires hardware capability i.
$\widehat{C}_j$	$\widehat{C}_j = (\widehat{C}^{\mathrm{cpu}}_j, \widehat{C}^{\mathrm{mem}}_j)$	Node capacity requirements.
$\widehat{C}^{\mathrm{cpu}}_j$	$\widehat{C}^{\mathrm{cpu}}_j \in \mathbb{N}_{\geq 0}$	Virtual CPUs.
$\widehat{C}^{\mathrm{mem}}_j$	$\widehat{C}^{\mathrm{mem}}_j \in \mathbb{N}_{\geq 0}$	Memory (bytes).
$\widehat{R}_j$	$\widehat{R}_j \in [0, 1]$	Maximum failure rate (exponential dist.).
	Application Links	
$\widehat{L}^{\mathrm{a}}_j$	$\widehat{L}^{\mathrm{a}}_j = (\widehat{\sigma}_j, \widehat{\rho}_j, \widehat{r}_j, \widehat{l}_j),\ j = 1, \ldots, \lvert\widehat{E}^{\mathrm{a}}\rvert$	Affine bounded application link description.
$\widehat{\sigma}_j$	$\widehat{\sigma}_j \in \mathbb{R}_{\geq 0}$	Rate bound.
$\widehat{\rho}_j$	$\widehat{\rho}_j \in \mathbb{R}_{\geq 0}$	Burst bound.
$\widehat{r}_j$	$\widehat{r}_j \in [0, 1]$	Transmission reliability.
$\widehat{l}_j$	$\widehat{l}_j \in \mathbb{R}_{\geq 0}$	Transmission delay (seconds).
$\widehat{L}^{\mathrm{s}}_j$	$\widehat{L}^{\mathrm{s}}_j = (\widehat{W}_j, \widehat{r}_j, \widehat{l}_j),\ j = 1, \ldots, \lvert\widehat{E}^{\mathrm{s}}\rvert$	Stochastically bounded application link description.
$\widehat{W}_j$	$\widehat{W}_j(x, \tau) = \Pr(\text{packets in } \tau \text{ seconds} \leq x)$	CDF of packets generated in τ seconds.
$\widehat{s}_j\ \widehat{s}_j \in \mathbb{N}_{\geq 0}$	Maximum packet size.	

Table 6.2: Notation used in the application request representation.

7 An Algorithmic Approach to Network Slice Construction

Based on the analysis of industrial networks and the abstraction models presented in the previous chapter, this chapter studies algorithms for constructing network slices. While construction of network slices is part of the NFV Orchestrator in the ETSI NFV-MANO reference model, the current implementations of the NFV-MANO mechanisms covered in Section 2.3 do not target dynamic and autonomous creation of network slices, but require an operator to manually define the network slice to be created, along with the required resources. After the creation of a network slice, the end-to-end performance can be validated by generating traffic while collecting network statistics. However, in an industrial network with very strict latency guarantees, it is difficult to assess the performance using traffic generators, due to the high impact of rare events. Instead, the lateny guarantees must be taken into account at the time the network slice is constructed.

The purpose of the algorithm proposed here is to map an application request forwarding graph onto a physical network, as well as calculating the new state (capacity, etc.) of the network. The input to the algorithm is a forwarding graph and the physical network, represented as described in the previous chapter. The output is a new resource representation of the new physical network state, as well as a description of the constructed allocation of the forwarding graph. Furthermore, it is possible that a forwarding graph cannot be mapped onto the physical network. In this case, the algorithm should report this. However, before defining the algorithm, we first give an overview of related work in the field of network virtualization.

7.1 Relation to Virtual Network Embedding

The idea of virtualizing computer networks and dividing a network into isolated slices has been around for several years as a way to provide flexibility in the network, e.g. [62–64]. In this context, algorithms for allocating communication and computation resources in a network have been considered several times in the literature. The deployment of virtual network functions and communication resources is similar to the VNE problem which refers to the task of mapping a set of virtual network requests, each consisting of virtual nodes and links, to a physical network [65, 66]. In VNE, the physical network is typically modelled as a graph where the nodes and edges have certain characteristics and constraints such as propagation delay, capacity or computational resources. Likewise, a virtual network request is a graph where the edges and nodes are subject to constraints such as maximum propagation delay, reliability or physical location [66]. It has been shown that the VNE problem is NP-hard [66] and hence much literature seeks approximation algorithms and heuristics for solving the problem. These algorithms may be designed to facilitate changes in the network (i.e. dynamic embedding) or tolerate failures in the network [66]. Other objectives include minimizing the cost of using the physical network or minimizing the energy usage.

7.1.1 Heuristic Techniques

Heuristic and metaheuristic algorithms search for solutions which are good, but not necesarily optimal. Heuristic methods for solving the VNE problem are typically divided into node allocation and edge allocation since these problems are simpler to solve individually. Simulated annealing is used to solve the the VNE problem (although referred to as the network testbed mapping problem) in [67] where the total link bandwidth is minimized. The solution space is explored by randomly moving virtual nodes in the network and then connect the nodes by shortest path. In [68] the authors consider a simple instance of the problem where edges and nodes have certain capacities specified as the number of overlays they can support. They consider continuously arriving requests and first solve the placement problem using the heuristic of placing the nodes onto the physical nodes which have both low computational load and low load on their adjacent edges. In a second step the virtual nodes are connected by the shortest path.

The solution presented in [69] uses a greedy node and link mapping where nodes are placed at the physical nodes with most free resources. The nodes are afterwards connected by the shortest path which satisfies the link requirements. Furthermore, links which are allowed to be split across multiple paths are embedded iteratively as a multi-commodity flow problem. They show that path splitting allows for much higher utilization of the network.

In [70] the node placement problem is described as a Markov Decision Process where an arriving request for l nodes is mapped to physical nodes through l actions. Only the final action leading to a successful embedding yields a non-zero reward. In order to calculate the reward, the agent must perform link mapping between the allocated nodes. Hence, in principle all possible actions must be evaluated to calculate the reward. To reduce the complexity of the problem, the authors use Monte Carlo Tree Search to randomly explore the search space. The proposed method leads to a high acceptance ratio of arriving requests at the cost of increased processing time. However, it has the advantage that the time spent on exploring the search space is a parameter to the algorithm.

7.1.2 Integer Programming and Relaxation Techniques

VNE can also be formulated as a mixed integer program. Although sophisticated algorithms for solving these programs exist, such as branch-and-bound and branch-and-cut methods, mixed integer programs are in general NP-hard and hence exact algorithms do not scale to larger problems. Nevertheless, integer programs are useful for studying problems at small scale and for finding heuristics. For instance, energy-aware VNE is studied in [71] by formulating a mixed integer program which is solved exactly to provide both insight into the involved trade-offs, and a bound which can be used for studying heuristic algorithms.

Furthermore, mixed integer programs can be used as a starting point for constructing approximation algorithms by relaxing the integer constraints. This way, a new optimization problem is defined which can be solved efficiently. However, the solution to the relaxed problem is not necessarily an optimal or feasible solution to the initial problem. Instead, it is anticipated that it is close to the optimal (or at least a near-optimal) solution in the solution space, and that a near-optimal feasible solution can be obtained by proper rounding of the variables [72]. One advantage of integer

relaxation techniques is that they can jointly solve the node placement and edge allocation problems [66]. Moreover, it is straightforward to include additional (convex) constraints to the integer program, and the objective can be chosen almost arbitrarily as long as it is convex. Another advantage of integer relaxation techniques is that it may be possible to formally derive a bound on the closeness of the approximate solution.

Depending on the problem, the relaxed problem may have a solution which is very far from the solution to the initial problem, and it may be difficult to find a valid and good rounding procedure [72]. Nevertheless, in case no rounding can be found, the relaxed problem provides a bound on the optimal solution to the initial problem, which is useful when evaluating approximations or heuristics.

Two algorithms based on integer programming relaxation based on deterministic and random rounding are presented in [73], where the authors consider continuously arriving virtual network requests and attempt to maximize the acceptance ratio of new requests. In [74] an integer linear program is constructed for solving the VNE problem while providing spare capacity between nodes to increase reliability. A greedy approach to solving the problem is presented which still provides the desired spare capacity.

7.1.3 Delay-Aware Techniques

The approaches mentioned above mainly considers capacity related constraints. However, in some scenarios, in particular in industrial networks, latency may also be of interest. Including latency constraints to the VNE problem is nontrivial and depends on the technologies used in the physical network. Some physical networks may provide latency guarantees, in which case including latency constraints are straightforward, but in other cases the network includes buffering or prioritization, which complicates the latency model. Specifically, in such case the amount of latency depends on the traffic characteristics which are typically random and nonlinear, and cannot be captured by the concept of capacity. Therefore, the literature on VNE with delay constraints is limited. In [75] transmission and processing delays are treated as constants associated to each edge and node. A technique which uses queuing theory to estimate the average latency is proposed in [76]. Specifically, they consider M/M/1 queues and estimate the utilization as the fraction between allocated and available link capacity and the service time as the mean packet length divided by link capacity. The authors do not attempt to solve the VNE problem with the proposed delay function, and this may not be straightforward since it requires optimizing over a non-linear function with integer constraints. Furthermore, the mean latency as well as the assumption on Poisson arrivals and exponential service time may not be sufficient for latency-critical systems. The same approach is taken in [77] where the resulting delay function is approximated by a piecewise linear function to simplify the problem description. They use the proposed function to formulate a convex optimization problem which calculates the amount of data to send through different paths in the network, but do not consider the placement of network functions. In [78] the delays in a network are modelled as Jackson queuing networks and an algorithm for embedding virtual networks under latency constraints is proposed. However, like the previous work they assume exponential arrivals and service, and only the mean latency.

7.2 Challenges in the Industrial Domain

While the VNE problem is very similar, and perhaps even identical, to the problem of mapping a forwarding graph onto a physical network, there are some important limitations of the current approaches. For instance, the resources in the physical networks and the forwarding graphs are described by "capacities" which refer to an abstract measure. For instance, an edge in the physical has a certain capacity, and an edge in the forwarding graph requires/occupies a certain capacity. While a capacity measure may be suitable for describing communication resources in some cases, it does not take traffic patterns into account, and hence is not suitable for applications that require strict latency guarantees.

A problem with the delay-aware techniques is the traffic is assumed to be Poisson distributed. While this may be reasonable when many flows are multiplexed on the same link, it does not provide a good model for traffic close to the edge of a network. More importantly, a Poisson model is very likely to underestimate the burstiness of the arrivals, which means that the calculated delay underestimates the actual experienced delay. Furthermore, the deterministic industrial protocols are not very well modelled by M/M/1 queues.

The reason why latency constraints are difficult to include in the VNE problem is that while latency is additive (the end-to-end latency is the sum of latencies at each link), the queuing delay depends on how traffic arrives to the queue. The traffic arrival depends again on the previous traversed queues as well as the application generating the traffic. Hence, all paths have to be evaluated in order to find the path with shortest delay. Moreover, the output of a queue depends on the other traffic sharing the queue. This means that there is limited amount of isolation in the system since the guarantees given to a flow which has already been allocated will be influenced by any new flows allocated to the same queue.

7.3 Algorithm Development

Based on the observations discussed in the previous section, this section constructs a heuristic based algorithm for allocating network slices which provide strict latency guarantees. The algorithm is based on the overall conclusions from the analysis in Section 5.3. Specifically, the cell network and the backend network are analyzed separately, and while the cell network allows for probabilistic arrivals, the delays in the backend network is modelled using deterministic network calculus and hence requires affine bounded arrivals. To limit the scope of the algorithm, we make the following assumptions:

Assumption 1: VNFs have already been allocated. We assume that the locations of VNFs are fixed so that the task is to allocate communication resources. While this assumption means that the VNFs cannot be placed to facilitate the allocation of communication resources, it is motivated by the fact that most VNE algorithms are divided into node placement and communication resource allocation.

Assumption 2: Cell network nodes have degree one and form an independent set. The communication resource allocation algorithm assumes that each cell node only has one link which connects the node to the backend network, and this link

is a cyclic resource link. This means that communication between cell devices is not possible.

Assumption 3: Backend network nodes are connected by acyclic resource links. To analyze the cell network and the backend network separately, the backend network must not contain cyclic resource links.

Assumption 4: Backend traffic is upper bounded. We assume that communication requirements between two backend nodes in the forwarding graph are specified as an affine function which provides an upper bound on the generated traffic.

Assumption 5: Multiplexed acyclic resource links are capacity constrained. Since the queuing delay experienced by a flow depends on all other flows sharing the same queue, it is impossible to provide end-to-end latency guarantees unless the arrival to each queue is bounded. We assume that the ingress queue to link j has a predefined constraint to the aggregate arrival rate so that $\rho_j^{\text{agg}} \leq \Gamma_j^{\text{ac}}$ and $\sigma_j^{\text{agg}} \leq \Sigma_j^{\text{ac}}$.

Assumption 2 allows the algorithm to be divided a cell network allocation and a backend allocation. As shown in Algorithm 1, the cell network allocation is executed first, and based on the guarantees provided by the cell network, the forwarding graph is reduced to only include the backend network. Specfically, the cell nodes are replaced by their adjacent backend nodes, and the delay and reliability requirements are updated according to the delay and reliability introduced in the cell network. Lastly, an upper bound on the arrival rate is calculated based on the number of resources allocated in each cycle at the cell network. Resources are then allocated in the reduced backend network, and finally the reduced forwarding graph is expanded again to include the cell network.

Algorithm 1 Resource allocation algorithm.

```
1: procedure Allocation(G,F̂)
2:     G, F̂ = CellAllocation(G, F̂) or fail
3:     F̂′ = ReduceForwardingGraph(F̂)
4:     G, F̂′ = BackendAllocation(G, F̂′) or fail
5:     F̂ = ExpandForwardingGraph(F, F̂′)
6:     return (G, F̂)
7: end procedure
```

7.3.1 Cell Network Allocation

The cell network allocation algorithm (Algorithm 2) allocates cyclic resources in the cell network. It does so by considering the edges in the forwarding graph which have a cell node as an endpoint, and then allocate the resources at the edge connecting the cell node to the backend node. By assumption 2, there is only one such edge. The edge is allocated by the AllocSingleEdge procedure which calculates the number of resources needed to provide the required reliability. If the resources are available, then the resulting reliability is stored in the forwarding graph edge as $f.r'$ and the path and delay are stored in $f.p'$ and $f.d'$, respectively. Finally, the capacity is of the resource link s_e^{sc} is reduced.

Algorithm 2 Cell network resource allocation algorithm.

```
1:  procedure CellAllocation(G,F̂)                    ▷ Resource graph G, forwarding graph F
2:      for f ∈ F̂.Ê do                               ▷ For each edge in forwarding graph
3:          u = F̂.b̂(f).u                             ▷ Source node
4:          if u ∈ G.E^c then                        ▷ If edge connects a cell node
5:              e = G.b^{-1}(u, ·)                   ▷ u is only connected to one edge, e, (assumption 2)
6:              AllocSingleEdge(G,e, f) or fail
7:      return (G, F̂)
8:  end procedure
9:  procedure AllocSingleEdge(G,e,f)                 ▷ Resource link e, application edge f
10:     r = ⌈Ŵ_f^{-1}(r̂_f)⌉ · ŝ_f                    ▷ Number of bytes to allocate
11:     if s_e^c < r then
12:         fail                                     ▷ Not sufficient resources
13:     f.r' = Ŵ_f(r)                               ▷ Resulting reliability
14:     f.p' = {e}                                  ▷ Set edge as initial path
15:     f.d' = l_e^c                                ▷ Set delay
16:     s_e^c = s_e^c − r                           ▷ Calculate remaining resources
17: end procedure
```

7.3.2 Backend Network Allocation

After resources have been allocated in the cell network and the forwarding graph has been reduced to only include backend network requirements, the backend network resources can be allocated. As shown in Algorithm 3, the overall heuristic in this allocation is to allocate resources along the path between two nodes in the forwarding graph with lowest delay. If the path with lowest delay satisfies the delay and reliability requirements, then a network slice satisfying the requirements can be allocated along the path. The FindPath procedure calculates three properties of the path from u to v: The delay, denoted $v.d$, the reliability, $v.r$ and the predecessor edge $v.\pi$. The path from u to v can be obtained by recursively following the predecessor edges until the source node u is reached. This is done by the ExtractPath procedure. After the path has been found, the path attributes are stored in f and the capacity required at each edge is subtracted from the representation graph G.

Algorithm 3 Backend network resource allocation algorithm.

```
1:  procedure BackendAllocation(G,F̂)                 ▷ Resource graph G, forwarding graph F
2:      for f ∈ F̂.Ê do                               ▷ For each edge in forwarding graph
3:          u, v = F̂.b̂(f)                            ▷ Source and destination nodes
4:          G' = (G.V, G.E^ac, ∅, b)                 ▷ Construct new graph without cyclic edges
5:          FindPath(G, f, u, v)                     ▷ Calculate path from u to v
6:          if v.d > d̂_f or v.r > r̂_f then
7:              fail                                 ▷ Not fulfilling delay and reliability requirements
8:          f.d' = f.d' + v.d                        ▷ Add delay
9:          f.r' = f.r' · v.r                        ▷ Resulting reliability
10:         p = ExtractPath(G', u, v)                ▷ Extract path from predecessor list
11:         f.p' = f.p' ∪ p                          ▷ Add backend path to total path
12:         G = SubtractPathCapacity(G, u, v)        ▷ Calculate residual capacity
13:     return (G, F̂)
14: end procedure
15: procedure ExtractPath(u, v)                      ▷ Source node u, destination node v
16:     if u = v then
17:         return {u}
18:     else
19:         p = {v} ∪ ExtractPath(u, v.π)
20:         return p
21: end procedure
```

However, as discussed previously, the queuing delay at each node depends on the

nodes that have previously been traversed in the path, which complicates the search for the path with lowest delay using shortest path algorithms. Nevertheless, shortest path algorithms can still be used to provide a good heuristic for finding a short path through the network.

Consider a flow with affine bounded arrival curve $A(t) = \sigma t + \rho$. The minimum service experienced by the flow on link L_n^{ac} is given by the leftover service from queues on the same link with higher priority, and the flows sharing the same queue. Let $\mathcal{J}$ denote the set of edges multiplexed with edge L_n^{ac}, i.e.

$$\mathcal{J} = \{L_k^{\text{ac}} \in \mathcal{M}_j \, \forall \, j \mid L_n^{\text{ac}} \in \mathcal{M}_j\}. \tag{7.1}$$

Since the burst at each edge is limited by Γ_j^{ac} (assumption 5), the leftover burst given from the edges in $\mathcal{J}$ is the sum of Γ_j^{ac} over the edges with equal or higher priority than L_n^{ac}

$$\rho_n^{\text{lo}} = \sum_{j \in \mathcal{J} : \epsilon_j^{\text{ac}} \leq \epsilon_n^{\text{ac}}} \Gamma_j^{\text{ac}}. \tag{7.2}$$

Since the flow on link L_n^{ac} is included in Γ_n^{ac} the worst-case leftover burst given to the flow is

$$\rho^{\text{lo}} = \rho_n^{\text{lo}} - \rho. \tag{7.3}$$

The leftover service from the edges in $\mathcal{J}$ is similarly given by

$$\begin{aligned} \sigma_n^{\text{lo}} &= \sum_{j \in \mathcal{J}} \Sigma_j^{\text{ac}} - \sum_{j \in \mathcal{J} : \epsilon_j^{\text{ac}} \leq \epsilon_n^{\text{ac}}} \Sigma_j^{\text{ac}} \\ &= \sum_{j \in \mathcal{J} : \epsilon_j^{\text{ac}} > \epsilon_n^{\text{ac}}} \Sigma_j^{\text{ac}}, \end{aligned} \tag{7.4}$$

and the leftover service given to the flow on link L_n^{ac} is

$$\sigma^{\text{lo}} = \sigma_n^{\text{lo}} + \sigma, \tag{7.5}$$

where σ is the capacity occupied by the flow. The worst-case queuing delay experienced at edge L_n^{ac} is then given by

$$\begin{aligned} d_n^{\text{q}} &= \frac{\rho^{\text{lo}} + \rho}{\sigma^{\text{lo}}} \\ &= \frac{\rho_n^{\text{lo}}}{\sigma_n^{\text{lo}} + \sigma}. \end{aligned} \tag{7.6}$$

It follows that σ is the only flow-dependent parameter of d_n^{q}. Nevertheless, it is required to keep track of both ρ and σ since they determine the capacity that the flow occupies as well as the output flow. The output flow of the queue is parameterized by

$$\rho^* = \rho + \frac{\sigma \rho^{\text{lo}}}{\sigma^{\text{lo}}}, \tag{7.7}$$

$$\sigma^* = \sigma. \tag{7.8}$$

The analysis above provides an important requirement to the shortest path algorithm that should be used for finding a path through the network. Since the analysis strongly relies on the fact that each queue do not occupy more service than the al-

located capacity for the queue, it must be able to take the capacity constraints into account. As the arrival parameters ρ and σ change through the path, the capacity required at node n is unknown until the path $1, 2, \ldots, n-1$ is fixed. One algorithm that allows for taking the capcity constraints into account is Dijkstra's shortest path algorithm [79]. The reason for this is that Dijkstra's algorithm never investigates edges from vertices with an undetermined shortest path. In other words, the length of an edge $k \to l$ is only evaluated if the shortest path from the source node to k is known. Consequently, the shortest path up to the edge being considered is fixed, and hence the output flow is known. Note that Dijkstra's algorithm only is guaranteed to find the shortest path when edge lengths are independent, which is not the case for the queuing delays as described above. However, it provides a good heuristic for finding a path with a low delay.

The FindPath procedure based on Dijkstra's algorithm is shown in Algorithm 4, and consists of three procedures. The FindPath procedure is equivalent to the regular procedure in Dijkstra's algorithm as described in [79], with the addition that the algorithm is terminated as soon as the destination vertex is reached, and does not continue until the shortest paths to all nodes have been found. The Relax procedure is called when a new shortest path to a vertex, u, has been found, and updates the distance to the vertices adjacent to u. Compared to the regular Relax procedure in Dijkstra's algorithm, this procedure also updates the reliability and arrival model associated with the new shortest path so that this information is propagated through the graph. Finally, the Dist procedure simply calculates the delay experienced along an edge based on the arrival model as described above. If an edge has insufficient capacity for a flow, then the distance is set to infinity. Note that it is assumed that ρ_e^{lo} and σ_e^{lo} have been pre-calculated based on (7.2) and (7.4).

7.4 Algorithm Evaluation

To evaluate the performance of the presented algorithm, a scenario comprising 10 cells and a backend network is considered (Figure 7.1). Each cell network consists of 5 cell devices which are all connected to the cell master node (c1-c10). Hence, the network contains a total of 68 nodes. The connection between the cell devices and the cell master is cyclic with period 1 ms, and each cell device has 512 bytes available for network slices. All parameters are listed in Table 7.1.

Each backend node contains a number of prioritized queues N. It is assumed that each node in the backend network has a buffer capacity of 2^{20} bytes, which are allocated to the individual queues exponentially so that the queue with priority $n \in [1, N]$ receives $0.5^{N-n+1}2^{20}$ bytes. The allocated buffer size defines the queue burst capacity Γ_j^{ac}. Furthermore, the prioritization queues equally share the available data rate so that $\Sigma_j^{\text{ac}} = 112.5 \cdot 10^6/N$.

The algorithm is evaluated by randomly generating forwarding graphs and monitoring the number of forwarding graphs that can be mapped successfully onto the physical network. Forwarding graphs are generated by randomly selecting a cell node, and then doing a random walk by randomly selecting the next node in the backend network. The forwarding graph is completed after $L \sim \mathit{Uniform}(2, 5)$ steps, and hence the final forwarding graph contains the cell node and between 2 and 5 backend nodes. The requirements to the edges in the forwarding graph are generated according to the parameters listed in Table 7.1.

Algorithm 4 FindPath procedure.

procedure FindPath(G, f, u, v) ▷ Resource graph G, application edge f, source and destination nodes u, v
  for $v \in G.V$ **do** ▷ For each vertex in G
    $v.\pi$ = NIL ▷ Initialize predecessor
    $v.d = \infty$ ▷ Initialize distance list
  $u.d = 0$ ▷ Distance to source
  $u.\sigma = \widehat{\sigma}_f$ ▷ Initial σ
  $u.\rho = \widehat{\rho}_f$ ▷ Initial ρ
  $Q = G.V$ ▷ Initialize vertex list
  while $Q \neq \varnothing$ **do**
    u =ExtractMin(Q) ▷ Extract vertex with shortest distance to source
    if $u = v$ **then** ▷ Destination found
      terminate
    for $e \in G.Adj[u]$ **do** ▷ For each edge adjacent to u
      $k = dst(e)$ ▷ Destination node
      Relax(G, u, k, e) ▷ Update distances
end procedure
procedure Relax(G, u, k, e) ▷ Resource graph G, edge e connecting u and k
  d' = Dist(G, u, k, e) ▷ Distance to k through e
  if $k.d > u.d + d'$ **then** ▷ New shortest path from u to k
    $k.d = u.d + d'$ ▷ Update distance
    $k.r = u.r \cdot r_e^{\text{ac}}$ ▷ Update reliability
    $k.\pi = e$ ▷ Update predecessor edge
    $k.\rho = e.\rho + \frac{e.\sigma(\rho_e^{\text{lo}} - e.\rho)}{\sigma_e^{\text{lo}} + e.\sigma}$ ▷ Update arrival model
    $k.\sigma = e.\sigma$
end procedure
procedure Dist(G, u, k, e) ▷ Resource graph G, edge e connecting u to k
  if $\Gamma_e^{\text{ac}} < u.\rho$ **or** $\Sigma_e^{\text{ac}} < u.\sigma$ **then**
    return ∞ ▷ Not enough capacity
  $d_e^{\text{q}} = \frac{\rho_e^{\text{lo}}}{\sigma_e^{\text{lo}} + u.\sigma}$ ▷ Worst-case queuing delay
  return $d_e^{\text{q}} + l_e^{\text{ac}}$ ▷ Delay including transmission time
end procedure

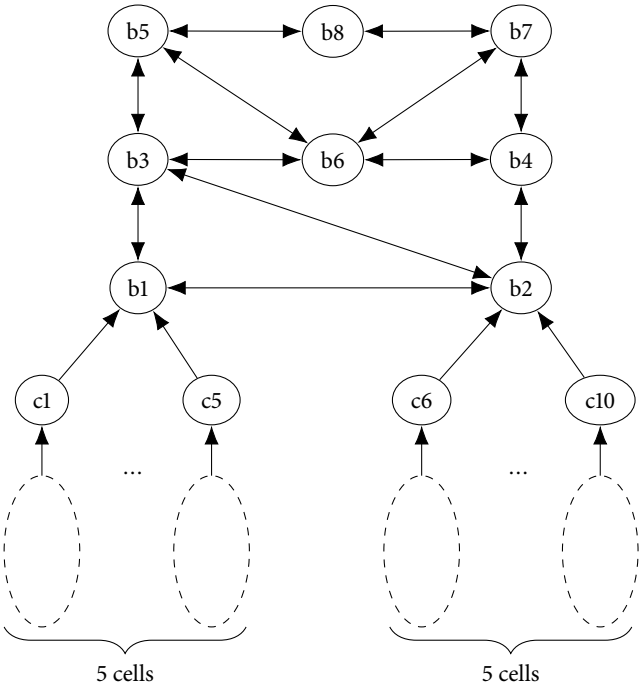


Figure 7.1: The network considered in the algorithm evaluation.

Parameter	Value
Network	
Data per cell device	512 bytes/cycle
Cycle time	1 ms
Backend data rate	$125 \cdot 10^6$ bytes/s
Link PER	10^{-11}
Edge transmission delay	100 ns
Backend transmission delay	500 ns
Total buffer size	2.0^{20} bytes
Number of queues per link (N)	3,5,10,20 bytes
Queue n burst/buffer capacity (Γ_j^{ac})	$2.0^{20} \cdot 0.5^{N-n+1}$ bytes
Queue rate capacity (Σ_j^{ac})	$125 \cdot 10^6/N$ bytes
Forwarding Graph	
Number of backend nodes	Uniform$(2, 5)$
Cell device packet distribution	Poisson
Size of cell device packets	Uniform$(8, 128)$
Rate of cell device packets	Uniform$(10^{-3}, 100)$ packets/s
Backend traffic ρ parameter	1500 bytes
Backend traffic σ parameter	Uniform$(1500, 10^6)$ bytes/s
Maximum PER	Uniform$(10^{-9}, 10^{-2})$
Maximum delay	Uniform$(5^{-3}, 0.1)$ s

Table 7.1: Parameters used in the algorithm evaluation.

The resulting cumulative number of acceptances are shown in Figure 7.2a for a various number of queues. The results are obtained by generating 1000 forwarding graphs (1000 evolutions), and mapping them onto the network. If the mapping succeeds, then the capacity of the network is reduced accordingly in the following evolutions. The experiment is repeated 100 times and the average number of acceptances at a given evolution is calculated. Initially, most of the forwarding graphs are accepted, but as the number of evolutions increases, the number of accepted forwarding graphs decreases due to the lower capacity in the network. Furthermore, the figure reveals a trade-off between the number of queues and the number of accepted forwarding graphs. Specifically, the scenario with 10 queues per link performs best, while the scenarios with 3 and 100 queues perform worst. This is further illustrated in Figure 7.2b which shows the acceptance ratios for the different number of queues after 1000 evolutions. The reason for this trade-off is that the total amount of buffer capacity allocated to the queues is the same, and hence links with a high number of queues contains many queues with low capacity which cannot serve many flows, if any. On the other hand, links with few queues cannot guarantee very low latencies. Hence, the system is delay limited when the number of queues is low and capacity limited when the number of queues is high.

Figure 7.2c shows the reason for forwarding graph rejections for the case with 10 queues. Initially, the rejections are due to requirements that cannot be satisfied, but as the number of evolutions increases, a higher percentage of the rejections are caused by capacity limits. Furthermore, as shown in Figure 7.2d the capacity limit is mainly caused by the cell network, while the edges in the backend network (Σ_e^{ac} and Γ_e^{ac}) have much available capacity. Therefore, improving the cell network allocation, e.g. using

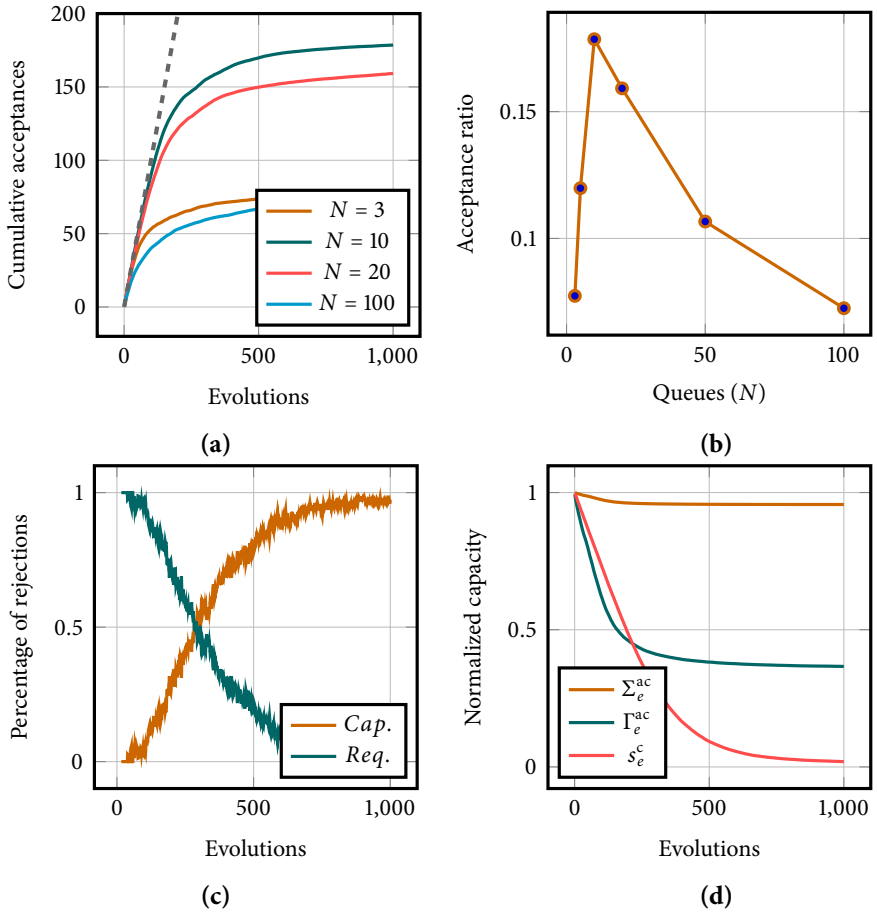


Figure 7.2: Algorithm evaluation results. *(a)* Cumulative acceptance vs. evolutions for different number of queues N. The grey dashed line indicates the case where all forwarding graphs are accepted. *(b)* Acceptance ratio after 1000 evolutions. *(c)* Percentage of rejections caused by capacity and requirement violations for $N = 10$. *(d)* Normalized mean capacity of the edges in the network. Σ_e^{ac} and Γ_e^{ac} are calculated from backend edges while s_e^c is the cell edge capacity.

the multiplexing schemes considered in Section 5.1, is likely to increase the number of forwarding graphs that can be mapped onto the network.

8 Discussion

This report concerns network slicing as an enabler for the various and dynamically changing network requirements expected in Industry 4.0. It investigates network calculus as a potential method for analyzing end-to-end properties in heterogeneous networks, and considers various strategies for slicing industrial communication protocols. Furthermore, it defines an abstract representation of industrial networks which allows for decoupling the allocation of network slices from the underlying technologies. Based on this abstraction model, an algorithm that demonstrates how network calculus can be used for constructing network slices with end-to-end latency and reliability guarantees is proposed.

8.1 End-to-End Analysis

A fundamental challenge of providing network slices in Industry 4.0 is the analysis of end-to-end properties in the networks. While the strict latency requirements in industrial networks have traditionally been handled by the use of deterministic master/slave communication protocols, Industry 4.0 is likely to introduce switched Ethernet and queuing based technologies, e.g. as a means to connect the manufacturing cells to the cloud. Although providing end-to-end QoS and analyzing end-to-end latency in switched networks have been investigated for several decades, the methods are often based on strong assumption and subject to a high degree of uncertainty, or very conservative. In the context of industrial networks, where safety is often a concern, conservative requirements are strongly favored over uncertainty. We study deterministic and stochastic network calculus as frameworks for obtaining bounds on the end-to-end delays in queuing networks. While deterministic network calculus is limited to analyzing worst-case delays in networks where upper and lower bounds on arrivals and servers are known, stochastic network calculus provides bounds on the percentiles of the delay distributions in networks with a wide range of random arrivals. However, industrial networks contain a mixture of both random and deterministic traffic, and this significantly complicates the analysis. In particular, in order to use deterministic network calculus, an upper bound on the arrivals must be defined which may not be possible for random processes, while it is nontrivial to describe deterministic arrivals, such as cyclic traffic, using stochastic network calculus. The specific scenario of interest determines whether one framework is more suitable than the other. The cyclic traffic dominating industrial communication technologies motivate for using deterministic network calculus in the cell networks, while stochastic network calculus is suitable in cloud networks with a high degree of statistical multiplexing and aggregate flows.

However, while deterministic and stochastic network calculus provide simple frameworks for analyzing end-to-end properties of a network, they are for several reasons not ideal. For instance, it can be very difficult to define bounds on the flows, especially if they are generated by high-layer applications and protocols. Furthermore, many high-layer protocols, most notably TCP, provides feedback between the sender and receiver, which influences the arrival rate of the application. While this may not be a big problem in current industrial networks with cyclic traffic and small amounts of data, it poses a challenge if the strict latency requirements extend to backend or cloud

networks where transport layer protocols are predominant.

8.2 Slicing Industrial Protocols

While switched networks are likely to be used for the backend network, the edge of industrial networks comprises communication technologies that are specific for industrial networks and designed to provide very high reliability with reserved data slots and deterministic latency guarantees. However, data slots reserved to individual slaves provide limited flexibility, and with an increased number of devices and the introduction of wireless technologies and dynamically changing end-to-end requirements, the ability to do multiplexing in the industrial edge network is required to maintain high utilization. This report considers two approaches for applying resource multiplexing to industrial communication technologies: By introducing a gateway and by allowing users to overwrite data from other users. In the gateway multiplexing scheme, a gateway has a number of reserved data slots which are shared by the users of the gateway. This scheme is simple to realize, but requires a gateway which is not always practical and may decrease the system reliability. However, in case of e.g. wireless sensors which transmit data randomly, the wireless receiver could act as a gateway to avoid reserving resources for each of the individual sensors. In the other multiplexing scheme, the devices are allowed to overwrite certain data slots which are allocated for other devices (i.e. replace the content). For instance, a device which sends error reportings very rarely but requires very high reliability could be allowed to overwrite cyclic control traffic. Since this scheme does not require reservation of resources, but it comes at the cost of decreased reliability of the overwritten application, it is a very efficient way of handling rare arrivals. Another use case of this multiplexing scheme could be to use overwriting for random access signaling, where devices, e.g. after being moved, can trigger the creation of a dedicated network slice. This would remove the need for reserving resources for this purpose, and hence increase the utilization of the network, but comes at the cost of a decreased isolation level. The overwriting scheme may be challenging to implement in practice since the communication resources in industrial protocols typically are supposed to be isolated, and hence overwriting resources may trigger a failure detection mechanism. Furthermore, the behaviour depends on the order in which packets pass through the network, which constraints which resources that a device can overwrite.

8.3 Abstract Representation

Since industrial networks may comprise many different technologies, realizing network slicing requires an abstract representation of the physical network that can be used in the process of constructing slices and allocating the required resources in the network. To this end, the representation needs to cover both computation and communication resources. There are many parameters that need to be taken into account in order to determine whether a VNF can be deployed on a machine, and an adequate representation is likely to become very comprehensive, as is the case for the ETSI specification [61]. While such a representation is required to validate that a VNF can be instantiated on a node, constructing a slice allocation algorithm that can handle such a complex representation with a very high number of constraints is very challenging. Therefore, there is a need for a simpler representation which captures the most im-

portant requirements, and can be used as input to an algorithm. This report defines an abstract representation of a physical network which attempts to solve this problem, with focus on the characteristics that are specific to industrial networks. The representation consists of a directed multigraph where vertices represent physical devices and servers, and edges represent communication technologies with certain characteristics such as cycle time, data rate, etc. Compared to the specification from ETSI [61] which defines link parameters such as delay and data rate, the representation presented in this report does not consider links as black boxes, but rather allows for analyzing queuing properties based on the traffic that is allocated to the links. This is required to properly describe the characteristics of industrial communication technologies, and allows for a higher degree of freedom when allocating communication resources. However, while the representation may contain the parameters that intuitively seem most important, defining a good representation requires an iterative approach which includes instantiation on actual physical systems. In particular, it is not sufficient to verify that the representation works well in an allocation algorithm, but also that it captures a sufficient level of detail to facilitate a deployment of the network functions. Hence, the representation presented in this report is only the initial step towards defining a good and adequate representation.

8.4 Slice Construction Algorithm

To illustrate how network calculus and the defined abstraction model can be used to construct network slices, a heuristic-based algorithm for slicing an industrial network is presented. It is assumed that VNFs have already been placed in the network, and hence the task is to interconnect the VNFs and the edge devices according to a forwarding graph in such a way that the requirements to the communication are fulfilled. The allocation algorithm is divided into two steps: Allocating cell network resources and allocating backend network resources. Cell network resources are allocated by reserving the minimum number of resources required to satisfy the given reliability constraint, while the algorithm attempts to allocate backend network resources along the path that minimizes the delay. However, since the experienced queuing delay depends on the arrival pattern at each queue, which in turn depends on the previously traversed queues in the path, finding the optimal shortest path is hard. Instead, the algorithm uses Dijkstra's shortest path algorithm as a heuristic for finding the path with minimum delay. However, since the recurrence relation that Dijkstra's algorithm exploits does not hold for this scenario, it cannot be guaranteed that the optimal path is found. This also means that the algorithm is not guaranteed to find a path that satisfies the requirements, even if such a path exists in the network.

An evaluation of the proposed algorithm reveals an interesting trade-off between the number of queues used per link, and the number of forwarding graphs that can be accepted. In particular, when a link has few queues, it is difficult to provide sufficiently low latency, while with a high number of queues the capacity of each queue becomes the limiting factor. Therefore, the ideal number of queues depends on the burstiness of the arrivals and the delay requirements. Furthermore, the edge network turns out to be the bottleneck in the considered scenario. While this depends on the specific traffic characteristics used in the evaluation, it suggests that the number of forwarding graphs that can be accepted could be significantly increased by improving the edge allocation. This could be done by applying different multiplexing schemes, such as

allowing for resource overwriting.

The algorithm and the physical network have many parameters which may influence the performance and the results. These parameters are not well explored in this report, and a more thorough evaluation is needed to understand how these impact the performance of the algorithm and the characteristics of the resulting network slice allocations. While the algorithm has many assumptions, it demonstrates how network calculus can be integrated into an algorithm for constructing network slices with end-to-end latency and reliability guarantees. However, it can be improved in several ways. First, network slices are allocated to minimize the delay between two nodes which may result in a slice that provides a much lower delay than needed. This occupies capacity that could be used by applications that need the low latency. A better approach would be to allocate a slice which guarantees the highest possible delay while satisfying the requirement. However, this is not straightforward to do since the end-to-end delay of a path is unknown until the destination node has been reached. Another thing that could be considered is the placement of nodes. Since joint optimization is likely to result in a higher acceptance ratio since more degrees of freedom are exploited. However, a static allocation may be sufficient for some functionality, such as caches, which can be statically deployed close to the edge of the network.

9 Conclusion

This report investigates end-to-end network slicing for Industry 4.0, which introduces wireless communications, cloud computing, virtualization, and other emerging information and communication technologies to industrial manufacturing systems. Network slicing refers to the process of slicing a physical network into sub-networks with certain characteristics such as ultra low latency or high data rate, and has been investigated in the context of 5G cellular systems as a technology for supporting heterogeneous application requirements. The applications in Industry 4.0 are expected to have similar heterogeneous requirements to the network, and hence network slicing is also a enabler in this context.

An important part of providing end-to-end network slices is to analyze end-to-end properties in a network. We show that deterministic and stochastic network calculus provide frameworks for analyzing worst-case and probabilistic bounds on end-to-end latencies in queuing networks, respectively, as long as the traffic does not contain a mixture of deterministic and stochastic arrival processes. Since industrial communication technologies often impose determinism, deterministic network calculus is an applicable framework for analyzing end-to-end guarantees close to the cells, while stochastic network calculus is suitable for cloud networks with a high degree of statistical multiplexing and aggregate flows. We also propose three slicing schemes for cyclic industrial communication technologies that facilitate non-deterministic traffic. Each scheme provides different characteristics in terms of utilization, reliability and isolation, and allows for satisfying heterogeneous application requirements while effectively utilizing the communication resources.

The introduction of wireless communication means that the end-to-end requirements are likely to change over time. Considering the strict end-to-end requirements, it is desired to adapt the network slices to new situations as fast as possible. Therefore, another significant part of slicing industrial networks is to automatically construct network slices that satisfy the given end-to-end requirements. In this thesis, we define a simple abstract representation for describing physical industrial networks, which can be used as input to a slicing algorithm. Furthermore, we construct a basic algorithm that demonstrates how network slices with strict end-to-end guarantees can be created based on the abstract representation and deterministic network calculus. While the algorithm is based on several assumptions and requires a more thorough performance evaluation, it demonstrates how the construction of network slices with very strict end-to-end requirements can be automated. Finally, it reveals an interesting trade-off between the number of accepted network slice requests and the number of queues used in a link, suggesting that the optimal number of queues depends on the characteristics of the traffic and delay requirements.

Bibliography

[1] Rainer Drath and Alexander Horch. "Industrie 4.0: Hit or hype?" In: *IEEE industrial electronics magazine* 8.2 (2014), pp. 56–58.

[2] Nasser Jazdi. "Cyber physical systems in the context of Industry 4.0". In: *IEEE International Conference on Automation, Quality and Testing, Robotics*. 2014, pp. 1–4.

[3] Malte Brettel et al. "How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective". In: *International Journal of Mechanical, Industrial Science and Engineering* 8.1 (2014), pp. 37–44.

[4] Jay Lee, Behrad Bagheri, and Hung-An Kao. "A cyber-physical systems architecture for industry 4.0-based manufacturing systems". In: *Manufacturing Letters* 3 (2015), pp. 18–23.

[5] Jean-Dominique Decotignie and Patrick Pleinevaux. "A survey on industrial communication networks". In: *Annals of Telecommunications* 48.9 (1993), pp. 435–448.

[6] Xun Xu. "From cloud computing to cloud manufacturing". In: *Robotics and computer-integrated manufacturing* 28.1 (2012), pp. 75–86.

[7] Peter Mell, Tim Grance, et al. "The NIST definition of cloud computing". In: (2011).

[8] Michael Armbrust et al. "A view of cloud computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[9] Andreas Frotzscher et al. "Requirements and current solutions of wireless communication in industrial automation". In: *Communications Workshops (ICC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 67–72.

[10] 3GPP. *Study on Architecture for Next Generation System*. TR 23.799 v14.0.0. 3rd Generation Partnership Project (3GPP), Dec. 2016.

[11] NGMN Alliance. *5G White Paper*. White paper. Next Generation Mobile Networks (NGMN) Alliance, Feb. 2015.

[12] NGNM Alliance. *5G Network and Service Management including Orchestration*. 2017.

[13] P Popovski et al. "ICT-317669-METIS/D6.2 Initial report on horizontal topics, first results and 5G system concept". In: *EU-Project METIS (ICT-317669), Deliverable* (2014).

[14] ETSI. *Network Function Virtualization—Network Operator Perspectives on NFV Priorities for 5G*. White paper. European Telecommunications Standards Institute (ETSI), Feb. 2017.

[15] Chengchao Liang and F Richard Yu. "Wireless network virtualization: A survey, some research issues and challenges". In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 358–380.

[16] Diego Kreutz et al. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[17] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The road to SDN". In: *Queue* 11.12 (2013), p. 20.

[18] ONF. *SDN Architecture*. TR 521 issue 1.0. Open Networking Foundation (ONF), 2014.

[19] ONF. *SDN Architecture*. TR 521 issue 1.1. Open Networking Foundation (ONF), 2016.

[20] ONF. *Applying SDN Architecture to 5G Slicing*. TR 526 issue 1.0. Open Networking Foundation (ONF), Apr. 2016.

[21] Jonathan van de Belt, Hamed Ahmadi, and Linda E. Doyle. "Defining and Surveying Wireless Link and Network Virtualization". In: *arXiv preprint arXiv:1705.03768* (2017).

[22] James E Smith and Ravi Nair. "The architecture of virtual machines". In: *Computer* 38.5 (2005), pp. 32–38.

[23] Stephen Soltesz et al. "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors". In: *ACM SIGOPS Operating Systems Review*. Vol. 41. 3. ACM. 2007, pp. 275–287.

[24] Alexandra Aguiar and Fabiano Hessel. "Embedded systems' virtualization: The next challenge?" In: *Rapid System prototyping (RSP), 2010 21st IEEE International symposium on*. IEEE. 2010, pp. 1–7.

[25] Gernot Heiser. "The role of virtualization in embedded systems". In: *Proceedings of the 1st workshop on Isolation and integration in embedded systems*. ACM. 2008, pp. 11–16.

[26] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. "Challenges in real-time virtualization and predictable cloud computing". In: *Journal of Systems Architecture* 60.9 (2014), pp. 726–740.

[27] Sisu Xi et al. "Rt-xen: Towards real-time hypervisor scheduling in xen". In: *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*. IEEE. 2011, pp. 39–48.

[28] Sisu Xi et al. "Real-time multi-core virtual machine scheduling in xen". In: *Embedded Software (EMSOFT), 2014 International Conference on*. IEEE. 2014, pp. 1–10.

[29] Geoffrey Phi C Tran et al. "Hypervisor performance analysis for real-time workloads". In: *High Performance Extreme Computing Conference (HPEC), 2016 IEEE*. IEEE. 2016, pp. 1–7.

[30] Alfons Crespo, Ismael Ripoll, and Miguel Masmano. "Partitioned embedded architecture based on hypervisor: The xtratum approach". In: *Dependable Computing Conference (EDCC), 2010 European*. IEEE. 2010, pp. 67–72.

[31] Dominik Reinhardt and Gary Morgan. "An embedded hypervisor for safety-relevant automotive E/E-systems". In: *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*. IEEE. 2014, pp. 189–198.

[32] Felix Bruns et al. "An evaluation of microkernel-based virtualization for embedded real-time systems". In: *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*. IEEE. 2010, pp. 57–65.

[33] Rashid Mijumbi et al. "Network function virtualization: State-of-the-art and research challenges". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 236–262.

[34] European Telecommunications Standards Institute (ETSI). *Network Function Virtualization (NFV); Architectural Framework V1.1.1*. Group Specification. ETSI NFV ISG, Oct. 2013.

[35] ETSI. *OSM Release Two—A Technical Overview*. White paper. European Telecommunications Standards Institute (ETSI), Apr. 2017.

[36] *Open Source MANO*. Visited on 29/05/2017. URL: https://osm.etsi.org/.

[37] *OPEN-O*. Visited on 29/05/2017. URL: https://www.open-o.org/.

[38] *OpenStack*. Visited on 29/05/2017. URL: https://www.openstack.org/.

[39] *OpenDaylight*. Visited on 29/05/2017. URL: https://www.opendaylight.org/.

[40] *ONOS*. Visited on 29/05/2017. URL: https://www.onosproject.org/.

[41] Christofer Price and Sandra Rivera. *OPNFV: An open platform to accelerate NFV*. White paper. 2012.

[42] Markus Fidler and Amr Rizk. "A guide to the stochastic network calculus". In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 92–105.

[43] Amaury Van Bemten and Wolfgang Kellerer. *Network Calculus: A Comprehensive Guide*. Tech. rep. 201603. Technische Universität München—Lehrstuhl für Kommunikationsnetze, Mar. 2016.

[44] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Vol. 2050. Springer Science & Business Media, 2001.

[45] Eric W. Weisstein. *Poisson Distribution. From MathWorld—A Wolfram Web Resource*. Visited on 02/05/2017. URL: http://mathworld.wolfram.com/PoissonDistribution.html.

[46] Florin Ciucu. "Network calculus delay bounds in queueing networks with exact solutions". In: *Managing Traffic Performance in Converged Networks*. Springer, 2007, pp. 495–506.

[47] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[48] Frank Kelly. "Notes on Effective Bandwidths". In: *Stochastic networks: theory and applications* 4 (1996), pp. 141–168.

[49] Yuming Jiang. "A note on applying stochastic network calculus". In: *Proceedings of SIGCOMM*. Vol. 10. Citeseer. 2010, pp. 16–20.

[50] Kyung Chang Lee, Suk Lee, and Man Hyung Lee. "Worst case communication delay of real-time industrial switched Ethernet with multiple levels". In: *IEEE Transactions on Industrial Electronics* 53.5 (2006), pp. 1669–1676.

[51] *SERCOS the automation bus—Communication Specification*. v. 1.3.1-1.12. sercos international. Sept. 2013.

[52] *EtherCAT System Documentation*. v. 5.1. BECKHOFF. Aug. 2016.

[53] Norman Finn et al. *Deterministic Networking Architecture*. Visited on 19/05/2017. Mar. 2017. URL: https://tools.ietf.org/html/draft-ietf-detnet-architecture-01. (work in progress).

[54] *IEEE 802.1 Time-Sensitive Networking Task Group*. Visited on 19/05/2017. URL: http://www.ieee802.org/1/pages/tsn.html.

[55] Clare Horsman et al. "When does a physical system compute?" In: *Proc. R. Soc. A*. Vol. 470. 2169. The Royal Society. 2014, p. 20140182.

[56] Dominic C Horsman. "Abstraction/Representation Theory for heterotic physical computing". In: *Phil. Trans. R. Soc. A* 373.2046 (2015), p. 20140224.

[57] The Metro Ethernet Forum (MEF). *Ethernet Services Attributes Phase 2, MEF 10.2*. Technical Specification. Oct. 2009.

[58] European Telecommunications Standards Institute (ETSI). *Network functions virtualisation (NFV); reliability; report on models and features for end-to-end reliability, V1.1.1*. Group Specification. ETSI NFV ISG, Apr. 2016.

[59] Jiajia Liu et al. "Reliability evaluation for NFV deployment of future mobile broadband networks". In: *IEEE Wireless Communications* 23.3 (2016), pp. 90–96.

[60] Jinkyu Kang, Osvaldo Simeone, and Joonhyuk Kang. "On the Trade-Off between Computational Load and Reliability for Network Function Virtualization". In: *arXiv preprint arXiv:1704.06864* (2017).

[61] European Telecommunications Standards Institute (ETSI). *Network Function Virtualization (NFV); Management and Orchestration, V1.1.1*. Group Specification. ETSI NFV ISG, Dec. 2014.

[62] Thomas Anderson et al. "Overcoming the Internet impasse through virtualization". In: *Computer* 38.4 (2005), pp. 34–41.

[63] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[64] Jonathan S Turner et al. "Supercharging planetlab: a high performance, multi-application, overlay network platform". In: *ACM SIGCOMM Computer Communication Review* 37.4 (2007), pp. 85–96.

[65] Andreas Fischer et al. "Virtual network embedding: A survey". In: *IEEE Communications Surveys & Tutorials* 15.4 (2013), pp. 1888–1906.

[66] Abdeltouab Belbekkouche, Md Mahmud Hasan, and Ahmed Karmouch. "Resource discovery and allocation in network virtualization". In: *IEEE Communications Surveys & Tutorials* 14.4 (2012), pp. 1114–1128.

[67] Robert Ricci, Chris Alfeld, and Jay Lepreau. "A solver for the network testbed mapping problem". In: *ACM SIGCOMM Computer Communication Review* 33.2 (2003), pp. 65–81.

[68] Yong Zhu and Mostafa H Ammar. "Algorithms for Assigning Substrate Network Resources to Virtual Network Components." In: *INFOCOM*. Vol. 1200. 2006. 2006, pp. 1–12.

[69] Minlan Yu et al. "Rethinking virtual network embedding: substrate support for path splitting and migration". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 17–29.

[70] Soroush Haeri and Ljiljana Trajković. "Virtual Network Embedding via Monte Carlo Tree Search". In: *IEEE Transactions on Cybernetics* (2017).

[71] Juan Felipe Botero et al. "Energy efficient virtual network embedding". In: *IEEE Communications Letters* 16.5 (2012), pp. 756–759.

[72] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[73] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping". In: *IEEE/ACM Transactions on Networking (TON)* 20.1 (2012), pp. 206–219.

[74] Md Mashrur Alam Khan et al. "Multi-path link embedding for survivability in virtual networks". In: *IEEE Transactions on Network and Service Management* 13.2 (2016), pp. 253–266.

[75] Marcelo Caggiani Luizelli et al. "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions". In: *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE. 2015, pp. 98–106.

[76] Michael Till Beck and Claudia Linnhoff-Popien. "On delay-aware embedding of virtual networks". In: *The sixth international conference on advances in future internet, AFIN*. Citeseer. 2014.

[77] Giorgos Chochlidakis and Vasilis Friderikos. "Low latency virtual network embedding for mobile networks". In: *Communications (ICC), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1–6.

[78] Ning Zhang et al. "Software Defined Networking Enabled Wireless Network Virtualization: Challenges and Solutions". In: *IEEE Network* 99 (2017), pp. 12–19.

[79] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd ed. The MIT Press, 2009.