Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Kommunikationsnetze

Design and Performance Modeling of an Application-Aware Network Abstraction Layer for Partially Deployed SDNs

Christian Sieber, M.Sc.

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

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr.-Ing. Gerhard Rigoll
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Wolfgang Kellerer
	2.	Prof. Dr. Tobias Hoßfeld

Die Dissertation wurde am 20.11.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 05.04.2019 angenommen.

Design and Performance Modeling of an Application-Aware Network Abstraction Layer for Partially Deployed SDNs

Christian Sieber, M.Sc.

05.04.2019

Abstract

Today, despite years of research and development, network management is still complex, error-prone and requires human intervention. This is due to the fact that today's traditional forwarding devices are feature-packed, monolithic devices with proprietary interfaces and non-standardized configuration data-models. In traditional networks, manually configured virtual networks and distributed routing protocols executed on the devices, form a distributed control logic. Software-Defined Networking (SDN) reduces complexity in the forwarding devices by moving control decisions to a logically centralized entity, the SDN controller. Through an open interface, the SDN controller can program the forwarding behavior of the devices by pushing simple forwarding rules to the devices. The advantages of SDN-based networks are comprehensively investigated in research. Operational experience from global-scale Internet companies show an increase in flexibility and efficiency of SDN-based networks in the field. However, techno-economical and operational considerations restrict most network operators from fully replacing traditional forwarding devices with SDN-capable devices in the short-term. Incremental migration strategies have to be devised which allow a network operator to reap the benefits from the partially deployed SDNs during a potentially long-lasting migration process. How to manage, operate and provide Quality of Service (QoS) in such hybrid SDN/legacy networks is still a challenge in several areas. This doctoral thesis at hand extends the state of the art in hybrid networking by tackling the following three challenges.

The first challenge is the design of a programmable abstraction layer for partially deployed SDNs. Based on an informational Request-For-Comments (RFC), which introduces the high-level architecture, the thesis proposes and demonstrates the detailed design of such an abstraction layer. The challenge here is to combine different management data-models, e.g., for QoS, with the traditional and SDN control plane for forwarding. Furthermore, trade-offs arise when choosing the level of detail of the abstraction layer. Additionally, undesired side-effects of frequent reconfigurations of traditional devices have to be considered. By modeling the network as a graph augmented with the devices' QoS processing pipelines and configuration side-effects such as data-plane interruptions, control applications are able to programmatically control forwarding and QoS simultaneously through a domain-specific programming language.

The design is implemented in a proof-of-concept setup and measurements of the set-up show the feasibility of the design.

The second challenge is the performance modeling of such network abstraction layers for hybrid SDNs. Two approaches are presented. The first approach uses queuing theory to derive an analytical performance model for the flexibility of hybrid networks based on the topology and the stage in the SDN migration process. The results show that traditional devices with slow management interfaces can severely restrict the global flexibility of hybrid networks. A survey of a collection of real-world topologies shows further that this holds true for most networks even in late stages of the SDN-migration process. Hence, the feasibility of SDN use cases requiring a high rate of reconfigurations, such as load-balancing or reactive control, have to be re-evaluated for hybrid networks. The second approach applies machine-learning for the performance estimation of abstraction layers at run-time. Here different performance models are compared and a machine-learning pipeline proposed. The results show that the pipeline enables robust estimations of the performance limits in environments with varying compute resource availability or changes in control traffic patterns.

The third challenge is the user- and application-aware resource allocation through the abstraction layer. The problem is divided in first, how to define application- and user-awareness as part of the abstraction layer, second, how to determine shares of network resources based on the needs of the applications and third, how to allocate the shares of network resources in hybrid networks. To this end, the abstraction layer is adapted to include applications and their requirements into the hybrid network abstraction. Measurements, application key performance indicators and user experience models are then combined to derive application- and user-aware utility functions. A mixed-integer linear program is formulated to calculate the fair shares of the available capacity per application using the utility functions and enforced via programmable pacers close to the applications. The results show an improved and predictable application behavior in scenarios where over-utilized links restrict the available throughput. Furthermore, the proposed approach can be implemented using only the management primitives the abstraction layer offers for hybrid networks.

Kurzfassung

Computernetze sind ein integraler Bestandteil im Arbeitsumfeld, von der automatischen Erfassung von Waren durch Sensoren, über die Bereitstellung von Diensten und Ressourcen aus einer oder mehreren zentralen Cloudinfrastrukturen, bis hin zu Telefon- und Videokonferenzen über Standorte hinweg. Während die Virtualisierung der Orchestrierung von Computersystemen in den letzten Jahren viele Vorteile gebracht hat, wie einfache Wart- und Konfigurierbarkeit, ist die Orchestrierung von größeren Netzen seit Jahrzehnten unverändert kompliziert, fehleranfällig und wenig flexibel. Dies liegt unter anderem an veralteten Verwaltungsprotokollen, unterschiedlichen Implementierungen der Hersteller und in sich geschlossenen Switches und Routern ohne Möglichkeiten der Programmierung aus der Ferne. Software-Defined Networking ist ein neuer Ansatz, der Geräte und Protokolle vereinheitlichen und austauschbarer machen soll. Hierbei werden die aktuell verteilten Steuerungsprotokolle auf den Geräten durch einen zentralen Controller ersetzt und die Geräte dabei zu generischen und programmierbaren Switches reduziert.

Die vorliegende Doktorarbeit beschäftigt sich mit der Umstellung von traditionellen Netzen auf Netze mit Software-basierten Switches. Es wird eine Abstraktionsschicht entwickelt, die die Verwaltung von Netzen mit traditionellen und Software-basierten Switches ermöglicht, sogenannte hybride Netze. Weiterhin beschäftigt sich die Arbeit mit der Leistungsbewertung dieser hybriden Netze. Die Leistungsbewertung wird auf der einen Seite mittels der Warteschlangentheorie für das Gesamtnetz durchgeführt und auf der anderen Seite durch maschinelles Lernen auf eine Instanz der Abstraktionsschicht während des Betriebs. Zusätzlich stellt die vorliegende Arbeit eine Erweiterung der Abstraktionsschicht in Richtung benutzer- und anwendungsorientierter fairer Ressourcenzuteiltung vor. Die Erweiterung ermöglicht es der Netzorchestrierung Ressourcen je nach Bedarf an Anwendungen zuzuteilen um jedem Netzteilnehmer eine zufriedenstellende Benutzererfahrung mit dem Netz zu ermöglichen. Die Ergebnisse der Doktorarbeit erweitern den aktuellen Stand der Technik der Verwaltung von hybriden Netzen in drei Richtungen.

Erstens wird eine programmierte Abstraktionsschicht vorgestellt, die in dieser Form bisher nicht existierte. Die programmierbare Abstraktionsschicht erlaubt die Verwaltung von hybriden

Netzen durch übergeordnete Netzanwendungen. Dabei wird die übliche abstrakte Netzstruktur um die inneren Komponenten der Geräte erweitert. Dies bezieht sich im Besonderen auf die Komponenten zur Dienstgüte, die bisher für Netzanwendungen nicht erkennbar waren und deren Konfiguration bisher menschlichen Eingriff benötigt haben. Erste Versuche mit der Abstraktionsschicht zeigen, dass diese Netzanwendungen erweiterte Eingriffsmöglichkeiten bereitstellen kann, Fehler bei der Konfiguration vermeiden kann und Unterbrechungen der datenübertragenden Netzschicht minimieren kann.

Zweitens wird eine Leistungsbewertung hybrider Netze und der Abstraktionsschicht durchgeführt, die es erlaubt hybride Netzarchitekturen zu bewerten. Die Ergebnisse zeigen hier auf, dass schon eine geringe Anzahl an traditionellen Geräten die Gesamtleistung des Netzes hinsichtlich der möglichen Konfigurationsrate erheblich einschränken können. Weiterhin wird eine Metrik entwickelt, die es ermöglicht das Potential einer Netzarchitektur bezüglich ihrer Eignung zur Migration auf Software-basierte Geräte zu bewerten. Basierend auf den Erkenntnissen müssen viele der bisher für Software-basierte Netze entwickelten Einsatzszenarien, die auf eine hohe Konfigurationsrate ausgelegt sind, für hybride Netze auf ihre Machbarkeit geprüft werden. Weiterhin wird eine mehrstufige Pipeline für das maschinelle Lernen entwickelt, welche es ermöglicht die maximale Leistung einer Instanz der Abstraktionsschicht im Betrieb abzuschätzen. Zusätzlich wird hier ein Verfahren entwickelt, das Ressourcenschwankungen des zugrundeliegenden virtuellen Systems erkennt und gegebenenfalls die Abschätzung korrigiert. Die Ergebnisse zeigen, dass das entwickelte Verfahren genaue Abschätzungen der Leistung auf eine Vielzahl von virtuellen Systemen und für verschiedene Instanzen der Abstraktionsschicht ermöglicht. Weiterhin werden Ressourcenschwankungen zuverlässig erkannt und die Abschätzung zügig korrigiert.

Drittens wird das Problem der fairen Ressourcenverteilung durch die Abstraktionsschicht in hybriden Netzen als Optimierungsproblem mit anwendungsspezifischen Kosten-/Nutzenfunktionen formuliert und die Vorteile gegenüber herkömmlichen Netzen untersucht. Dabei werden auch verschiedene Einsatzszenarien der untersuchten Anwendungen berücksichtigt und basierend auf existierenden Benutzerstudien modelliert. Die Ergebnisse zeigen, dass der vorgeschlagene Ansatz die Benutzererfahrung mit den Anwendungen erheblich steigern kann und eine faire Verteilung der vorhandenen Netzressourcen erlaubt.

Contents

Chapter 1

Introduction

Modern society increasingly depends on local- and wide-area networks. Emerging trends and technologies such as the Internet of Things, Cyber Physical Networks or 5G will further reinforce this dependency and bring connectivity to everything, from refrigerators and traffic lights, to production lines. Sufficient capacity, low latency, flexibility and dependability are not nice-to-have anymore, but have to be an inherent part of every network's design. At the same time network operators have to be cost-efficient. Networks should operate autonomously, hence requiring minimal human intervention, available capacity should be fully utilized and new use cases be implemented without replacing existing devices or having to buy new ones.

Traditional forwarding devices are composed of three high-level parts, also referred to as *planes*. The *data-plane* is responsible for fast processing and forwarding of packets according to simple rules. It is commonly implemented as an Application-Specific Integrated Circuit (ASIC) to achieve the required high packet processing rates close to the rate of the underlying physical interface, e.g., typically 1 Gbps to 40 Gbps over copper-based wires. The *control-plane* sits on top of the data-plane and generates the forwarding rules which are pushed to the ASIC. It is commonly a piece of software running on a Central Processing Unit (CPU) inside the device. The forwarding rules for the ASIC are derived based on forwarding graphs generated by distributed routing algorithms such as Open Shortest Path Forwarding (OSPF) or Border Gateway Protocol (BGP). The third part is the *management-plane*. The management-plane is responsible for configuration, maintenance and monitoring of the devices. The network operator can use the management interfaces provided by the devices, e.g., Command Line Interface (CLI) or Hypertext Transfer Protocol (HTTP)-based interfaces, to configure the different components and algorithms of the device. For example, the operator may use them to configure the parameters of the routing algorithms, Virtual Local Area Networks (VLANs) or power saving options.

Network management is still heavily human-centered with minor autonomous behavior. The root cases are manifold. There exist no established management data-models and no programmatic access to the devices. Furthermore, vendors implement custom CLIs and data-models, often resulting in vendor lockin where network operators find themselves compelled to buy all their required network hardware from the same vendor. As a result of this complexity of data-models, protocols and interfaces, network management depends on domain experts, studying of complex handbooks and frequent manual configuration. Network Management Systems (NMSs) offer some support, but are often not extensible, do not offer interfaces to network control applications and still require human interaction to write device-specific code snippets for configuration actions. In the end, this leads to a high vulnerability to network failures of traditional networks. Surveys account the human factor for a large percentage of network outages, as a symptom of an overwhelming complexity of the system. A study accounts configuration errors by humans for 50 % to 80 % of network outages [98]. Another comprehensive data-center study by a big cloud provider shows that, although networks are designed and configured to provide a high level of redundancy, configuration errors can still lead to dramatic network outages [72].

Software-Defined Networking (SDN) is a paradigm shift which disaggregates the forwarding devices by moving the control-plane away from the devices and into a logically centralized entity, referred to as SDN- or network controller [61, 104]. The network controller then adds and modifies forwarding rules on the devices through an open interface based on its global view of its network domain. As a result, the devices are reduced to simple rule-based forwarding devices with no internal control-logic. As of today, production SDN networks can only be found in data-centers and wide-area networks of the largest Internet companies. Research and the experience of those companies show that SDN enables fast adaptation to changes in the network. Furthermore, the ability to quickly add and modify rules increases network utilization of data-center and wide-area networks [92].

Thanks to their size, the largest Internet companies can migrate whole networks and data-centers to SDN at once. Furthermore, they are less dependent on the hardware vendors, as their size allows them to develop and manufacture SDN-capable forwarding devices according to their own specifications [178]. But, other network operators are forced by techno-economical constraints to deploy SDN incrementally. The incremental migration from traditional networking to SDNs and the operation of such partially deployed hybrid networks bring their own sets of challenges. The doctoral thesis at hand investigates five challenges in the research area of SDN migration and operation of such intermediate hybrid networks.

1.1 Research Challenges

The following section gives an overview over the research challenges tackled in this thesis. The challenges revolve around the design and performance modeling of an Network Services Abstraction Layer (NSAL) for partially deployed SDNs with awareness for the applications in the network. An NSAL abstracts hardware and implementations details of the deployed network components and translates abstract control decisions to device-specific configurations. The term NSAL is part of the terminology introduced in RFC7426 [RFC7426]. RFC7426 introduces the idea of an NSAL for hybrid networks which allows control applications on top of the abstraction layer, such as network orchestration systems, to reconfigure heterogeneous networks through a unified interface.

The challenges arise from two basic issues with hybrid networks. First, the heterogeneity of the devices. Traditional devices were not designed with interoperability or predictable reconfiguration performance in mind. This is a problem when moving from infrequent per-device reconfigurations to frequent centralized control decisions. Furthermore, the difference of the devices regarding Quality of Service (QoS) features makes it challenging to implement application-aware resource allocation in the network. Second, the necessity of the NSAL as intermediate layer between control applications and the devices. The reactivity of the network control depends thus not only on the devices but also on the control message processing and translating performance of the NSAL. Next, the challenges are introduced in detail.

C1: Quantifying Flexibility and the Interplay with the Data-Plane

The NSAL is a layer between the network and the control applications. It depends, in terms of ability to reconfigure and performance it can offer to the control applications, on the deployed devices and mechanisms in the network. The first challenge is therefore determining and quantifying relevant influence factors on the design and flexibility of the NSAL. To this end, measurement procedures have to be developed. The measurements procedures have to be able to record the timings of reconfigurations and show the interplay of reconfigurations with the data-plane for a range of forwarding devices. Of particular interest are here devices suitable for incremental SDN deployment strategies, mechanisms for in-network resource allocation and the impact of data-plane impairments on higher layers, e.g., the impact on the transport and application layer. The hereby gained measurement results serve as a foundation for the design and performance modeling of the NSAL as described in the subsequent challenges.

C2: Design of an Abstraction Layer for Partially Deployed Software-Defined Networks

The second challenge is the actual design of the NSAL for hybrid networks. The requirements to the design are manifold. First, a common abstraction for the control- and management-plane

has to be found, including QoS-configuration abstraction which requires interaction with both planes. This is challenging as traditional and SDN devices are conceptually different and supported QoS mechanisms differ between devices. Second, the influence factors determined by **C1** have to be considered. Third, potential design trade-offs have to be analyzed and deployment guidelines derived from the analyses. Forth, an architecture has to be devised which provides the abstraction to control applications through a programmable interface.

Performance Modeling, Analytical (C3) and Learning-based (C4)

Network management is mostly proactive and static in traditional deployments. A new network is set-up, configured and tested and afterwards the configuration stays unchanged for months. In contrast to that, SDN and OpenFlow provide new possibilities. The logically centralized view and a fast binary protocol facilitate reactive network management ideas where the configuration is changed upon specific events such as link failures, a sudden rise in traffic volume or even the arrival of specific network packets. Therefore, it can be expected that the number of reconfigurations considerably increases in modern SDN-enabled networks. For partial deployments this can be expected as well, as a network operator will want to profit from the investment in SDN devices even if not all devices in the network are replaced.

Therefore, the performance of the NSAL is critical for the operation of a hybrid network. An overloaded NSAL increases reconfiguration delay for control applications and can not react fast enough to events in the network. Hence, the third and forth challenge is the performance modeling of the NSAL from two research directions. An accurate performance model can be used on the one side for system dimensioning (offline) and on the other side for capacity estimation (online). Furthermore, an accurate performance model allows to assess the feasibility of control use cases based on their frequency of reconfigurations.

In detail, the third challenge (**C3**) tackled in this thesis is the analytical modeling of the NSAL for system dimensioning in terms of flexibility. The measurements in Chapter 3 will show a significant difference between traditional and SDN devices regarding their maximum reconfiguration rate. Hence, the NSAL's reconfiguration rate depends, among other factors, on the current state of SDN migration. An analytical model has to be found which allows to assess the feasibility of reconfiguration work-loads in partially migrated SDNs.

When moving in the direction of real-world deployments, the performance of a software-based NSAL executed on a shared computing system with varying workloads is complex to model in detail. Machine-learning can help to create a model at runtime based on monitoring samples. The forth challenge (**C4**) is therefore the learning-based performance modeling of NSAL instances at runtime for capacity estimation. Mechanisms and procedures have to be

developed which can cope with noise in the samples, adapt to the execution environment at hand and still guarantee accurate estimations.

C5: Application- and User-Aware Resource Allocation through the NSAL

The measurements regarding data-plane impairments and user-awareness in this thesis will show how critical a loss-free transport of application data is. Interruptions due to reconfigurations, link over-utilization or forceful packet loss by traffic policing, can severely impact the applications and users relying on the network. Hence, resource allocation has to be an integral part of every NSAL design. Therefore challenge **C5** concerns itself with the integration of application-aware resource allocation into the proposed NSAL.

In detail, the challenge encompasses the following questions: First, how to integrate applications as part of the NSAL abstraction in order to make them accessible for control applications? Second, how to determine a resource allocation which is application-aware? This includes deriving a relationship between network resources, application Key Performance Indicator (KPI)s and the users' Quality of Experience (QoE). Third, how to allocate the resources for each application in the hybrid network through the NSAL, considering the heterogeneous nature of the devices and only partial SDN-deployment?

1.2 Contributions

The following section highlights the contributions to the respective research areas.

Measurements of Flexibility and Interference [9, 12, 15, 16]

The first contribution concerns itself with measurements of the interference of management actions on the data-plane, transport-level and application-level. In particular, the measurements target the following questions: How often can a device be reconfigured? How long does it take until the change is measurable on the data-plane? Are there any side-effects during the reconfiguration, such as lost packets? Furthermore, the measurements take a closer look at how the configuration of policing disrupts the data-plane and higher-layer applications, such as video streaming clients.

[15] presents the measurement results regarding reconfiguration times of SDN and traditional devices. The measurements show that the investigated traditional devices are not designed for fast reconfigurations and management actions are processed for several hundred milliseconds before being applied to the data-plane. SDN-capable devices on the other hand show sub-millisecond reconfiguration times. [12] takes a closer look at interruptions as a consequence of management actions. The results show that the investigated SDN-capable devices exhibit no data-plane interruptions during reconfiguration of the forwarding rules. For

the traditional devices, the measurements focus on VLAN reconfiguration, as VLAN tunnels are an integral part of a popular SDN migration strategy. Here the results show that changing the VLAN configuration can trigger reboots of the forwarding device's interface. The reboot results in data-plane interruptions of about 6 s.

[9] and [16] concern themselves with the effects of policing on the data-plane, transport-level and application-level. The results show that policing has a negative effect on both transport- and application-level. On transport-level, policing results in lost packets due to the interaction between Transmission Control Protocol (TCP) congestion control and policing dropping excess packets. On application-level, policing results in fluctuating throughput. This is challenging for applications which adapt their behavior to the available throughput, such as video streaming clients.

NSAL Design, Challenges and Trade-Offs [10, 12]

The concept of a NSAL is defined in [RFC7426] and enables the evolution of network control and network management towards holistic programmable network control. As of writing, there does not exist a design for such an NSAL for partially deployed SDN networks. In a first step, [10] demonstrates abstractions for QoS which allow, with the help of a traditional NMS, to configure QoS in hybrid networks. [12] then proposes a complete design for a programmable NSAL for SDN and traditional networks. Furthermore, the paper discusses challenges and trade-offs which influence the design. A Domain Specific Language (DSL) is introduced and its effectiveness for QoS use cases in hybrid networks evaluated. The results show that, with the help of the proposed NSAL, data-plane interruptions can be minimized by combining knowledge from the control- and management-plane.

Offline Performance Modeling: Feasibility of Reconfiguration Rates [12, 15]

The measurements in [12, 15] show a difference in reconfiguration times of traditional and SDN devices. Hence, traditional devices with slow reconfiguration times can slow down network control applications. In [15], the impact of traditional devices on the global reconfiguration rate is investigated in detail using queuing theory and a set of real-world topologies. In the paper, the NSAL is modeled as a *M/D/1* system for different stages of the incremental SDN migration process. The results show that even a small number of traditional devices can severely impact the global reconfiguration rate. As a consequence, the feasibility of common SDN use cases, such as real-time load-balancing, have to be re-evaluated for hybrid networks.

Online Performance Modeling: Learning of Performance Limits [11, 13, 18]

The performance of software processes executed on CPUs is generally influenced by, among others, the hardware platform, the CPU scheduler and co-location of other processes. Software-

based NSALs are no exception. An overloaded NSAL increases the control latency and slows down the propagation of important notifications from the forwarding devices to high-layer control applications. Therefore, it is important to better understand the performance of NSALs in dynamic environments. [13] presents the design and implementation of a scalable NSAL load generator. By emulating distributed control applications with flexible request generators, the generator is able to induce a specified target utilization at the NSAL. Furthermore, the flexible request generators allow to generate realistic control traffic following Poisson arrival processes. [11] introduces models for the performance of NSALs. Different models are compared and evaluated on different hardware and virtualization platforms. As a result, a model is proposed which can explain the performance of a software NSAL instance with a small margin of error. The benchmark and results of [11, 13] are combined in [18] to train the performance models at runtime. Furthermore, through the use of outlier detection using Support Vector Machines (SVMs), concept drifts in the model are detected and the model is refreshed. The whole training process is specified in a parameterized machine learning pipeline. The results show that the pipeline is able to estimate the performance of an NSAL accurately from measurements at runtime. Additionally, changes to the underlying hardware platform are detected in a timely manner and the model is refreshed fast.

Application-Awareness [1, 6–8, 20, 17]

The studies [1, 6–8, 20, 17] contribute to the research area of application-awareness. In detail, they take a close look at the main driving force behind the increasing global Internet traffic: Video streaming [170]. On the one side, the studies measure, model and optimize the interaction between network parameters, e.g., throughput and packet loss, on the KPIs of video streaming [6, 7]. On the other side, the studies propose and compare novel video encoding and quality adaptation mechanisms [1, 8, 20, 17] to optimize the interplay between network and application. The results show that even global-scale video platforms can still improve their adaptation to the network and that the frequency of adaptation decisions can be reduced with minimal loss in video quality [6]. The proposed mechanisms for encoding improvements and quality adaptation show that variable bit-rate encoding can reduce the required throughput of video streaming [8] and that machine learning [20] and scalable video encoding [17] can help to cope better with variations in the available throughput.

User-Awareness [4, 5, 2, 19]

User-awareness considers the relationship between the technical KPIs of a system, e.g., video quality in video streaming, and the experience of the user, denoted as the QoE. The studies [4, 5, 2, 19] contribute to this area of research by developing, conducting and evaluating user-experience studies to better understand video streaming as the main contributor to the

global Internet traffic. The results show that the average video quality, the duration on each quality level and the quality level switching frequency and amplitude are the strongest influence factors on the user's QoE for video streaming. In the context of this work, the findings enable a mapping of application KPIs to the experience of the user and through this enable an application- and user-aware resource allocation.

Application- and User-Aware (Fair) Resource Allocation [3]

This contribution revolves around the extension of the NSAL with application- and user-aware resource allocation. The contribution can be divided in three parts. First, the definition of user-awareness which requires subjective QoE studies. Second, the step from user-awareness to application-awareness in terms of application KPIs and QoS requirements. Third, the allocation of the resources in the network based on the QoS requirements. The mapping from network parameter to and from KPIs towards user experience is combined in [3] to design an application- and user-aware resource allocation schema for the NSAL. First, utility functions are derived from application KPIs and user experience models. Through the utility functions and a Mixed-Integer Linear Program (MILP) formulation, fair shares of the available capacity are calculated per application and enforced via pacing at the end-hosts. The results show that in this way available network resources can be used efficiently and the performance of the applications in the network becomes predictable even in scenarios where the network is overutilized. In terms of user-experience, almost all investigated application classes exhibit an improvement in the eyes of the user.

Forwarding Devices based on Commodity Hardware [57, 14, P-NAII]

Along-side the paradigm shift for cost-efficient networking through SDN, network operators aim to deploy cheap commodity hardware for packet forwarding and processing, instead of expensive ASICs. The paper [14] and the patent application [P-NAII] investigate the performance of modern off-the-shelf compute platforms with respect to the packet forwarding. In particular, caching and architectural penalties are quantified and a novel optimization scheme for platforms with segregated memory architectures outlined. The paper [57] investigates this topic further and proposes a novel CPU cache allocation algorithm for co-located packet processing functions. The results highlight the possible fluctuations in performance when using commodity platforms for packet forwarding. This can have negative consequences on the NSALs, as traditional and SDN-enabled switches usually provide predictable forwarding performance. Future work has to investigate this issue further and define interfaces between the server orchestration and the NSALs. Responsibilities have to be divided and algorithms found to combine the areas of compute and packet forwarding.

1.3 Thesis Outline

Next, the outline of the thesis is presented.

Chapter 2, *The State of the Art from the Migration to the Operation of Hybrid Networks*, first gives an overview over the principles of SDN. Second, it discusses the evolution of SDN from simple network controllers for experimental network environments towards complex and programmable NSALs. Third, the chapter elaborates on the relevant technical details of SDN. In particular, the popular OpenFlow protocol is discussed and details on the configuration of QoS on traditional and SDN-enabled devices is given. Afterwards, the chapter discusses the state of the art of incremental migration strategies and of the operation of partially deployed SDN networks. The chapter concludes with a short summary and an outlook on the general challenges tackled in this thesis.

Chapter 3, *Measuring Flexibility and the Impact of Configurations*, presents the measurement results regarding the flexibility of traditional and SDN-enabled devices. Flexibility is defined and quantified here in terms of the reconfiguration timings of the devices. The focus is on VLAN tagging, a popular technique for migration strategies due to its wide-spread availability on traditional devices. Three different traditional devices and four SDN-capable devices are evaluated. Furthermore, the chapter conducts measurements to investigate possible negative impacts of the reconfigurations on the data-plane. First, VLAN tagging is evaluated while monitoring the data-plane. Second, the impact of traffic policing as mechanism for QoS is investigated on the transport-level (TCP) and on the application-level at the example of a video streaming use case.

Chapter 4, *Design of an NSAL for Partially Deployed Software-Defined Networks*, introduces the proposed NSAL and evaluates the design based on a proof-of-concept implementation using a migration strategy based on VLAN tunnels. At first, the chapter discusses the design challenges, how the challenges influence the proposed design and which trade-offs are the result of conflicting design objectives. Afterwards, the chapter discusses how reconfigurations could be combined and how the combined reconfiguration time and possible data-plane impairments can be minimized through the NSAL. The NSAL is then evaluated by two migration use cases, one with QoS settings, and the chapter is concluded with a summary and discussion.

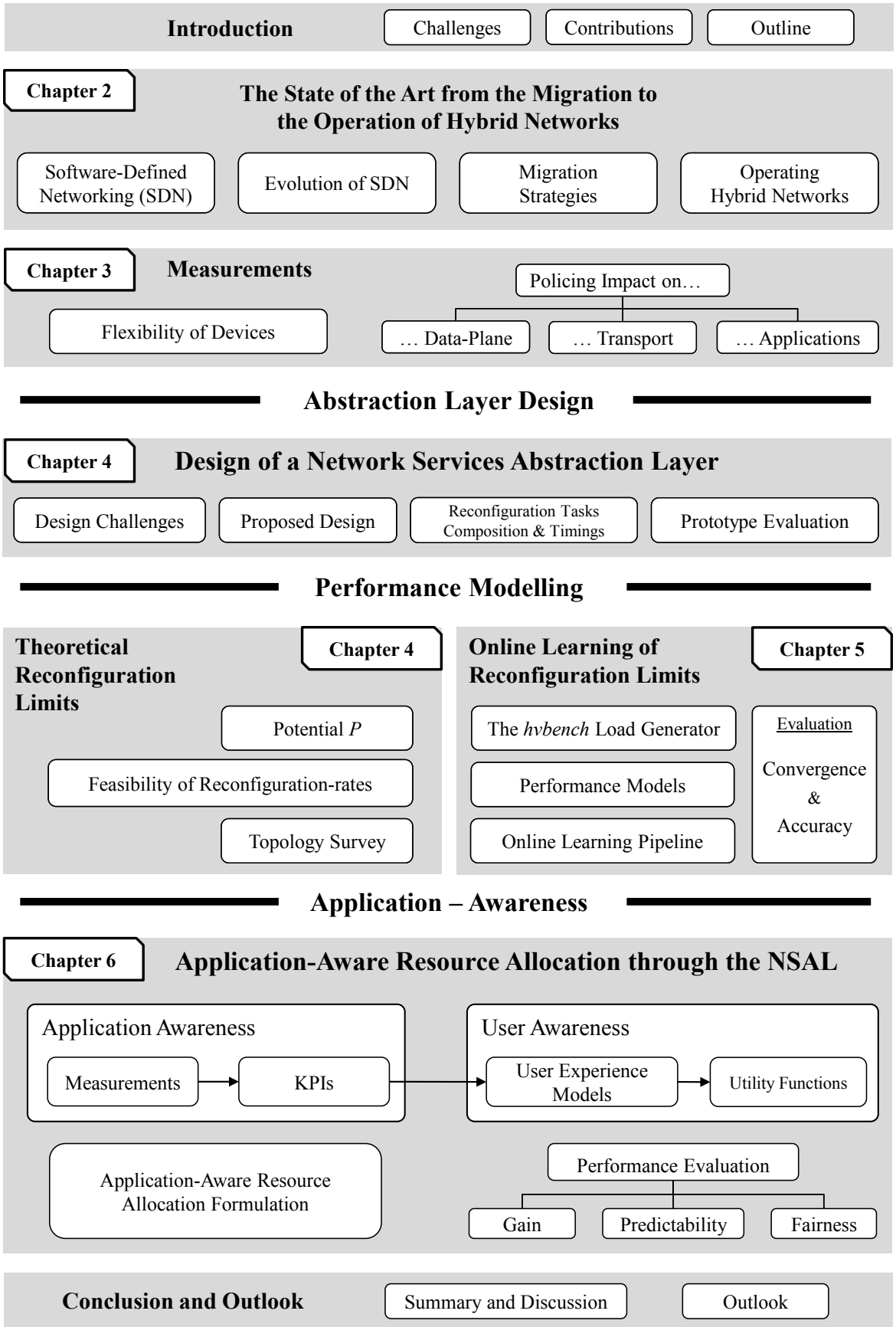
Chapter 5, *Theoretical Performance Limits of an NSAL for Hybrid Networks*, models the proposed NSAL as a queuing system and investigates the theoretical performance limits of the rate of reconfigurations. At first, the system model is introduced and the feasibility of reconfiguration rates defined. The system model shows that the feasibility of a rate depends on the topology and on the stage in the SDN deployment process. Afterwards, the potential P is

proposed, a metric describing the expected gain in terms of reconfiguration rate for a topology independent of the deployment stage. Then, the gain and potential P of a large collection of real-world topologies is evaluated. The chapter concludes with a summary and discussion about the impact of the results on migration planning.

Chapter 6, *Performance Modeling of a Software-Based NSAL at Runtime*, proposes a machine-learning pipeline for the online training of performance models which describe the runtime behavior of programmable NSALs. With the performance models, the maximum reconfiguration rate of a specific software instance can be estimated. At first, three different performance models are proposed and the errors of the models evaluated with measurements in static scenarios on different hardware platforms. Afterwards, a machine learning pipeline based on Orthogonal Distance Regression (ODR), sample weighting and SVM for outlier detection is introduced. The pipeline is evaluated then in scenarios with fluctuating availability of compute resources. The focus of the evaluation is on the estimation error and convergence time. The chapter is concluded with a summary of the results and deployment guidelines in terms of parameter and model selection.

Chapter 7, *Application-Aware Resource Allocation through the NSAL*, introduces resource allocation with application- and user-awareness for the NSAL. First, the required extensions to the graph-based NSAL model are described. Second, application- and user-awareness is defined based on application KPIs and user experience models. Afterwards, utility functions are derived from measurements which map QoS requirements in terms of guaranteed throughput and delay to application KPIs and user QoE. Then an MILP is formulated to solve the problem of max-min fair resource allocation for a given set of applications and utility functions. The output of the MILP in terms of resource allocation is then implemented through the NSAL via packet pacing at the end-hosts. An evaluation is conducted with an increasing amount of parallel applications sharing a constrained link. The evaluation highlights the benefits in terms of fairness between applications and predictable application performance compared to best-effort scenarios. The chapter is concluded with a summary of the results, possible future research directions and current shortcomings of the proposed solutions.

Chapter 8, *Conclusions and Outlook*, concludes this thesis by providing a short summary of the results and contributions. Furthermore, future research directions are discussed.


Introduction
Challenges
Contributions
Outline
Chapter 2
The State of the Art from the Migration to the Operation of Hybrid Networks
Software-Defined Networking (SDN)
Evolution of SDN
Migration Strategies
Operating Hybrid Networks
Chapter 3
Measurements
Flexibility of Devices
Policing Impact on…
… Data-Plane
… Transport
… Applications
Abstraction Layer Design
Chapter 4
Design of a Network Services Abstraction Layer
Design Challenges
Proposed Design
Reconfiguration Tasks Composition & Timings
Prototype Evaluation
Performance Modelling
Theoretical Reconfiguration Limits
Chapter 4
Potential P
Feasibility of Reconfiguration-rates
Topology Survey
Online Learning of Reconfiguration Limits
Chapter 5
The hvbench Load Generator
Performance Models
Online Learning Pipeline
Evaluation
Convergence & Accuracy
Application – Awareness
Chapter 6
Application-Aware Resource Allocation through the NSAL
Application Awareness
Measurements
KPIs
User Awareness
User Experience Models
Utility Functions
Application-Aware Resource Allocation Formulation
Performance Evaluation
Gain
Predictability
Fairness
Conclusion and Outlook
Summary and Discussion
Outlook

Chapter 2

The Long Journey of SDN: The State of the Art from the Migration to the Operation of Hybrid Networks

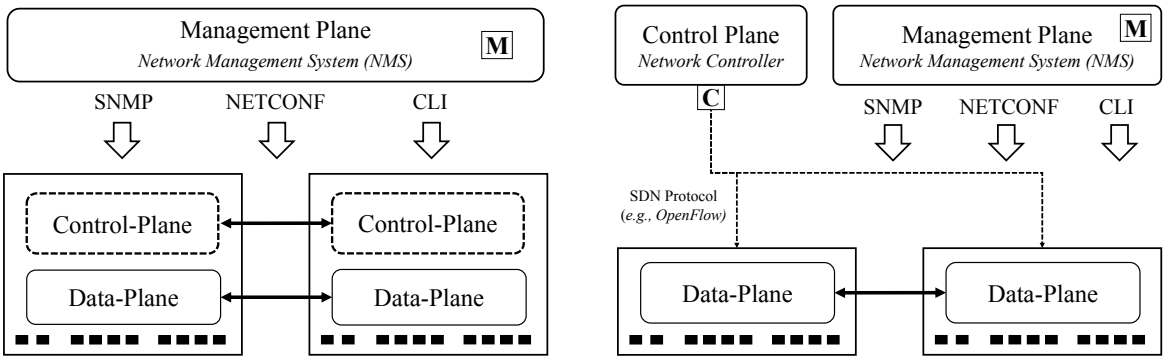
Software-Defined Networking (SDN) takes packet processing and forwarding decisions away from the network devices and puts a logically centralized entity, the SDN controller, in charge. This stands in contrast to traditional networking where devices make forwarding decisions autonomously based on distributed routing protocols. SDN promises ease of management, near real-time control, fine-grained monitoring and increased security.

But in the short-term it will not be economically feasible for most network operators to replace all traditional switches with SDN devices. Hence, migration strategies have to be found which, on the one side, guarantee reliable operation of the hybrid network, and on the other side, make advanced SDN features available early during the migration process.

In the chapter at hand, we first introduce SDN in detail and compare it to traditional networking. Afterwards, we discuss existing migration strategies from the literature. Subsequently, through the the shortcomings of the existing approaches, we motivate the need for the proposed Network Services Abstraction Layer (NSAL) as a means for hybrid network operation. We conclude this chapter with a survey over relevant literature in the area of hybrid networking.

2.1 Software-Defined Networking Overview

Figure 2.1 highlights the key differences between traditional and SDN devices. Traditional devices (Fig. 2.1a) combine data-plane and control-plane in one monolithic device. The control-plane consists of distributed routing protocols such as Open Shortest Path Forwarding



(a) Traditional Networking

(b) Software-Defined Networking

Figure 2.1: Traditional and SDN networking side-by-side. SDN moves the control-plane from the devices to a logically centralized controller. In both cases the management-plane is responsible for device maintenance and Quality of Service (QoS) configuration.

(OSPF) or Border Gateway Protocol (BGP) which communicate with other devices to converge to a consistent and global view of the network domain. The forwarding rules are then derived from the global view and pushed to the data-plane of each device. The data-plane is in charge of processing and forwarding packets based on the rules dictated by the control plane. The devices are managed through protocols such as NETCONF or Simple Network Management Protocol (SNMP) or simply by custom device-dependent Command Line Interfaces (CLIs).

SDN (Fig. 2.1b) on the other hand disaggregates the forwarding devices and reduces them to the data plane. The control plane is moved to the logically centralized SDN controller. OpenFlow [180] is a popular protocol to connect the central controller with the data-plane of the devices. Via OpenFlow, the controller can push match-action rules to the devices' data-planes and request counters per rule such as sent or received packets. Furthermore, via the two features *packet_out* and *packet_in*, the controller is able to send out packets via a device or receive packets from the device, respectively. In SDN, device management is done by protocols such as NETCONF, SNMP or by OF-CONF, a management protocol for OpenFlow devices.

2.2 Evolution of SDN Towards Abstraction Layers

SDN has undergone an evolution from simple network controllers towards recent proposals of holistic network services abstraction layers. In the following, we briefly discuss the history and relevant recent developments of SDN. We begin with the introduction of the OpenFlow protocol in 2008 [121]. With OpenFlow, major device vendors began to support the idea behind SDN. There are other SDN protocols such as Forwarding and Control Element Separation (ForCES) [RFC5810] or Protocol-Oblivious Forwarding (POF) [145], but OpenFlow is the

only protocol with broad vendor adoption. The interested reader is referred to [104] and [127] for a comprehensive study of the history of SDN and alternatives to OpenFlow.

In 2008, McKeown et al. propose the OpenFlow switch specification and protocol [121] as a means to implement and experiment with novel network protocols in campus networks, such as routing or load-balancing algorithms, using commercial forwarding devices. The protocol is a compromise between the vendors desire to hide the internals of their devices and researchers' needs to implement novel algorithms. Furthermore, it is designed to not require any hardware changes to further motivate the device vendors to implement the protocol.

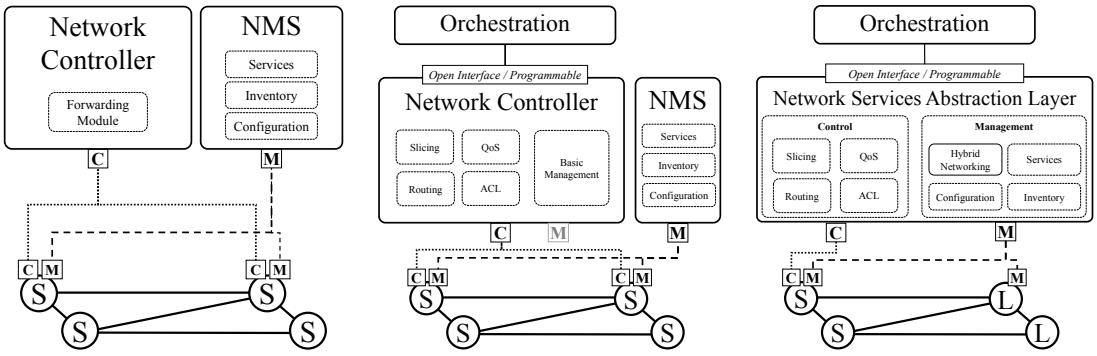
Simple Controllers, Fig. 2.2a

Shortly after, an ecosystem evolved around OpenFlow, which spawned the first simple network controllers as depicted in Figure 2.2a. Controllers such as NOX [75], Beacon [59] and Maestro [35] were proposed. These controllers implement OpenFlow as southbound protocol, i.e., as controller to switch protocol, and offer some basic services such as routing, Access Control Lists (ACLs) and load-balancing. Features such as a comprehensive northbound interfaces, i.e., control application to controller interfaces, or the integration with Network Management Systems (NMSs) are not part of their design.

Sophisticated Controllers, Fig. 2.2b

Driven by modern compute resource orchestration systems, e.g., OpenStack [181], which can provide and operate large number of virtual machines, network controllers evolved into feature-rich platforms with comprehensive Application Programming Interfaces (APIs) for orchestration. Through them, the orchestration can exert control over routing and isolation in terms of forwarding, security and network resource limitations for the virtual machines. Controllers such as *OpenDaylight* [122] or *ONOS* [28] are supported by device manufacturers, service providers and researchers alike. They provide features such as, among others, high-availability, scalability, data-plane abstractions, service chaining and APIs for network virtualization. Furthermore, they can be extended with third-party applications and internal modules and implement multiple southbound protocols besides OpenFlow, such as NETCONF, OF-CONF and SNMP. However, the focus of the controllers are on control and, as of now, the management part is not yet evolved to the level of NMSs. Features such as network application monitoring, long-term network performance monitoring and management, fault management, configuration and change management, downtime scheduling, sophisticated reporting and integration of other type of devices, such as sensors and access points, are generally missing in the controllers.

Our measurements in Chapter 3 show that for a reliable network operation a tighter coupling between network controllers and NMSs is necessary. The measurements highlight in particular



(a) Simple Controller (b) Sophisticated Controller (c) NSAL [RFC7426]

Figure 2.2: Evolution of SDN from simple controllers primarily used in all-SDN (Ⓢ) experimental deployments to holistic abstraction layers for hybrid networks (Ⓢ/Ⓛ). The NSALs integrate southbound control and management into one entity and provide an open and programmable northbound interface for higher-level resource orchestration systems.

that management actions, such as Virtual Local Area Network (VLAN) or QoS configuration, can interrupt the data-plane and have to be considered also by the network controller.

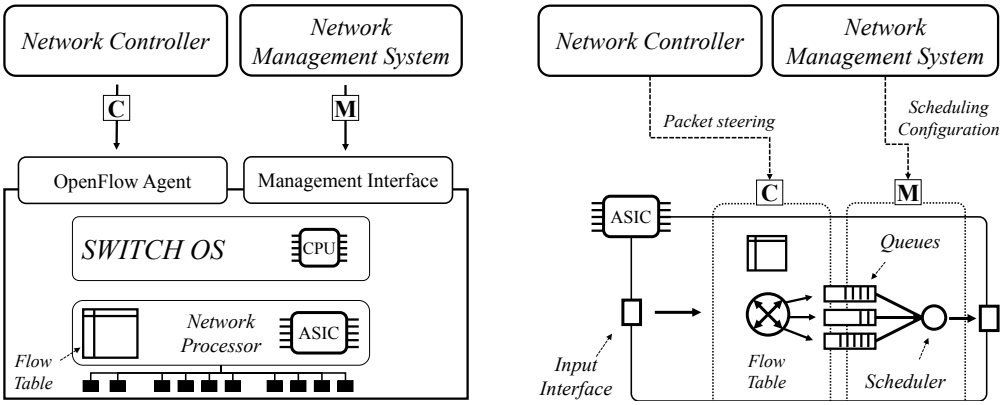
Network Services Abstraction Layers, Fig. 2.2c

NSALs bridge the gap between control and management by introducing a third layer which encompasses the functionality of control and management. The NSAL abstracts the data-plane, control-plane and management-plane and provides a unified interface for northbound applications such as orchestration systems. That way, the NSAL can not only be aware of the data-plane and the control-plane, but also of the management plane. For example, changes to the devices, such as scheduled downtimes, can be coordinated and links drained beforehand to ensure uninterrupted network operation.

2.3 SDN/QoS Technical Implementation

In the following, we discuss relevant details of the implementation of forwarding devices in general and OpenFlow-enabled devices in particular. If not otherwise stated, the information provided refers to OpenFlow version 1.4. OpenFlow provides a standardized interface to the forwarding plane. In a nut-shell, you are able to directly manipulate the forwarding rules of the device by adding, changing and deleting rules from one or more flow tables. In the remainder of this paragraph we illustrate the OpenFlow mechanisms by describing the path of the packets through the device. For the sake of simplicity, some more complex mechanisms of the OpenFlow protocol are left out.

Figure 2.3a illustrates the abstract architecture of an OpenFlow-enabled forwarding device. First, the packets are received on one of the physical ports. A physical port is always associated



(a) OpenFlow Device Architecture (b) Technical QoS Details

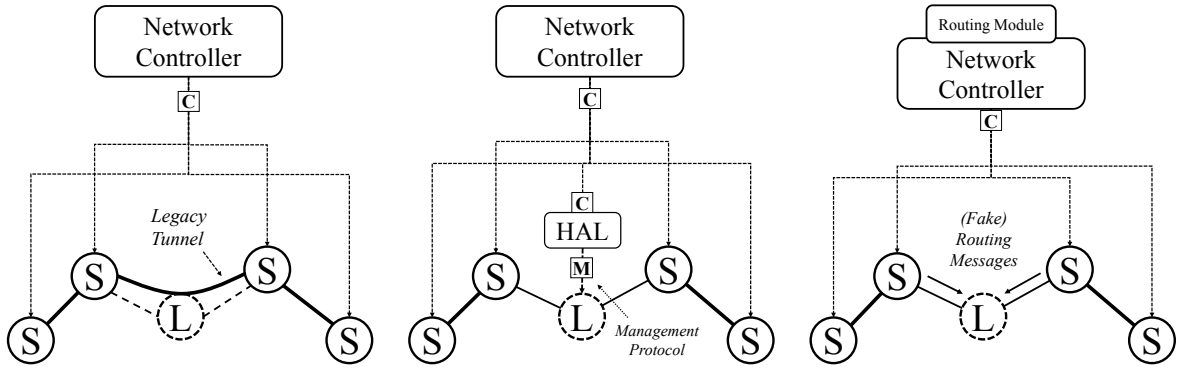
Figure 2.3: Technical details to the implementation of SDN on forwarding devices. Figure (a) illustrates the general architecture of an SDN-enabled forwarding device. Figure (b) highlights the issue with QoS configuration of such devices. Both entities, the network controller and the NMS, have to be involved to configure scheduling on current SDN-enabled forwarding devices.

with a logical switch and therefore the packets get delegated to a specific logical switch. In the logical switch, there are one or more flow tables. Each rule in the flow table is composed of one or more match fields, of one or more actions and one instruction to take when a packet matches the match fields. If none of the rules match in that first flow table, the packets are either dropped or send through the control channel to the controller. Instructions enable to further customize the processing of the packets in the logical switch in the case of a packet match, e.g. adding or removing headers, such as the VLAN header. At the end of the processing, each packet should include one *Output* action specifying the target interface or otherwise the packet is dropped.

2.4 QoS Configuration in Traditional Networking and SDN

There are four basic types of QoS options when processing packets and flows. First, *policing/metering* where packets are dropped if they exceed a specific maximal rate. Second, *shaping* or *pacing* where packets exceeding a specific rate are delayed, i.e. stored in a buffer and released at later time. Third, *scheduling* where packets are put into queues and released by a scheduling strategy, e.g., Weighted Fair Queuing (WFQ). Furthermore, there are tail dropping strategies, e.g., Random Early Drop (RED), which discard already queued packets.

In traditional networks, the QoS configuration is generally done by an human expert or by an NMS through the devices' management interfaces. Packets are steered into the queues or policer elements based on rules and header fields such as the VLAN header or the Multiprotocol Label Switching (MPLS) tag. The main problem of QoS configuration in



(a) Tunnels via Traditional Mechanisms (b) Traditional-to-SDN HAL (c) (Fake) Routing Messages

Figure 2.4: Three common incremental migration strategies found in the literature.

traditional networks is the fact there is no unified management data model for QoS across devices and vendors. There is no standardized way to determine the number and size of queues, the available scheduling strategies or policing options. Often the only way to determine the available options is to study the manual or rely on human domain experts.

In SDN, the packet steering and QoS configuration are independent of each other as depicted in Figure 2.3b. E.g., in the case of OpenFlow, you can define an action for a packet to put it into a specific queue. However, which scheduling strategy is deployed to process the queue can not be defined that way. The scheduling strategy has to be defined beforehand through a management interface. Metering on the other hand is an optional extension of the OpenFlow protocol itself (not depicted in Fig.2.3b). Meters, which limit the maximal rate of a flow, can be added, modified or removed through OpenFlow. Hence, SDN introduces the problem of divided responsibility in terms of QoS configuration. Both entities, the network controller and the NMS, have to be coordinated to implement a queuing QoS strategy in the network.

2.5 Migration Strategies

A short-term replacement of all traditional devices with SDN-enabled devices is not an option for most network operators. Economical and operational reasons restrict the number of devices which can be replaced over time. Hence, incremental deployment strategies have to be found. Next we discuss the three main incremental migration strategies found in the literature: a) configuring tunnels through the traditional domain, b) implementation of traditional Hardware Abstraction Layers (HALs), and c) combining central and distributed routing by (fake) routing protocol messages emitted by the SDN devices.

2.5.1 Tunnels Through The Traditional Networking Domain

Figure 2.4a illustrates the tunnel-based migration strategy. Here traditional devices are pre-configured, e.g., by an NMS, to forward traffic between the SDN devices. Hence, the network controller is unaware of the traditional devices and they are seen by the controller as links. The resulting hybrid SDN network can be operated in the same way as a fully-deployed SDN network. However, the network controller has no control over the hidden traditional nodes and is unable to monitor their utilization. This can decrease the efficiency of algorithms such as for load-balancing or QoS, which might miscalculate the network utilization as a result of the missing information.

In the following we discuss the tunnel-based migration strategies found in the literature. The works can be put into the categories based on the used tunneling-protocol. Approaches based on MPLS and VLAN are proposed as of now. In [36, 110, 111], Levin et al. and Canini et al. propose *Panopticon*, which uses VLAN tunnels in the traditional domain to connect the SDN devices in the network. This enables an abstraction of the network which only consists of SDN devices and traditional devices are reduced to simple forwarders along the configured VLAN tunnels. In [114], Lu et al. introduce *HybNET*, a framework for managing combined traditional and SDN devices. As with Panopticon, traditional switches are reduced to simple VLAN forwarding devices connected and control decisions are made at the SDN devices. In [150], Tu et al. show how to combine SDN routing decisions with MPLS labels in the traditional domain.

2.5.2 Combining Distributed and Central Routing

Figure 2.4c illustrates the migration strategy based on skillfully crafted routing messages. The approach here is to inject an artificial global view into the traditional devices to control the route computation of these devices. To this end, the network controller calculates a (fake) topology per device with nodes, links and link weights. Afterwards, the controller uses the *packet_out* feature of the SDN devices to deliver the routing messages to the traditional devices. After the fake topology is injected into the traditional devices, the devices compute the shortest paths, including backup paths, and populate their local Forwarding Information Bases (FIBs) according to the fake topology.

In the following we discuss the routing-based migration strategies found in the literature. The works can be distinguished by the used routing protocol. OSPF-, Interior Gateway Protocol (IGP)-, BGP-, Address Resolution Protocol (ARP)-based and Spanning Tree Protocol (STP)-based approaches are proposed. Furthermore, there is one approach which uses ACL rules to achieve the same goal.

OSPF-based. In [38, 42], Caria et al. partition OSPF networks into sub-domains by strategic placement of SDN-enabled nodes. This way, the inter-domain traffic is still controlled by OSPF, while SDN can be used for traffic engineering of intra-domain traffic through fine-tuned routing protocol updates. The results show that in topologies with a majority of intra-domain traffic, a full SDN deployment is not necessary. In [39], Caria et al. conduct link capacity planning of partitioned OSPF networks. The results show that, while a pure-OSPF network can require a spare capacity of over 50 %, the hybrid SDN/OSPF approach greatly reduces the amount of spare capacity needed. In [153, 155], Vissicchio et al. introduce the concept of *fibbing*. *Fibbing* generates fake OSPF routing messages to create a virtual topology to control the forwarding behavior at the traditional devices. In [46], Chemalamarri et al. propose *SYMPHONY*, a hybrid SDN controller with a traditional routing component. The authors use *Quagga* at the SDN controller to create a *Legacy Routing Server* which uses the SDN controller to distribute and receive OSPF updates. In [81], He et al. present the design and implementation of a hybrid SDN space network where ground stations deploy traditional equipment and the satellites are combined in an SDN domain. The main focus of the work is the integration of OSPF and BGP through *Quagga* into the SDN controller *POX*. In [130], Peng et al. propose combining OSPF and SDN for traffic engineering in wireless mesh networks.

IGP-based In [151], Vanbever et al. demonstrate how to augment the traditional IGP topology with virtual nodes and virtual links to present each traditional device a fake virtual topology. Through this virtual topology, the traditional devices behave as desired by the central SDN controller.

BGP-based In [124, 135], Nascimento and Rothenberg et al. present *RouteFlow* and the RouteFlow Control Platform (RFCP). The proposed framework shows how to move BGP routing decisions from the core and edge of the network to the centralized SDN controller. In [69], Gämperli et al. introduce an emulation framework for evaluating the convergence time of combined SDN and BGP routing networks.

STP-based In [118], Markovitch et al. propose a hybrid network architecture which exploits the STP protocol, VLANs and a few OpenFlow switches to divide the network into loop-free sub-domains. The results show that with only 2 % to 10 % deployment of SDN-enabled devices, it is possible to enhance traffic engineering and fail-over capabilities of traditional networks.

ARP-based While most related work regarding the combination of central and distributed routing considers traffic in the traditional domain as shortest-path-only, Jin et al. propose *Telekinesis* [95], which uses fake layer 2 packets to trick the Medium Access Control (MAC) learning of the traditional switches and provide route selection in the traditional domain. That way, a forwarding entry for a single MAC address can be created on the traditional switches

via OpenFlow *packet_out* from the closest SDN device. The results show that with 20 % of the devices updated, 70 % of the paths can be controlled. In [96], the same authors extend the previous approach and propose *Magneto*, which uses fake ARP messages to end-hosts in addition to tricking the MAC learning of the traditional switches to achieve better control over the traffic. The results show that a 20 % deployment is sufficient for full network control.

ACL-based In [88], Huang et al. demonstrate *HybridFlow*. *HybridFlow* clusters traditional switches together with one SDN switch. Static configuration on the HybridFlow switches via ACL rules steer all traffic to the SDN devices in each cluster and therefore give the SDN devices the full control over the network.

2.5.3 SDN Hardware Abstraction Layers for Traditional Devices

Figure 2.4b illustrates the migration strategy based on SDN HALs for traditional devices. The OpenFlow protocol was designed to be compatible with current network processors on the market to facilitate easy adoption. Therefore, for devices with replaceable software, it is possible to replace the current traditional control plane with custom OpenFlow agents. Furthermore, it is shown in the literature that even if the software on the device can not be replaced, the provided interfaces such as the CLI or SNMP, are in some cases enough to implement an external OpenFlow agent. Next, we discuss the HAL approaches proposed in the literature.

As part of the European research project *FP7 ALIEN* [176], several proposals and demonstrations of OpenFlow-control over traditional equipment were made. In [27, 68], Belter et al. and Fuentes et al. demonstrate an HAL and OpenFlow agent for traditional devices which allows the traditional devices, such as DOCSIS devices, to appear as OpenFlow devices. In [128], Ogrodowczyk et al. and in [129], Parniewicz et al. describe the HAL in detail. In [60], Farias et al. propose *LegacyFlow*. *LegacyFlow* translates OpenFlow actions to configuration commands of the SNMP, CLI or Hypertext Transfer Protocol (HTTP) Representational state transfer (REST) interfaces of the traditional switches. The results show a latency of up to 800 ms when applying VLAN configuration via SNMP. In [44], Casey et al. propose SDN *Shim*, a Field-Programmable Gate Array (FPGA)-based approach for SDN experimentation. The FPGA device is connected to one of the traditional ports and all traffic is forwarded to this port via VLAN configuration and afterwards, via OpenFlow rules, redistributed to the output ports. In [63], Feng et al. propose the *OpenRouteFlow* architecture consisting of an *OpenRouteFlow* controller and of an *OpenRouter* agent, connected by the OpenFlow-compatible *OpenRouteFlow* protocol. The *OpenRouter* agent is deployed on the traditional switches and translates OpenFlow rules into ACL or Routing Information Base (RIB) rules for the specific device.

2.5.4 Discussion

Next we discuss the three migration strategies. The migration strategy to choose highly depends on the existing devices in the network and on the desired network control use cases. Furthermore, for companies there are individual techno-economical factors to consider. But there is a lack of research on the techno-economical factors of SDN migration. Therefore, in the following, we focus on the technical aspects.

A traditional HAL can be implemented on the devices and does not require any traditional extensions to the network controller. If all network control operations are supported by the combination of HAL and device, this presents a straight-forward migration strategy. However, research in the area of traditional HALs as of today is limited to prototype implementations and no study of the broader applicability is conducted.

Migration by traditional tunnels is well studied, e.g., for VLAN-based tunnels [110], and can be implemented on a broad set of devices. VLAN configuration is a basic switch feature and available on most managed forwarding devices. Problems arise through the fact that SDN controllers are unaware of the tunnels and thus can not monitor the tunnel devices. However, an NSAL with awareness of both SDN and traditional devices can set-up and monitor both the tunneling traditional devices and the SDN devices. From the perspective of network operation, the tunnel strategy has also the advantage that the tunnel configuration can be easily checked and understood by a human operator.

Migration strategies based on combining distributed and central routing offer flexible, per-destination, routing in the traditional networking domain. The research on this areas shows that the proposed approaches are robust and applicable to a wide-range of network topologies and use cases. However, the requirements are higher than for the tunnel-based strategies. The approaches require all devices to support compatibly implementations of a supported link-state protocol. Furthermore, network devices usually do not provide insights into the internal workings and state of the distributed routing protocols. Thus, for a human network operator, retrieving the current state and performing troubleshooting can be difficult.

2.6 Operating Hybrid Networks

Next, we take a closer look at different aspects of how to operate a hybrid traditional-SDN network. Requirements to the operation of a network are manifold. The requirements range from secure and reliable, over flexible and cost-effective, to manageable. The challenges arise from the fact that most existing methods for network operation can not be applied to hybrid networks. Either they are designed for only traditional networks in mind or already tailored to full SDN deployments. There are three main problems. First, the different routing paradigms,

central vs. distributed routing. Second, the problem of two control entities, network controller and NMS. Third, different supported features. In SDN the switches rely on match-action rules and the interaction with the controller, while traditional switches have a large collection of features built-in into the hardware. In the following, we discuss the state of the art regarding the operation of hybrid networks.

First of all, four papers investigate the categorization of different approaches for migration and operation. In [134], Rothenberg et al. survey hybrid networking approaches and discuss the two approaches *LegacyFlow* and *RouteFlow* in detail. In [89, 152], Vissicchio et al. and Huang et al. survey the related work in the area of hybrid SDN networks, define different transitions models and show the current research challenges in this area. In [133], Rathee at al. do a classification of different approaches for the coexistence of SDN and traditional in the same network. Furthermore an overview over topology discovery options in hybrid networks is given.

There are two papers showing the feasibility of such hybrid networks by sharing the experience from experimental deployments. In [139, 140] Salsano et al. present the design of a Open Source Hybrid IP/SDN (OSHI) networking node and the *Mantoo* management tools. OSHI describes the design of a network node for experimenters with SDN-enabled and traditional routing implementation. In [99], Kanaumi et al. introduce the *RISE* testbed, a nation-wide experimental network in Japan and the lessons learned during the design and deployment of the SDN devices.

2.6.1 Interfacing Traditional NMSs and Network Controllers

In [142], Sharma et al. propose the i-NMCS (integrated network management and control system) framework. The framework shows how SDN controller and traditional NMSs can be combined to provide QoS for selected flows in the network. In [166], Zhang et al. propose SDNMP, an SNMP interface for SDN controllers through which traditional NMS can manage the SDN-enabled devices via the SDN controller. For example, the NMS is able to query the topology as seen by the SDN controller, retrieve statistics and inspect the flow table of each device.

2.6.2 Replacement Order of Traditional Switches

In [87], Huang et al. propose heuristics for deciding which part of a traditional network to replace first with SDN-enabled switches. The performance of the algorithms is evaluated by two criteria. The first criteria is the running time of the algorithms. The second criteria is the fraction of traffic in the network which passes through the SDN-enabled switches.

2.6.3 Security

In [24], Amin et al. discuss how to detect ACL policy violations as a result of a topology change in hybrid SDN networks. In [157], Wang et al. propose *Woodpecker*, a framework for detecting and mitigating Link flooding attacks (LFA), a type of Distributed Denial of Service (DDoS) attack. The results show that *Woodpacker* is able to detect LFA attacks and, through appropriate load-balancing decisions, is able to reduce the link-load on the attacked links by 50 %. In [107], Kwon et al. propose *BASE*, an incrementally deployable anti-spoofing mechanism. In [48], Chen et al. propose *SAVSH* and tackle the problem of source Internet Protocol (IP) address validation in hybrid networks. SDN nodes are strategically deployed to filter packets with forged source IP address. The results show that with 15 % of full deployment costs, 90 % of the network prefixes can be checked.

2.6.4 Migration of Traditional Device Configuration to SDN

In [119], Martínez et al. use an Web Ontology Language (OWL) and a learning algorithm to automatically parse and abstract the configuration of traditional devices by analyzing the input and output commands of the CLI of the device. The work provides an interesting approach to investigate the capabilities of a heterogeneous device zoo and can help to settle for a specific migration strategy. In [125], Nelson et al. propose *Exodus*, a system which takes the traditional router configuration as an input and translates it to *Flowlog*, a SDN programming language. *Flowlog* can then be compiled to comparable flow rules for SDN switches.

2.6.5 Failure Recovery and Convergence Time

In [45], Chang et al. show how to combine traditional and SDN-enabled devices to speed up convergence time after a link or node failure. The authors implement a 2-stage forwarding table, the first stage controlled by SDN, which makes it unnecessary for the traditional switch to update the whole routing table after a link or node failure. In [51], Chu et al. discuss single link failure recovery in hybrid SDN networks. In the proposal, each traditional switch is assigned an SDN device as backup tunnel destination in case of link failure. The results show that only a small number of SDN device is needed to allow congestion-free single link failure recovery and full reachability between all nodes.

2.6.6 IP Traffic Matrix Estimation

In [40], Caria et al. uses OpenFlow byte counters and SNMP on traditional devices to construct a IP traffic matrix in hybrid SDN/OSPF networks. The results show that a low number of

SDN devices is sufficient to generate a full IP traffic matrix and that optimal placement of the SDN devices with respect to the measurements is aligned with the placement strategy for traffic engineering. In [132], Polverini et al. also investigate the IP traffic matrix estimation problem in hybrid SDN networks and draw similar conclusions.

2.6.7 Traffic Engineering

In [76], Guo et al. propose *SOTE*, an heuristic algorithm for traffic engineering in SDN/OSPF hybrid networks. The algorithm is evaluated for different deployment ratios and the results show that a 30 % deployment is enough for near optimal traffic engineering performance. In [77], the same authors propose a genetic algorithm to decide which traditional devices to upgrade to SDN, also from the perspective of the traffic engineering. The authors conclude that a deployment ratio of 40 % gives the best results with the genetic algorithm and enables near optimal traffic engineering. In [83], Hong et al. discuss traffic engineering in incremental deployments and show that with a deployment ratio of 20 %, the maximum link usage is one third of the all-traditional deployment. In [86], Hu et al. propose a Polynomial Time Approximation Scheme for maximizing the capacity of the hybrid network through traffic engineering. The results show that near optimal performance can be achieved with a 50 % deployment ratio. In [94], Jia et al. discuss traffic control and show that you can control 95 % of the flows with only 10 % upgrading costs.

In [100], Kar et al. propose and compare different algorithms for selecting traditional devices to be upgraded to SDN with the goal of achieving 100 % SDN path/hop coverage, i.e. every path between two nodes traverses at least one SDN device. In [41, 54], Caria and Das et al. discuss an optimal placement strategy for SDN nodes in traditional networks for the purpose of traffic engineering.

In [22], Agarwal et al. discuss the theoretical improvement in terms of traffic engineering capabilities of partially deploying SDN-enabled devices in the network with shortest-path hop-by-hop routing for the traditional devices. The authors present Fully Polynomial Time Approximation Schemes (FPTAS) to solve the traffic engineering problem. The results show that a low number of SDN devices in the network is enough for significant performance gains in the network. In [148], Sun et al. extend the approach to disjoint multi-path planning.

In [79], He et al. also propose heuristics for traffic engineering based on slight modifications of the traditional routing table to support programmable flow splitting.

2.6.8 Miscellaneous

In [123], Mishra et al. propose a framework which maps OpenFlow match features to unused IP ranges in the traditional network. This requires OpenFlow-enabled switches at every entrance and exit of a sub-domain and static routes on the traditional devices for every end-to-end communication in the network. The framework enables SDN-like policies in a partially deployed SDN network. In [154], Vissicchio, et al. discuss consistent and reliable network update sequences for networks with multiple control planes, e.g. a central SDN control plane and a distributed traditional control plane. In [115], Lukovszki et al. provide an exact and approximation algorithm for incremental deployment of middleboxes in SDN networks. The proposed algorithms are also applicable to incremental deployment of SDN devices in traditional networks.

Mobile Networks, 4G In [163], Kyung et al. propose a migration strategy for 4G networks which incorporates traditional equipment in a SDN/Network Function Virtualization (NFV) deployment strategy. The authors show which and how components of the mobile stack can be virtualized through the SDN controller with the goal of operational expenditure (OPEX) reduction.

Energy Savings In [156], Wang et al. investigate the potential for energy savings in partially deployed SDNs. The authors propose a heuristic for the non-deterministic polynomial-time (NP)-hard problem of finding a minimal set of required nodes. The results show that the heuristic can save 40 % percent of the power consumption in a scenario with a deployment ratio of 60 %.

Techno-economical Perspective In [53], Das et al. discuss migration strategies from the techno-economical perspective and propose a near-optimal greedy algorithm for determining the migration sequence. The results show that already a low number of strategically placed SDN devices enables sufficient traffic engineering options.

Bootstrapping In [101], Katiyar et al. propose an extension to the DHCP protocol to assign a newly attached switch to its controller.

2.7 Summary

In this chapter, we first introduced a) SDN and its evolution, b) relevant technical details of forwarding devices and the SDN protocol OpenFlow and c) migration strategies and proposals for how to operate hybrid networks.

SDN and OpenFlow were proposed as a way to ease the management of network infrastructures and provide a way for experimenters to innovate using programmable network devices.

The first network controllers which were introduced offered basic features such as L2/L3 forwarding and load-balancing, but only worked with OpenFlow-enabled devices. The current generation of network controllers is more sophisticated. These controllers implement multiple north- and southbound protocols, enable network virtualization and provide APIs to interact with NMSs and orchestration systems. However, although they implement traditional southbound protocols, they do not implement any migration strategy and have limited awareness of the traditional devices. Furthermore, there still exists two entities, the controller and the NMS, with partly overlapping responsibilities. This makes it hard for network control applications to interact with heterogeneous and/or partially deployed SDN networks. NSALs could bridge the gap between management and control and provide a holistic view and control of heterogeneous networks. By providing a single abstraction and interface to network control applications, the control applications can evolve independently of the deployed devices or migration status. As of writing, the author is not aware of any proposals for an NSAL as described in [RFC7426]. The proposed NSAL design in this work is a first step towards a comprehensive cross-plane abstraction layer.

Chapter 3

Measuring Flexibility and the Impact of Reconfigurations and Traffic Policing on the Network and Applications

Designing a Network Services Abstraction Layer (NSAL) requires understanding of how traditional and Software-Defined Networking (SDN) devices in the network react to reconfigurations. Furthermore, application-awareness requires an understanding of in-network Quality of Service (QoS) mechanisms and their impact. The chapter at hand summarizes measurement results from experiments regarding the reconfiguration properties of devices and QoS mechanisms. The subsequent chapters of this thesis pick up on the measurement findings and incorporate them into the NSAL design and performance models. Figure 3.1 illustrates the three areas of focus of the measurements, *flexibility* in terms of timings, *data-plane interruptions* and *policing impact on transport and application layer*.

Flexibility / Timings In traditional networks, reconfigurations are rare and, as the measurement results will show, devices are not designed to be reconfigured often. For SDNs on the other hand, frequent reconfigurations are expected. Use cases such as short-term load-balancing or reactive control for new flow arrivals require frequent changes to the flow rules. Therefore, this chapter investigates the flexibility of forwarding devices. The flexibility is investigated in terms of reconfiguration characteristics of traditional and SDN-enabled devices in the network. We design a testbed where we measure the execution time of reconfiguration requests on different devices from multiple vendors.

Data-Plane Interruptions We argue that the reconfiguration frequency of SDNs and hybrid networks is higher than for the mostly static traditional networks. Hence, data-plane interruptions due to reconfigurations can become a challenge for the design of an NSAL. Interruptions result in lost packets, decreased QoS and might also trigger falsely fail-over

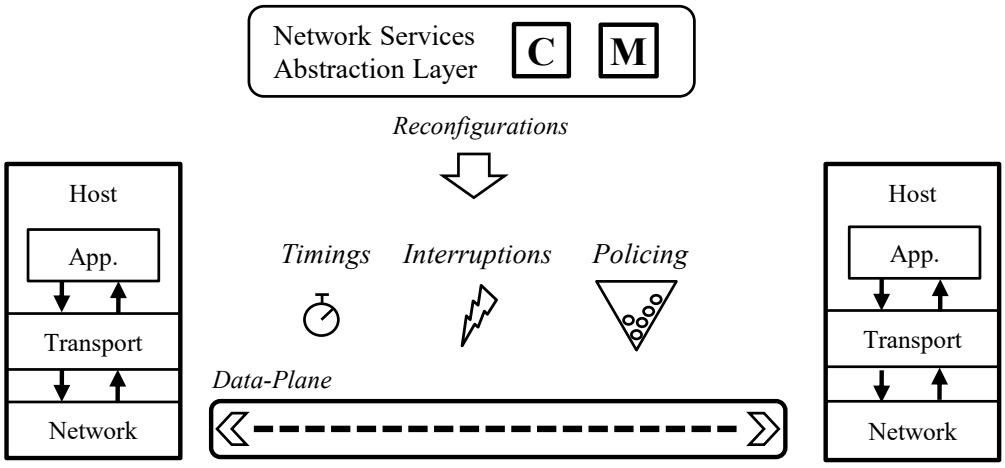


Figure 3.1: The goal of the measurements is (i) to quantify how long reconfigurations take to be completed, denoted as the *flexibility* of a device, (ii) to quantify the impact of reconfigurations on the data-plane in terms of interruptions of the Internet Protocol (IP) packet stream and to investigate the impact of the policing on the higher layers, i.e., on (iii) the transport and (iv) on the application layer.

mechanisms, depending on the duration of the interruption. In Section 3.2, we therefore measure the influence of the reconfigurations on a stream of packets traversing the device while the reconfigurations are triggered. In particular, we measure reconfigurations such as creating, modifying or removing Virtual Local Area Network (VLAN) tags from interfaces or changing the configuration of QoS mechanisms.

Policing Impact on Transport and Application Layer Application-awareness is often implemented by configuring traffic policing to limit an application's throughput at a point in the network. Studies show that application policing is wide-spread in the Internet [66], in particular in mobile networks and for video streaming applications. Therefore, in this chapter, we also take a closer look at how the configuration of traffic policing impacts the data-plane's ability to forward application-level data and how it ultimately impacts the applications itself. In particular, we want to answer the following questions: First, how does traffic policing configurations impact Transmission Control Protocol (TCP)-based application transmission streams? and second, how is the application ultimately impacted by the policing of the stream? We answer the first question by exploring the parameter space of the token bucket policing algorithm in combination with modern TCP algorithms in a testbed. The second question is answered by evaluation of a popular adaptive video streaming service for different policing configurations and measurements of the impact on application-specific metrics.

We first discuss the measuring of flexibility in Section 3.1. Afterwards, we discuss the impact of the reconfigurations on the data-plane in Section 3.2. Subsequently, we evaluate the impact of policing on the transport layer and on the applications in Section 3.3 and

Section 3.4, respectively. We conclude the chapter by summarizing and discussing the results in Section 3.5.

The content of this chapter is based on the results presented in the following publications. The measurements regarding the reconfiguration timings of traditional and SDN devices are presented in [15]. In [12] data-plane interruptions as the result of reconfigurations are investigated. [9] and [16] present the results of a large-scale measurement study of the effects of policing on video streaming. The measurements of the impact of policing on the transport layer are only published in this thesis as of writing.

[15] C. Sieber, R. Durner, and W. Kellerer. "How fast can you reconfigure your partially deployed SDN network?" In: *IFIP Networking Conference*. 9 pages. Stockholm, Sweden, 2017, p. 9. DOI: 10.23919/IFIPNetworking.2017.8264845.

[12] C. Sieber, A. Blenk, A. Basta, D. Hock, and W. Kellerer. "Towards a Programmable Management Plane for SDN and Legacy Networks." In: *IEEE Conference on Network Softwarization (NetSoft)*. 9 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502428.

[9] C. Sieber, A. Blenk, M. Hinteregger, and W. Kellerer. "The Cost of Aggressive HTTP Adaptive Streaming: Quantifying YouTube's Redundant Traffic." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 6 pages. Ottawa, Canada, 2015. DOI: 10.1109/INM.2015.7140478.

[16] C. Sieber, P. E. Heegaard, T. Hoßfeld, and W. Kellerer. "Sacrificing Efficiency for Quality of Experience: YouTube's Redundant Traffic Behavior." In: *IFIP Networking Conference*. 9 pages. Vienna, Austria, 2016. DOI: 10.1109/IFIPNetworking.2016.7497231.

3.1 Measuring Flexibility

We define flexibility of a forwarding device as the reconfiguration characteristics of the device and by the control delays associated with the reconfigurations. We consider three types of delay associated with the reconfiguration. Furthermore, we consider the duration the data-plane is blocked, i.e., is not forwarding data and packets are dropped. Figure 3.2 illustrates the types of delay we are interested in and the duration the interface is blocked. t_0 is the time the reconfiguration request reached the device under test. t^p is the time of acknowledgment (or processing time on device) of the reconfiguration request, subsequently referred to as reconfiguration time. Hence, t^p gives the duration the device does not accept a new reconfiguration until the previous one is finished. t^b is the blocking time, e.g., an interface restart as cause of atomic task execution. During this time, no data is forwarded on the specific interface. t^d is the time difference between task reception at the device (t_0) and the time the

effect is measurable on the data plane. For example in case of VLAN tagging, this is duration from the reconfiguration request to change the VLAN tag of an interface until the first packet leaves the device with the new VLAN tag. t^d, t^b, and t^p are relative to t_0 and there is no strict ordering between the types of delay, e.g., a device can acknowledge a change after or before the data plane effect is measurable.

To the best of our knowledge, there are no existing works about measuring management timings of traditional devices in the literature. There are several works providing measurements of SDN switches and also control plane delays such as in [80], or [108], [106], [162]. However, the provided measurements investigate the installations of batches of rules and not the installation of a single rule as in our scenario.

3.1.1 Testbed Set-up

Figure 3.3 shows our measurement setup. We use a tap device connected to the data plane and to the OpenFlow/management port. For SDN devices, we tap the OpenFlow controller port, for traditional devices we tap the TELNET connection to the management port. The tap is connected to a high precision measurement card with nanosecond precision. On the data plane, we generate a stream of marked User Datagram Protocol (UDP) packets with a packet size of 1400 Bytes, with constant inter-arrival time and with a data-rate close to the interface's line rate. We decode the incoming packets on the OpenFlow/management port and record the time of the last packet of the incoming reconfiguration command. That way we can identify the exact packet when the reconfiguration was applied (t^d) and compare it to the time of the reconfiguration command entering the device. Additionally, we record any periods of packet loss (t^b) and the duration until the interface acknowledges the processing of the command (t^p).

3.1.2 Results

We measure different SDN and traditional devices from multiple vendors, different years of release and sizes in our testbed. For traditional devices we include HP V1910 (for small

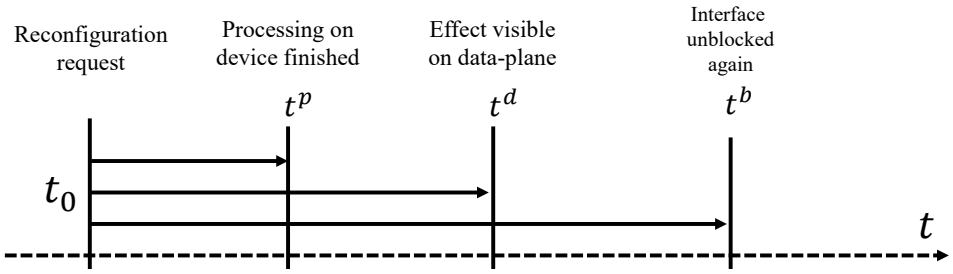


Figure 3.2: Processing time on device t^p, data-plane effect delay t^d and blocking delay t^b. All three times can be, but do not have to be, independent of each other, depending on the specific device.

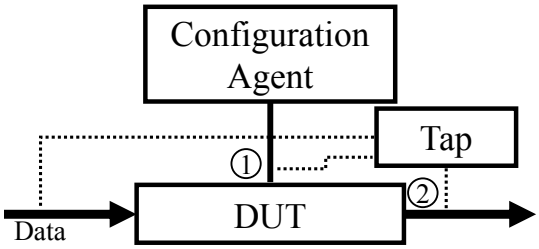


Figure 3.3: Measurement set-up consisting of a high-precision tap device, the self-developed reconfiguration agent and the Device Under Test (DUT) to be measured. Timing characteristics are measured by tapping the management port (1) and the data plane interfaces (2).

organizations, 2010) and Cisco Catalyst 4500 (for campus access & distribution, 2007). For SDN-enabled devices we include two OpenFlow hardware switches Pica8-P3290 and Pica8-P3297 (for small cloud data centers, 2012). Additionally, we include the NEC PF5240 (for data center, 2011) in the evaluation, which can be used in traditional and SDN-enabled mode, and the software switch OpenvSwitch (OVS) in SDN-enabled mode.

As measuring all possible combinations and parameters of all devices is infeasible, we focus for the measurements on the SDN transition strategy similar to Panopticon [110], which uses VLAN tagging to control traditional devices in a partial SDN deployment. For traditional networks, performing a VLAN reconfiguration is usually a text command sent via Command Line Interface (CLI)/TELNET to the switch. With the OpenFlow protocol, VLAN tagging can be set-up using OpenFlow flow modification messages. We use a custom OpenFlow Controller based on the libfluid [168] framework to populate the tables of the switches and install the rules that are measured. For the traditional devices, we use a custom configuration agent which accesses the devices via TELNET. Each measurement is repeated between 50 and 100 times.

3.1.2.1 Traditional Devices

Figure 3.4 shows the measurement results for the reconfiguration times (t^p) for the evaluated *traditional* switches using the command to add VLAN tagging to a port. The reconfiguration times (t^p) are shown as a box-plot with the median, 25 % and 75 % quartile and outliers. The results differ in two orders of magnitude from a median of 3.73 ms for the Cisco device to a median of 652 ms for the NEC. The HP device shows a median delay of 25.3 ms. The variance of the results is small for all three devices.

Table 3.1 gives a summary of the results for t^p, t^b and t^d of the measurement of the traditional devices for the VLAN tagging. The standard deviation is omitted in the table. For the NEC device, the standard deviation of t^b is about 20 ms, for the CISCO and HP device 1.4 ms. The results show that there is a significant variation between the three evaluated

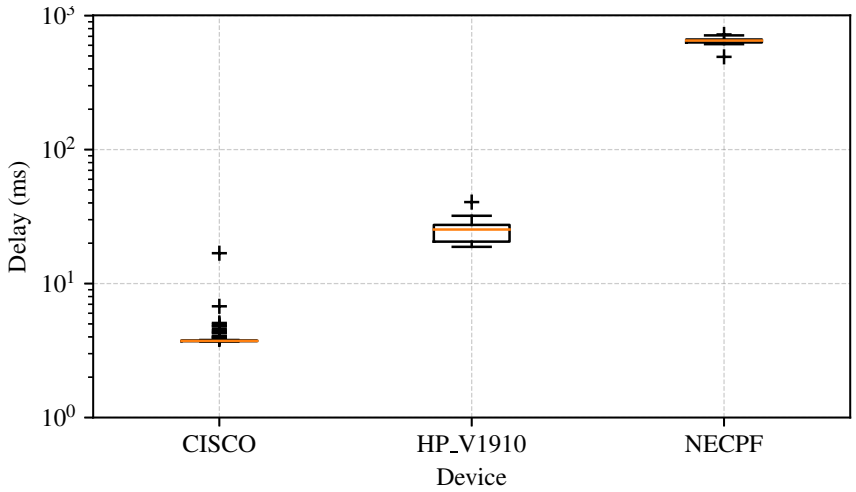


Figure 3.4: Reconfiguration times (t^p) for VLAN tagging via TELNET for the three traditional devices Cisco Catalyst 4500 (left), HP V1910 (middle) and NEC PF5240 (right). The Cisco device exhibits the fasted reconfiguration time with a median of 3.73 ms, followed by the HP device with 25.3 ms and the NEC device with 652 ms.

devices. For example, while the NEC device's Ethernet interface is unavailable for about 126 ms after the VLAN reconfiguration (t^b = 125.8 ms), the HP switch is available again after less than 1 ms.

3.1.2.2 SDN Devices

For measuring SDN devices there are more conditions to take into account. As other works have shown [106], the behavior of OpenFlow switches depends on the number of installed rules and the priority of the installed rules. Especially the priorities of the rules are important as higher priority rules mask lower priority rules and the switch has to make sure that the effects are independent of the order in which the rules are added. Therefore, for some cases, it can be necessary to search all current rules before the new one can be added. We measure the delay for four cases of flow table population: **None**, the flow table is empty. **Decreasing**, 1000 installed rules with decreasing priority, the measured rule has the lowest priority. **Increasing**, 1000 installed rules with increasing priority, the measured rule has the highest priority. **Same**, 1000 installed rules with the same priority, the measured rule has also the same as the others.

Table 3.1: Traditional devices VLAN tagging results

Switch	t^p	t^b	t^d
NEC PF5240F	648.9 ms	125.8 ms	316.2 ms
Cisco Catalyst 4503-E	4 ms	5.6 ms	8.1 ms
HP-V1910	24.4 ms	0.3 ms	15.2 ms

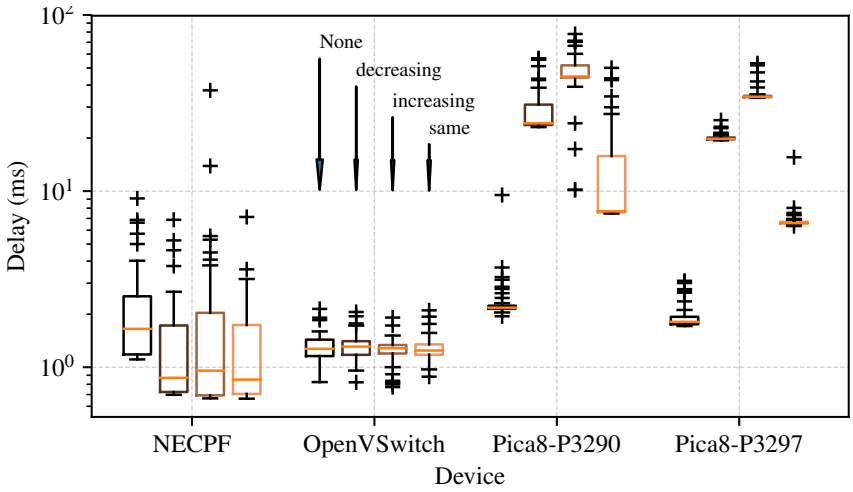


Figure 3.5: Rule push time for the four OpenFlow-enabled devices NEC PF5240, Pica8 P3290/P3297 and OVS. *None*, *decreasing*, *increasing*, *same* denote the number and priority order of pre-installed flows. OVS and NEC PF5240 can keep the rule push time between 0 ms and 1 ms for all four flow insertion scenarios. For both Pica8 devices, the time ranges between 1 ms for a previously empty flow table to 45 ms for 1000 pre-installed rules with increasing priority where the measured flow push has the highest priority.

Figure 3.5 shows the measured delays starting from pushing an OpenFlow rule that enables VLAN tagging until the first packet leaves the switch with a VLAN tag (t^d). Except OVS, the rule table population strategies affect each investigated devices differently. OVS behaves the same for all four strategies with a median delay of 1.27 ms to 1.31 ms. The approach used in OVS is described in [131] and uses atomic rules which avoid overlapping rules at all and therefore reduce the effects of priorities and population order to the rule update delay.

For the NEC, the delay varies between 0.85 ms and 1.65 ms. For the Pica8 devices we observe a delay of about 2 ms for an empty flow table. The delay increases for 1000 pre-installed rules with increasing priority. Here we observe a median delay of up to 34 ms and 45 ms for the two devices. This confirms the results from previous work that the population strategy is important. With the same priority the reconfiguration times are lowest while for the increasing priority the reconfiguration times increase. In general the variance of the configuration times using OpenFlow are much higher compared to the traditional reconfiguration through management commands. t^p was not measured in the testbed for OpenFlow devices.

3.2 Reconfiguration Impact on the Data-Plane

In the following we discuss the impact of the measured reconfigurations on the data-plane. In particular, we are interested in if reconfigurations cause any loss of packets. Any side-effects

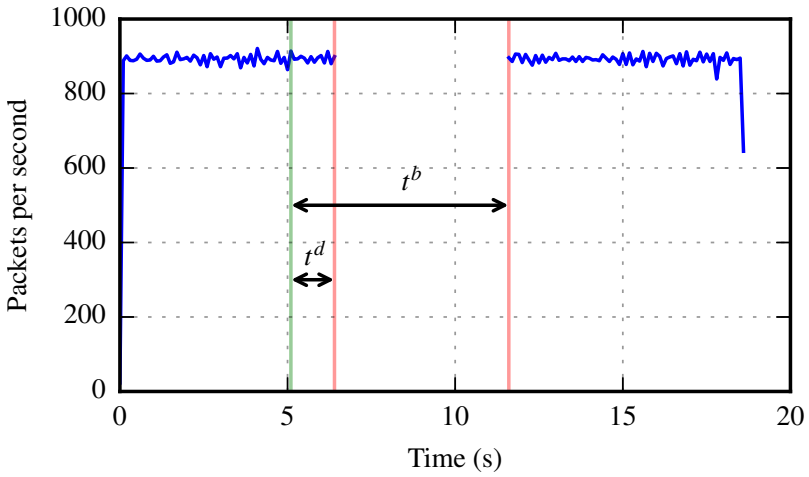


Figure 3.6: A packet stream traversing an interface of a NEC device while the processing pipeline, i.e., the QoS scheduler of the interface is changed. The reconfiguration triggers the restart of the interface which results in dropped packets during the restart process. The duration between the device accepting the reconfiguration and the interface restart t^d is about 1 s. The blocking duration t^b (which includes t^d), during which all packets are dropped, is on average about 6 seconds.

of reconfigurations are important in the design of a NSAL, as negative effects on the data-plane can severely limit the number of reconfigurations a NSAL can issue to the devices.

Figure 3.6 illustrates a measurement where we select an alternative path in the processing pipeline, i.e., changing the QoS scheduler of the NEC device in our testbed. At $t_0 + t^t$ with t^t being zero, the switch receives the configuration command through the management interface. As we define the blocking delay t^b and the processing on device t^p independent of each other, t^b is the time from $t_0 + t^t$ until the interface is up again and transmits data. For this task and device type, t^b is about 6 seconds on average. t^p is the time between the reception of the configuration command and the switch's confirmation.

An automated management system may want to frequently change the scheduler, depending on the current situation. The measurements show that this is not possible with the evaluated state-of-the-art hardware in our testbed. Hence, the design of the NSAL has to consider mechanisms for detecting potential negative effects of management actions. For example, the NSAL can drain the traffic on the interface before issuing the reconfiguration to prevent any loss of packets during the reconfiguration.

3.3 Impact of Policing on Transport Layer

Policing is the most wide-spread used mechanism for reconfiguring the network to control application throughput, as it does not require buffer space in the network elements for packets exceeding the configured rate [66]. In following, we evaluate traffic policing at a device as a

mechanism for a NSAL to limit the amount of resources assigned to a network application. In particular, we discuss the effects of policing on the dominant transport protocol TCP. TCP behavior is dictated by the congestion algorithm in use. *CUBIC* [78] is the default algorithm configured on modern Linux-based (kernels 2.6.19 and above) systems. Furthermore we evaluate *BBR* [37], a recent congestion control proposal by Google and *DCTCP* [23], the default algorithm on use in Windows Server systems. Additionally, we evaluate *QUIC*, recent proposal by Google as Hypertext Transfer Protocol (HTTP) transport alternative which is based on UDP instead of TCP.

The measurement set-up consists of three nodes. One node with a simple HTTP/TCP server. One node with activated Token Bucket (TB) filtering based on Linux Traffic Control [185] and one node with a simple HTTP download client. We investigate three parameters. The policing rate and burst size configured at the traffic filter and the TCP algorithm configured at the server. Figure 3.7 illustrates the relative download duration of a 10 MB file. The relative download duration is defined as the ratio between the recorded download duration and an idealistic download duration based solely on file size and policing rate ($\frac{FileSize}{PolicingRate}$). The relative download duration is shown for different policing rates (in Kilobits per second) and different burst sizes (in Kilobit). Values > 1 indicate that the policing rate disturbed the congestion control and the download takes longer than the policing rate would allow. Values close or equal to 1 show that the TCP sending rate equals the policing rate and no lost packets are observed. Values < 1 are possible due to the TB policing which allows burst of packets, e.g., at the beginning of the transmission when the bucket is full. Note that the absolute values of the relative download time are only valid per algorithm and can not be compared across the algorithms. Also note the different scale for *BBR* compared to the other three algorithms. The different behavior can be explained by the different congestion control approach taken by *BBR*. While the other algorithms base their sending rate calculation solely on observed packet loss, *BBR* tries to estimate the available rate based on delay and packet loss. For some combinations of policing rate and burst size, *BBR* misjudges the available throughput and throttles the sending rate unnecessarily.

The figures 3.7a to 3.7d show the results for *BBR*, *Cubic*, *DCTCP* and *QUIC*. Multiple observations can be made from the figures. First, *Cubic*, *DCTCP* and *QUIC* are affected by policing in a similar manner. For the three algorithms the policing rate has no visible impact on the relative download duration. Instead, the duration decreases for larger burst sizes. An decrease for larger burst sizes is expected as this allows TCP to exceed the policing rate longer at the beginning of the transmission. *BBR* exhibits different behavior. The download duration increases for higher policing rates and lower burst sizes. Furthermore, while the maximum

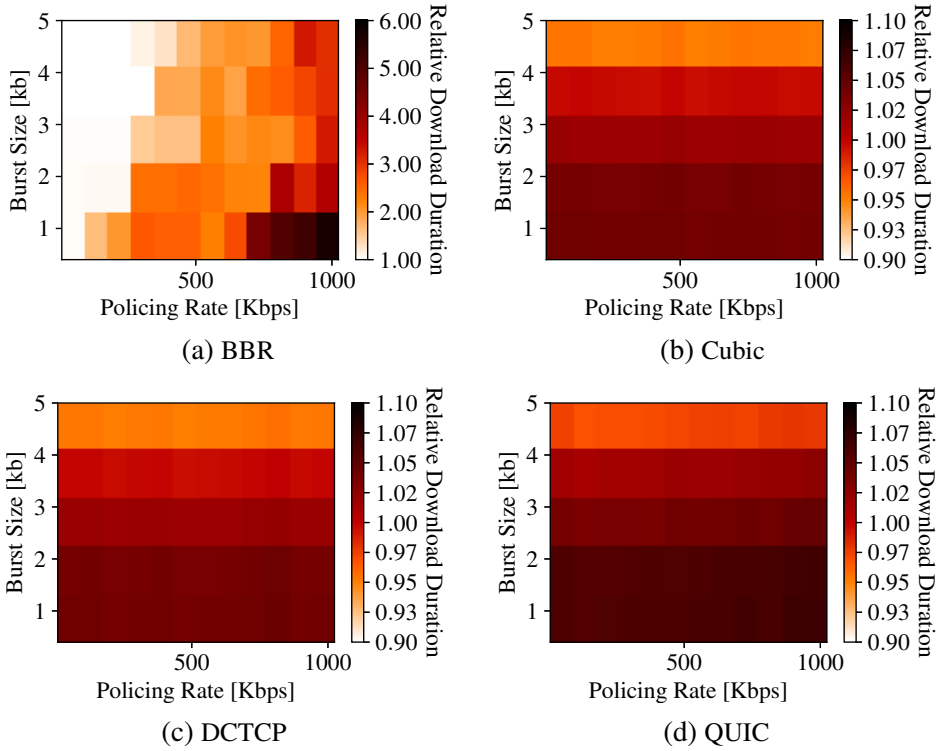


Figure 3.7: Impact of policing on TCP *Cubic*, *BBR*, *DCTCP* and *QUIC* in terms of the download time relative to an idealistic download duration based on the burst size and policing rate. While *Cubic*, *DCTCP* and *QUIC* exhibit similar behavior (download duration dictated by burst size), *BBR* is influenced by both parameters and in general shows longer download durations.

relative download duration for the other algorithms varies between 5 % and 10 %, *BBR* exhibits a download duration of up to six times the idealistic rate.

In a nutshell, the results show that policing negatively impacts the transport layer's ability to determine the available throughput, resulting in lost packets and unnecessary throttling of the sending rate. Hence, to implement application-awareness in the NSAL, policing in the forwarding nodes of a network should be avoided. Next, we investigate the negative impact of policing on the layer above the transport layer, the application layer.

3.4 Impact of Policing on Application Layer

In the preceding section we discussed the impact of policing on TCP streams in the network. The results there show that policing leads to packet loss and inability to estimate the available throughput. For the design of an application-aware NSAL it is important to further understand the impact of policing on the applications on top of TCP. In this section, we take a look at policing of the most widespread Internet video application (YouTube) and how it effects

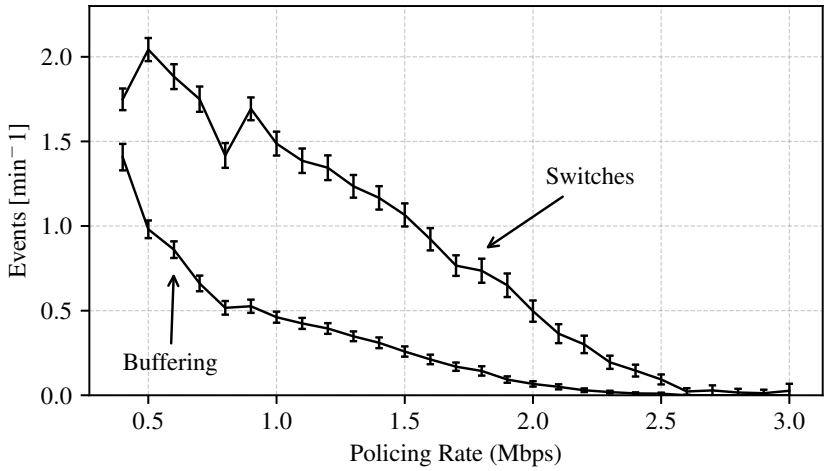


Figure 3.8: Impact of policing on the performance of YouTube's adaptation algorithm. f denotes the configured policing rate. The left axis denotes the frequency of buffering and switching events during the playback with the configured policing rate. The figures shows the negative influence of policing on the playback quality adaptation, as the adaptation it not able to prevent switches and buffering for policed rates below 2.5 Mbps.

the performance of the application. We first describe the experiment set-up. Afterwards we discuss the selected video content for the experiments and present the results.

The experiment set-up consists of a virtual machine *Xubuntu 14.04 64-bit* running a browser (*Firefox 21*) with the YouTube player, an Secure Hypertext Transfer Protocol (HTTPS) proxy inside the virtual machine and a virtual network which limits the available throughput by TB filtering. The set-up is connected to the Internet through a lightly utilized lab network and through the university's Internet connection. We randomly select videos with a duration of 1, 2, ..., 10 minutes with an allowed deviation of 5 seconds. In total, 35 different videos were accessible during the whole experiment time and are included in the evaluation. Average duration of selected videos is 5.3 minutes. The average bit-rate for each quality level is 0.11 Mbit/s for 144p, 0.24 Mbit/s for 240p, 0.37 Mbit/s for 360p, and 0.72 Mbit/s for 480p.

Next, we discuss the result of the experiment. Figure 3.8 shows the buffering rate and the switching rate for the different policing rates and averaged over all videos. The error bars indicate the 95 % confidence interval of the measurement points. The figure shows that the buffering rate decreases non-linear with increasing policing rate, while the corresponding decrease in quality switching rate is approximately linear. At the lowest evaluated bandwidth 0.4 Mbps, we observe an average buffering rate of 1.4 [min^{-1}]. The quality switching rate is on average between 1.75 [min^{-1}] and 2.0 [min^{-1}] for 0.4 Mbps to 0.5 Mbps. At about 2.6 Mbps, the three metrics reach their minimum of zero for the buffering and switching events. We conclude that for a bandwidth of 2.6 Mbps all videos in our result set are, on average, played

back without buffering events and quality switches. Furthermore we see that switching events are more frequent than buffering events and decrease slower for increasing policing rate.

In a nutshell, the negative impact of policing on the transport layer carries on to the application layer for the investigated use case of adaptive video streaming. The inability to estimate the available throughput accurately leads to unstable adaptive behavior which manifests in the shape of frequent quality switches and buffering events for the user.

3.5 Discussion

In the following, we discuss the measurement findings and relate the results to the following chapters of the thesis. Regarding the flexibility, the measurements answer the following two questions. First, how long does it take a device to implement a reconfiguration request into the network? and second, does the reconfiguration cause any undesired negative effects on the data plane?. The measurements regarding the flexibility show that even though the devices offer well-defined functionality, e.g. VLAN tagging, and well-defined forwarding speeds on the data-plane, e.g., 1 Gbps and 10 Gbps, the timing characteristics of reconfigurations vary significantly. The results show that all OpenFlow devices in our testbed allow fast reconfigurations, where the exact duration depends on the number of previously installed rules. The reconfiguration times of the traditional devices on the other hand vary between less than 1 ms and up to 650 ms. The measurements emphasize the need for a sophisticated reconfiguration management with detailed and device-specific reconfiguration models. A NSAL has to be aware of how long a device needs for a reconfiguration to prevent routing black holes or routing loops in the network. Furthermore, the NSAL has to know how often a device can be reconfigured per time interval to prevent network applications from overloading the network with reconfigurations and to define sensible rate-limits on those applications.

Regarding the negative side-effect for both device categories, the measurements results show no blocking duration t^b for the evaluated SDN devices. Hence, no packets are dropped as a consequence of a reconfiguration via OpenFlow. For the traditional devices, the blocking time ranges from about less than 1 ms to 126 ms. The longest blocking duration can be observed for changing the QoS scheduler, where we observe blocking durations up to 6 s. The measured blocking durations up to 6 s are an previously overlooked challenge for an NSAL. Network control applications unaware of this problem may choose to frequently change the packet scheduling, depending on the current situation in the network. However, this can lead to frequent data-plane interruptions and must be prevented by the NSAL or at least disclosed to the network application requesting the reconfiguration.

In order to evaluate policing as mechanism for application-awareness in the network, we asked the following questions. First, how does traffic policing configurations impact TCP-based application transmission streams? and second, how is the application ultimately impacted by the policing of the stream? Looking at the measurements regarding the impact of configuring policing in the network on the TCP streams and on a popular video application, following conclusions can be drawn. First, policing interferes severely with TCP congestion control algorithms, resulting in unnecessary retransmissions of data packets. Second, due to the unstable throughput caused by the policing, adaptation algorithms on the layer above the transport protocol, e.g., HTTP/HTTP Adaptive Streaming (HAS) in case of YouTube, are failing to estimate the available throughput accurately. The algorithms then make suboptimal adaptation decisions resulting in playback stalling and frequent quality switching events. In a nutshell, configuring policing in the network has significant negative effects and should be avoided in combination with current congestion control algorithms.

Chapter 4

Design of a Network Services Abstraction Layer for Partially Deployed Software-Defined Networks

With Software-Defined Networking (SDN), a set of standardized interfaces emerged, e.g., OpenFlow and Forwarding and Control Element Separation (ForCES), to control the forwarding behavior of network elements. Despite research and standardization efforts, the management plane is still eluding an equivalent device- and vendor-neutral programmability. Thus, innovation in the management plane is hampered by a dependency on human experts, domain knowledge that is hidden in human-centered manuals, and the huge amount of diverse device capabilities and configuration interfaces.

Accordingly, in this chapter we propose a Network Services Abstraction Layer (NSAL) architecture that provides a unified interface to the control and management plane of heterogeneous devices, i.e., SDN and traditional devices with an enriched information model which incorporates the insights gained from the device measurements in Chapter 3. We discuss the properties of the chosen level of configuration abstraction and show how applications northbound of the abstraction layer are well prepared against undesired side-effects of management actions. By example of a popular approach for enabling OpenFlow in mixed-SDN/traditional networks based on Virtual Local Area Network (VLAN) tagging, i.e., Panopticon [110], we provide a proof-of-concept implementation of the proposed architecture in a test-bed. VLAN tagging is a feature available on most managed traditional devices and therefore a likely candidate for a migration strategy in heterogeneous environments.

As the measurements in Chapter 3 highlight, there is a need to understand timing characteristics of the devices in the network and potential side-effects such as unexpected data-plane interruptions for certain reconfigurations. We show how a management application can use

the abstraction layer to discover and configure Quality of Service (QoS) options in the network, monitor the devices, and prevent undesired traffic interruptions in the traditional domain.

Section 4.1 gives a detailed definition of the problem and challenges the proposed NSAL is solving. Section 4.3 introduces the architecture, components, design trade-offs, monitoring and the device models of the NSAL in detail. Section 4.4 discusses how the timing informations gained by the measurements allow to calculate optimal reconfiguration orders for multi-device reconfigurations. Section 4.5 evaluates the proposed NSAL by two use cases from the domain of the management of partially deployed SDNs. Section 4.6 summarizes the chapter and puts the NSAL in the context of the subsequent chapters of this thesis.

The content of this chapter is based on the results presented in the following publications. [12] presents the design of an NSAL following the [RFC7426] proposal. Furthermore, the two investigated use cases are discussed and the Domain Specific Language (DSL) is introduced. The architecture is implemented and for a special use case demonstrated in [10].

[12] C. Sieber, A. Blenk, A. Basta, D. Hock, and W. Kellerer. "Towards a Programmable Management Plane for SDN and Legacy Networks." In: *IEEE Conference on Network Softwarization (NetSoft)*. 9 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502428.

[10] C. Sieber, A. Blenk, D. Hock, M. Scheib, T. Hohn, S. Kohler, et al. "Network Configuration with Quality of Service Abstractions for SDN and Legacy Networks." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2 pages. Ottawa, Canada, 2015. DOI: 10.1109/INM.2015.7140446.

4.1 Design Challenges and Problem Definition

Figure 4.1 introduces the abstract reference architecture for an NSAL. The reference architecture is proposed in the informal Request for Comments (RFCs) 7426 as *Software-Defined Networking (SDN)*: *Layers and Architecture Terminology* [RFC7426]. The figure shows a partially deployed SDN network where some of the devices are already migrated to SDN-enabled devices (Ⓢ). Traditional/legacy devices are denoted by (Ⓛ). The NSAL is located on top of the network and consists of a control module for the SDN-enabled devices and a management module for the management plane of both types of devices. The NSAL provides an interface to the network orchestration and network control applications which allows to request the topology and current state of the network, including monitoring information, and to initiate reconfigurations to the network.

The RFCs does only provide the general architecture and does not go into details on how to programmatically combine the control- and management-plane. Questions like, which

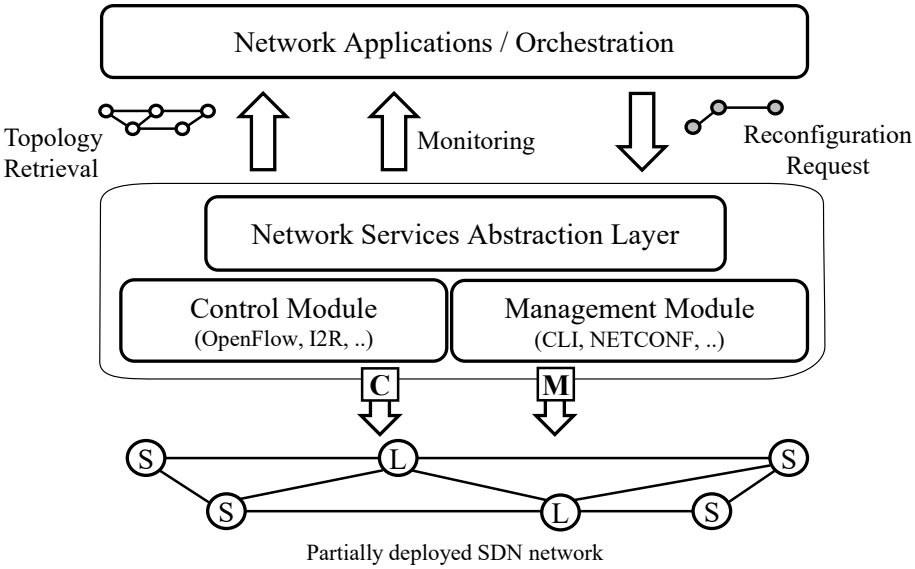


Figure 4.1: A NSAL as defined by RFC7426: Software-Defined Networking (SDN): Layers and Architecture Terminology [RFC7426] is a layer consisting of a control and management module which provides northbound control applications a unified interface for reconfigurations and monitoring.

packet scheduling strategies are supported by a particular device?, or how long does it take until a particular reconfiguration is applied to the data-plane?, can only be discovered by studying the device's handbook or conducting measurements. Recent proposals for vendor-neutral management data-models, such as OFCONF, are still lacking majority and do not provide a standardized way of device capability discovery, device status monitoring, timing characteristics and side-effect discovery. For SDN-enabled devices, the control protocols such as OpenFlow or ForCES offer a more detailed and standardized data-model than found in traditional devices. However, those protocols do not offer timing information or QoS feature discovery and especially in the case of the wide-spread OpenFlow, a particular device does not have to support all standardized features. Furthermore, it is unclear which reconfigurations and features trigger fast-path processing and which slow-path processing on a specific device. Next, we discuss the key challenges when designing an NSAL for partially deployed SDN networks. The described challenges C2.1 - C2.6 are sub-challenges of *C2: Design of an Abstraction Layer for Partially Deployed Software-Defined Networks*.

4.1.1 C2.1: Different Control and Management Data-Models

While communication protocols and forwarding behavior of network elements are highly standardized, the management and control data-models of the devices are less standardized. This presents a key challenge for any NSAL which wants to control a heterogeneous network. Challenge C2.1 is therefore the design of an abstraction layer considering different control and

management data-models. Section 4.3.4 tackles this challenge by proposing a graph-based representation of devices' packet processing capabilities. That way a control application can traverse the graph to find the desired features and configuration options.

4.1.2 C2.2: Level of Abstraction

There is a lack of research in the literature about which level of abstraction is the most suitable one for an NSAL. Is it better to include as many parameters and exceptions from all the different devices in the network into the NSAL data-model or is it better to design it light and try to find the best parameter settings for each device automatically? A more detailed data-model gives network applications on top of the NSAL more freedom in adjusting device parameters. However, a very detailed model results in an increase in complexity for each of the network applications, but the gain of adjusting every device's parameters is not clear and requires research for every device and use case. A slim data-model enables rapid development of network control applications. However, the NSAL has to infer proper default values for parameters not visible to the applications, which might result in non-optimal parameter choices. Challenge C2.2 is therefore the selection of a suitable level of abstraction. The challenge is tackled in Section 4.3.2 by analyzing the trade-offs and defining a sensible level of detail.

4.1.3 C2.3: Different Reconfiguration Delays

Especially traditional devices are not designed to be reconfigured fast or often. Hence, configuration interfaces are often text-based and not streamlined for efficient processing of reconfiguration requests. This is confirmed by the measurements in Chapter 3 which show that reconfiguration delays differ greatly between different devices. Two questions arise from this. First, how to provide the reconfiguration delay information to the network applications and that way enable timed reconfiguration execution? Second, how to prevent network applications from accidentally or deliberately overloading a slow device in the network? Challenge C2.3 is therefore the design of a mechanism for synchronizing reconfigurations across devices and preventing device overload. The challenge is tackled in Section 4.4 of this chapter. Furthermore, Chapter 5 takes a closer look at the theoretical problem of having devices with different reconfiguration delays in a network.

4.1.4 C2.4: Non-inferable Reconfiguration Side-Effects

Section 3.2 shows how reconfigurations can trigger periods of packet loss on a network interface of up to 6 s long. The knowledge of these side-effects is only known to domain experts and poses challenge C2.4 for the design of an NSAL. An NSAL has to be aware of these side-effects

and either deploy automated countermeasures, e.g., temporary rerouting traffic, or inform applications north of the NSAL about the side-effects of their desired reconfigurations. The challenge is tackled in Section 4.4 by proposing a methodology for exposing the side-effects to the northbound control applications.

4.1.5 C2.5: Monitoring of Heterogeneous Devices

As there is no unified interface to the heterogeneous devices, monitoring the devices also poses a challenge for the NSAL. In order for the NSAL to provide programmable access to the network for northbound control applications, providing the current state of the network as part of the NSAL interface is necessary. Most devices support the Simple Network Management Protocol (SNMP) monitoring protocol, but every device and vendor supplies their own Management Information Base (MIB). The MIB describes which metrics of a device can be monitored and how they are accessed. OpenFlow-enabled devices can be monitored through the OpenFlow-protocol which supports basic counters per flow and per interface. Hence, the NSAL has to provide a unified way for northbound applications to collect long-term metrics from OpenFlow-enabled and traditional devices and on-demand real-time monitoring as part of a programmable interface. Challenge C2.5 is therefore the design of a suitable monitoring subsystem for the NSAL. Section 4.3.3 proposes a suitable and flexible monitoring subsystem for hybrid networks based on distributed configuration agents and dynamic monitoring task execution. The monitoring design is evaluated for two selected use cases.

4.1.6 C2.6: Performance and Reliability

Introducing the NSAL as a single unifying layer between devices and network management applications raises concerns about the performance and scalability of this layer. The layer has to process each reconfiguration with a low delay and must support multiple concurrent reconfiguration requests. Furthermore, it should offer a unified view and thus be logically centralized, but physically distributed to prevent a single point of failure and ensure scalability for scenarios with a high reconfiguration load. These aspects of the NSAL design, scalability, load-balancing, distributed operation and reliability are out of scope of this thesis. However, in Chapter 5 we tackle the theoretical reconfiguration performance limits of hybrid networks and in Chapter 6 we study the problem of determining the performance limit of an NSAL instance at run-time.

4.2 Background

Network management in general has received a lot of attention in the past and in the present and is comprised of different topics. This chapter can be seen in the context of configuration management, e.g., setting configuration options, monitoring, e.g., on-demand monitoring tasks, and device capability discovery, e.g., QoS options. In the following, we introduce publications most relevant to the topics of this chapter.

The ForCES protocol [RFC5810], which is positioned as alternative to OpenFlow, shares similar concepts with the chapter at hand. In ForCES, the hardware of the forwarding elements, e.g., switches, are modeled as processing pipelines of so called *Logical Function Blocks* which can be discovered and configured. In this chapter, we aim for less complexity and model the switches configuration interfaces as easy to use elements and abstract many low-level details of ForCES. At [179], an informal working group of large network operators headed by Google is working on transferring software-defined principles to the management plane. This could greatly simplify the implementation of the device-specific modules in our configuration agents. OFCONF [180], a data model for NETCONF, is the counterpart of OpenFlow for the management plane. However, OFCONF is still new and the provided data models are of limited scope. Furthermore, it is tailored to the specific needs of OpenFlow and does not consider switch features outside of the scope of OpenFlow, e.g., input shaping. In [180], the Open Networking Foundation (ONF) is working on a Core Information Model (CIM) which specifies physical, logical and virtual switch components, relationships and protocols in great detail. Our abstraction aims for a representation closer to the physical switch and does not consider higher layer relationship between protocols. The defined relationship in CIM can be implemented on top of our NSAL. In [188], *OASIS* is working on a Topology and Orchestration Specification for Cloud Applications (TOSCA). However, the abstraction focuses on higher level network services such as Database Management System (DBMS) and their relationship to other entities, not on details of the individual forwarding elements.

In [49], the authors describe how domain knowledge is required for network configuration, but hidden in domain experts and human-centered switch manuals, thus, inaccessible for network automation. The authors introduce *COOLAID*, an interface similar to a database Application Programming Interface (API) to create a logically centralized abstraction of the network configuration. In [25], the authors argue that the management plane is too complex due to devices exposing all their internal details and parameters. This leads to error-prone configuration, fragmentation of management tools, and hard to understand configuration parameters. They introduce *CONMan*, an abstraction layer which exposes device configuration with inter-connected protocol configuration modules and dependencies. In [50], the authors introduce *PACMAN*, a platform for automated operation and configuration management. The

work defines active documents, which describe an abstract configuration task. One active document represent higher-level abstractions, spanning multiple actions and one or more devices. The work represents a vertical subset of the NSAL in our work, but designed without northbound interface and focused on composed atomic tasks. In [47], the authors introduce *SWItch*, a framework for the management of data center networks. *SWItch* uses namespaces trees, similar to our switch component graph, to model the devices. *COOLAID*, *CONMan*, *SWItch* and *PACMAN* are designed to be operated by humans and to be responsible for the management of the traditional control algorithms. This differs from our work as we see and design the management plane as a building block underneath a network services abstraction layer tailored for automation. Furthermore, monitoring in the abstraction layer and the management task timings in terms of delay and blocking are not part of their work.

In [147], the authors depict *Statesman*, a network-state management service deployed in the Microsoft Azure cloud. *Statesman* offers a graph abstraction northbound and is designed to resolve conflicts between different management applications accessing the graph. Compared to our work, the abstractions are not as detailed, e.g., no QoS support, and the focus is on network states instead of atomic tasks. Domain knowledge regarding the cost of change, i.e., the blocking time due to configuration change, are not available to northbound applications.

In [102], the authors conclude that monitoring the frequency of atomic management tasks, e.g., configuration of a VLAN on an interface, can be used to classify seldom touched configuration options as important and dangerous. This could be an extension to the estimation in the abstraction layer introduced in our work. A way of automatically learning the capabilities of a certain device is introduced in [119]. The authors show how an ontology-based information extraction system can deduce the capabilities of a device by analyzing the Command Line Interface (CLI) of the device. This could allow rapid prototyping of the device models discussed in our work. In [114], the authors introduce *Hybnet*, a network manager for a hybrid SDN/traditional networks. The work is related to the second use case in our work and to the *Panopticon* approach. We see this as complementary to our work and as part of the NSAL. The same applies to [95, 165], where the authors show OpenFlow agents and control for traditional devices.

4.3 Proposed Design of a Programmable Abstraction Layer

In the following, we propose a reference architecture for a programmable abstraction layer considering the challenges C2.1 - C2.5 as defined above. The NSAL and a DSL are the key elements of this architecture. The NSAL provides a unified interface to configuration and monitoring of the underlying network. Furthermore, it provides a stateless Hypertext Transfer

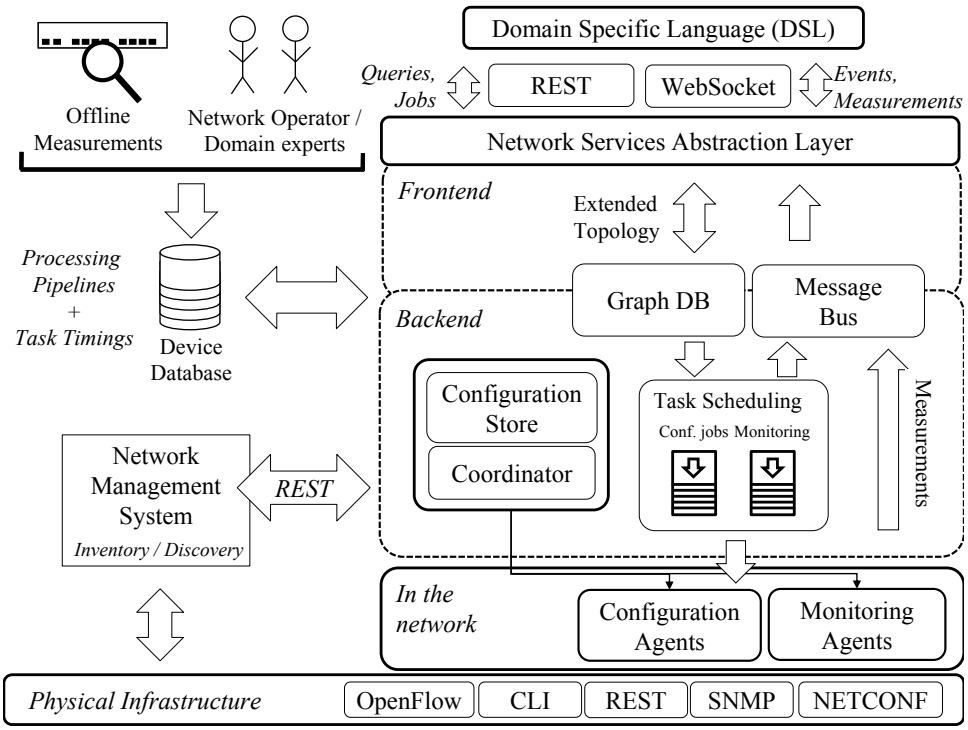


Figure 4.2: The proposed architecture for an NSAL. The northbound-facing front-end consists of a synchronous REST and asynchronous WebSocket interface. The back-end consists of a messaging bus, distributed agents in the network and a configuration task queue. A custom device database stores the devices timings and processing pipelines models. A proprietary Network Management System (NMS) provides discovery and inventory services via a REST interface.

Protocol (HTTP)-based northbound Representational state transfer (REST) interface and a WebSocket-based interface for asynchronous events, e.g., measurements and task completion events. The DSL is a Python dialect tailored to the NSAL. It simplifies the programming for northbound applications and coordinates the request/response model of the REST interface with the asynchronous events of an event bus. Next, the architecture is introduced in detail.

4.3.1 Architecture

Figure 4.2 illustrates the proposed architecture for an NSAL. In order to support the NSAL, multiple components are required southbound. In the network, distributed configuration and monitoring agents execute configuration tasks, e.g., setting VLAN tagging through a CLI, and gather device statistics such as device ports. Each agent is responsible for one or more devices and uses device-specific modules to interface with the heterogeneous devices. Configuration tasks and monitoring configuration are distributed among the agents through a task queue and operational configuration is stored in a key-value store. A logically centralized event bus acts as a broker for measurements and task completion events. The device database provides a

model of the capabilities and the processing pipelines of a specific device type. A custom graph database stores the high-level network topology and the processing pipeline of the devices and makes it available through the NSAL. For the discovery of the network infrastructure, including the inventory with the vendor and the model of each networking device, we utilize a commercial NMS.

We implemented our proposed architecture relying on several open source projects, most notable Apache ZooKeeper [190] as configuration store, Apache Kafka[169] and crossbar.io [172] as backend and frontend messaging bus, and RabbitMQ [183] as messaging queue. For topology discovery and device inventory, we use the proprietary NMS StableNet [177].

4.3.2 Design Trade-offs

Next, we discuss the design trade-offs of the NSAL and its provided data model as a response to the design challenge C2.2, the level of abstraction problem. The key metrics are the *model fidelity* and the *practicality* as shown in Table 4.1. The data model fidelity describes how many details of the heterogeneous configuration interfaces and hardware features of the different devices should be exposed to the northbound management application. Practicality describes the qualitative result of the "usefulness" of a combination of the three metrics *NSAL-intelligence*, *northbound-intelligence* and *hardware feature utilization*.

First, we comment on the case of a low model fidelity. A low model fidelity requires the NSAL to make more decisions on its own, as many parameters are hidden from the northbound application. This is comparable to an intent- or policy-based interface where the NSAL receives abstract requirements and translates them to device-specific configurations. The metric *NSAL intelligence required* behaves inverse to the model fidelity. A low model fidelity requires the NSAL to make decisions on its own about parameters not visible to the northbound application. A high model fidelity requires less intelligence in NSAL, as most decisions about parameters are the responsibility of the northbound control application. This is comparable to a human-centric network management where a domain expert writes custom configuration scripts for each device.

Table 4.1: NSAL Design Trade-Offs

Model fidelity →	low	medium	high
NSAL intelligence required	high	medium	low
Northbound intelligence required	low	medium	high
Device modeling effort	low	medium	high
Hardware feature utilization	low	medium	high
Practicality →	low	high	low

From the point of view of the modeling effort per device type, a low fidelity NSAL design requires the least modeling effort, as one has to create a basic common model for all switching devices only, e.g., to distinguish OpenFlow from traditional devices. However, a low model fidelity is not able to fully leverage all features of the hardware. For example, the OFCONF management data model for OpenFlow does only support data-rate based scheduling, even if the hardware could support mechanisms like Priority Queuing (PQ) or input queuing. Furthermore, specific hardware features outside of a simple model cannot be considered from the global perspective of the northbound management application.

A high model fidelity in turn only requires minimal NSAL intelligence. In the best case, one northbound request translates exactly to one required configuration change, e.g., changing a weight of a queue. On the other hand, this increases the complexity of the decision logic in the NSAL, as it has to be able to handle many options and also device-specific exceptions and limitations, e.g., one feature blocks another switch feature. Then again, a high model fidelity in combination with a complex management module algorithm allows to globally utilize a large portion of the features of specific device types, e.g., all the different scheduling strategies, with the cost of a high modeling effort per device type.

The design of our NSAL aims for a medium practicality. Whenever possible, we design the NSAL in a way that one northbound change requests translates to one southbound configuration job. Furthermore, we do not expose all hardware features of the heterogeneous devices to the management module to facilitate rapid development of novel management plane algorithms. Device-specific features and parameters not covered by the NSAL are silently set to meaningful default values by the configuration agents, still based on hand-crafted rules by domain experts.

4.3.3 On-Demand and Long-Term Monitoring

In following we describe how the proposed NSAL tackles design challenge C1.5, the on-demand and long-term monitoring of heterogeneous devices. We see network monitoring in the NSAL mostly as an enabler for northbound applications to verify the configuration. Hence, traffic engineering techniques, e.g., load-balancing, are not in the focus, but possible with the provided primitives. Figure 4.3 depicts the two modes of monitoring supported by the platform, namely task-based on-demand monitoring and continuous monitoring. Both modes are defined based on the unified NSAL specification, but executed in the context of the device-specific modules in the monitoring agents. The choice of how to implement the monitoring, e.g., SNMP traps or CLI polling, is up to the device-specific module and based on human domain experts implementation.

The **on-demand monitoring** creates monitoring tasks consisting of the device and component to monitor, e.g., interface X of device A, the metric to monitor, e.g., interface status or

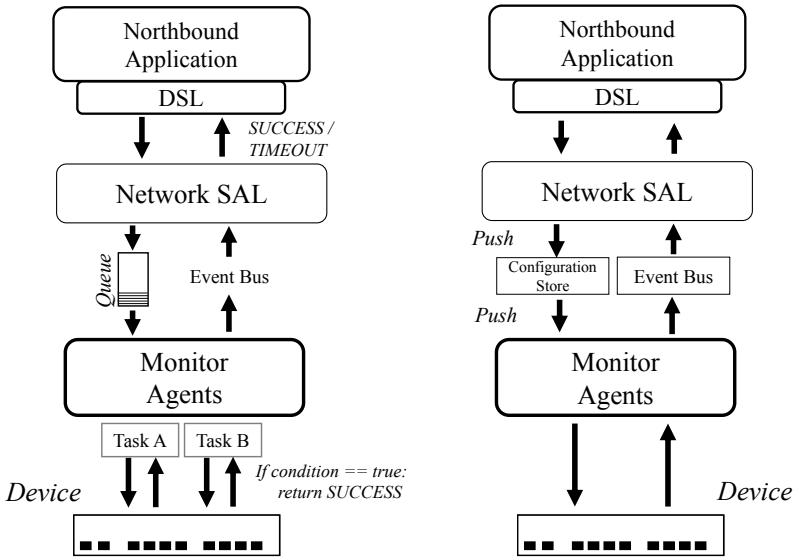


Figure 4.3: The two modes of monitoring supported by the proposed architecture. On-demand monitoring executes arbitrary monitoring tasks close to the device with high frequency and defined stop criteria. Long-term monitoring data configuration is pushed in the configuration store and from there to the agents.

received bytes, a condition which terminates the task, e.g., average of received bytes samples is smaller than 1000 Bytes, and a maximum execution time of the whole task. The condition is expressed as arbitrary (Python) code that is executed in an isolated execution environment with access to numerical statistic programming packages and the collected samples. On-demand monitoring tasks allow for high frequency monitoring without burdening the northbound interfaces with unnecessarily frequent samples and are designed for distributed synchronous operation (*do - wait for condition - continue*).

The **continuous monitoring** mode is designed for long-term data acquisition and asynchronous events. Continuous monitoring is not expressed as a monitoring task, but as a permanent configuration in the configuration store. The application on top of the NSAL can set an interval and optional threshold for all metrics defined in the NSAL. The configuration is saved to the configuration store, which triggers the monitoring agent to update its local configuration. If the metric exceeds the threshold, the agents send the measurement sample to the message bus where the application on top of the NSAL, i.e., the management application, can listen for the stream of samples.

4.3.4 Extended Graph and Device Models

In the following, we describe the structure of the (extended) topology graph which in-cooperates the processing pipelines and QoS features of the devices into the high-level topology. The

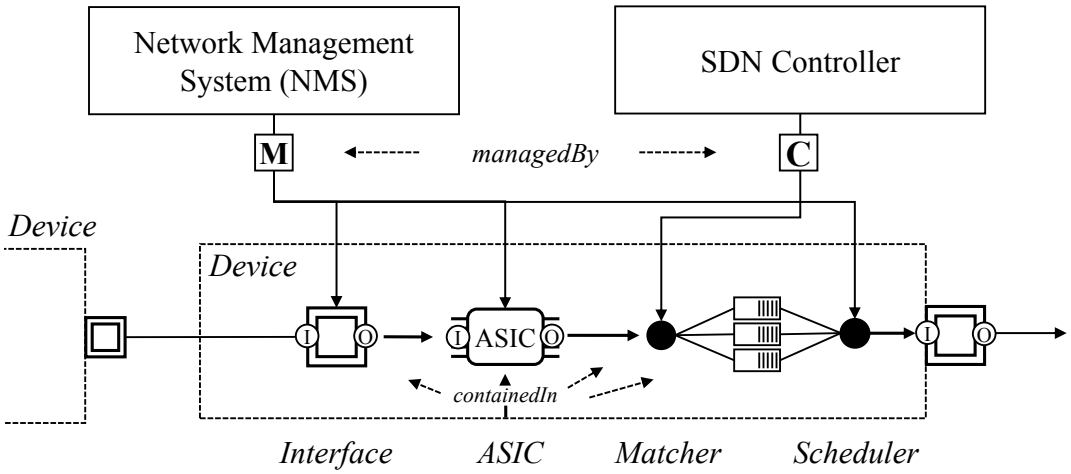


Figure 4.4: Example of a extended graph with processing pipeline of an OpenFlow switch with two connected ports, an Application-Specific Integrated Circuit (ASIC), a matcher, three queues, and a scheduler. *managedBy* relationships define the responsible configuration entities. *containedIn* relationships define dependencies, e.g., of a device or interface shutdown. Input (Ⓘ) and Output (Ⓞ) define the packet processing path.

extended graph tackles the design challenge **C2.1** of different control and management data-models and QoS discovery.

We use three devices from our testbed as reference, an NEC PF5240F, a Cisco Catalyst 4503-E and an HP-V1910 switch. We define the graph as a list of components with attributes and unidirectional relationships/links between the components. The components can be put into two categories; components that describe the processing pipeline, e.g., a queue or scheduler, and components that describe switch features, e.g., an OpenFlow interface component. In the following we introduce the components associated with the processing pipeline and discuss component relationships.

Table 4.2 summarizes the relationships between the switch components used in the extended network graph. There are the *input* and *output* relationships, which are used to specify the processing pipelines. *managedBy* allows to delegate the configuration interface of a component to a different component. For example, a hardware interface of a switch does not have its own configuration interface, but is configured through the CLI or NETCONF configuration interface. *containedIn* specifies the physical or logical dependency of a component and allows to deduce failure or maintenance impacts from the graph. For example, an interface restart results in unavailability of the whole processing pipeline inside the interface.

To describe the processing pipeline, we use an *input* and *output* relationship. Depending on the type of the component, a component can have zero or more inputs and zero or more outputs, e.g., a scheduling component has multiple inputs and one output. Figure 4.4 gives an example model of a simple OpenFlow switch with two connected ports. The interface shown

Table 4.2: Relationships Between Switch Components

Type	Description
managedBy	Refers to the component/device which accepts the configuration changes for the component.
containedIn	Specifies which logical or physical component houses the component.
input & output	Specifies the packet processing workflow for a component.

to the left is directly connected to the ASIC, the switching engine. The input path from the interface to the right to the switching engine does not contain any components. Therefore it has no input shaping mechanisms. The output path to the interface to the right offers three queues, which are connected to a Weighted Fair Queuing (WFQ) scheduling component and a matcher. The matcher is associated with a *managedBy* relationship with the SDN Controller (Ⓒ). All other configurable components are managed by an NMS (Ⓜ). All components of the switch use the *containedIn* relationship to associate them with the device.

Some of the switches in our testbed allow to adapt the processing pipeline to specific use cases. We model this by introducing an alternative path start and end component in the graph. This enables applications on top of NSAL to traverse the different alternatives like they would traverse links and nodes in a standard high-level topology graph. Furthermore, this simplifies modeling the processing pipeline, as it does not inflate the definition of generic scheduler and generic queue components. Figure 4.5 presents a simplified model of the processing pipeline of the NEC switch in our testbed using alternative path components. The processing path on the top provides 8 queues scheduled by a WFQ scheduler, while the path on the bottom schedules only 6 queues with a WFQ scheduler and two 2 queues, plus the output of the WFQ, with a PQ scheduler.

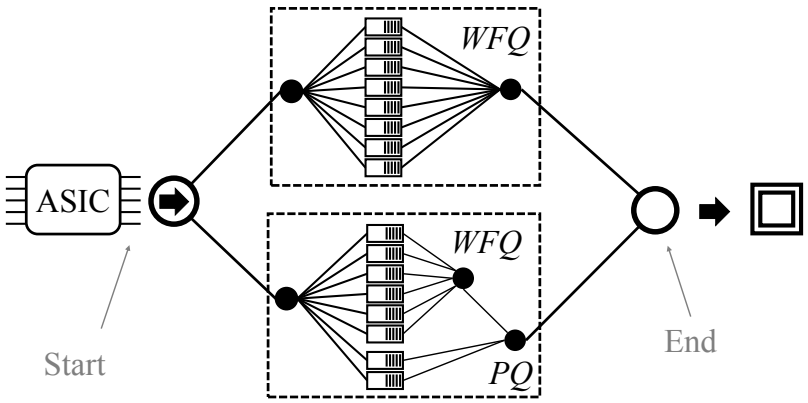


Figure 4.5: Example of an alternative output graph path from the switch's ASIC to an interface of a NEC device. The top path offers eight queues scheduled using WFQ. The bottom path two queues with PQ and six queues with WFQ.

The introduced extended graph enables control applications to discover path and QoS options simultaneously. For example, a control application which embeds flows with low-latency requirements into the network can discover the three queues and the scheduler in Figure 4.4. The application can do this without having to know about implementation details or vendor-specific configuration interfaces. After discovering the options, the application can reconfigure the scheduler and the matcher to isolate specific flows. By using the *managedBy* relationships, the NSAL is able to forward the configuration to the responsible entity for the matcher (Ⓒ) and for the scheduler (Ⓜ).

4.4 Task Composition and Overall Timing Estimation

In this section, we discuss how an application on top of the NSAL can perform management operations and query an estimation of the timing characteristics of the tasks to be executed. Querying the impact of a management operation beforehand tackles design challenge C2.4: non-inferable reconfiguration side-effects.

Management operations are expressed as atomic configuration *tasks*. One task, e.g., assigning a VLAN tag to an interface, either succeeds or fails and can be characterized by its timing, e.g., the time it takes for the desired configuration to put into effect, and side-effects, e.g., a necessary interface restart. In the following, we define a configuration task in detail. Afterward, we introduce the methodology for estimation of task timings and side-effects based on the measurements in Chapter 3. We give a generic solution to encompass a variety of use cases from enterprise networks to networks with strict timing characteristics. In the presented use cases we reduce the complexity as some aspects are not needed to describe the selected use cases.

A composite task T is defined as a set of atomic tasks t ($T := \{t_1, ..., t_n\}$). Each atomic task is defined by four types of delay. t^0 denotes the time the atomic task is executed, t^t the transport delay caused by the NSAL, i.e., the task forwarding to the agent, the processing time by the agent and the physical propagation delay between agent and the target device,

$$estimate(T) := \begin{array}{c} \\ t_1 \\ t_{..} \\ t_n \end{array} \begin{array}{c} \begin{array}{cccc} t^t & t^p & t^b & t^d \end{array} \\ \begin{pmatrix} t_1^t & t_1^p & t_1^b & t_1^d \\ .. & .. & .. & .. \\ t_n & t_n^p & t_n^b & t_n^d \end{pmatrix} \end{array} \tag{4.1}$$

$$exec(T) := \begin{matrix} \\ t_1 \\ t_{..} \\ t_n \end{matrix} \begin{matrix} t^t + t^p \\ \begin{pmatrix} t_1^t + t_1^p \\ t_{..}^t + t_{..}^p \\ t_n^t + t_n^p \end{pmatrix} \end{matrix} \tag{4.2}$$

A composite task can either be *estimated* or *executed* through the NSAL. An estimation tries to estimate t^t, t^d, t^b, and t^p based on the measurements. In our prototype implementation, the estimation returns the average values of the measured delays and in case of the transport delay, the average measured transport delay (round-trip) of previously executed tasks on a specific device. Note that the device database does not only contain measurements, but additional knowledge of domain experts is still required and implemented to capture corner cases, e.g., changing from scheduler A to B restarts the interface, changing from B to A not. Equations 4.1 and 4.2 summarize the output of the estimation and execution function, respectively. We focus on t^b in the remainder of this chapter, as interruptions are most relevant to our investigated use cases.

4.5 Use Cases and Prototype Evaluation

In the following, we introduce and evaluate two use cases implemented on our proposed architecture. Both use cases are from the domain of SDN-hybrid networking. In the first use case we evaluate the VLAN-based Panopticon [110] approach from the perspective of the management plane and show how the domain-specific knowledge provided by the NSAL can reduce or prevent short-term interruptions of virtual networks. In the second use case, we combine the first use case with QoS discovery and configuration. We use the NSAL to discover alternative processing pipelines with low latency scheduling to prioritize traffic of specific VLAN tunnels. Furthermore, in this use case, we deploy task prediction and the on-demand monitoring to prevent mid-term, i.e., about 6 seconds, service interruptions of virtual networks.

4.5.1 UC1: VLAN Tunneling / Panopticon

Figure 4.6 depicts the test-bed topology for the VLAN tunnel use case UC1. Two SDN domains are connected by a traditional network consisting of five switches. The five switches allow three paths between two SDN domains where the two traditional edge nodes are shared by all three paths. For sake of simplicity, the relevant traditional hardware interfaces are numbered from 1 to 15. t_x denotes the atomic task to change the VLAN tagging of interface x.

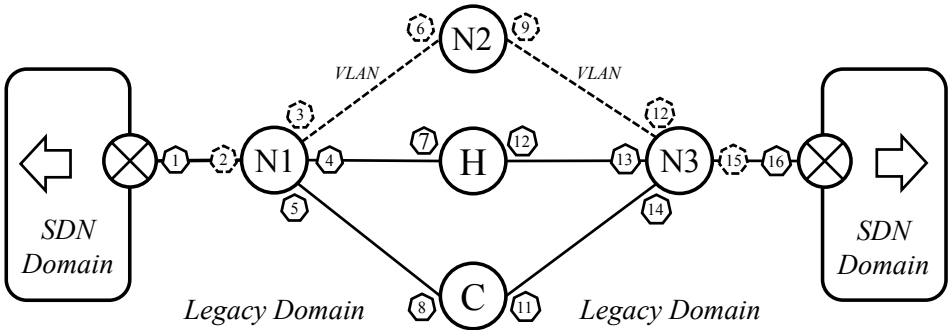


Figure 4.6: Two SDN domains in our test-bed connected by a VLAN tunnel. Dashed interfaces indicate the configured VLAN tunnel on the top path. **C** (Cisco Catalyst 4500), **N** (NEC PF5240) and **H** (HP V1910) denote the type of the switch.

Table 4.3: Task execution order and resulting interruptions

ID	Execution order	Blocking Duration t^b
1	$\{t_2, t_7, t_{13}\}, \{t_4, t_{12}, t_{15}\}$	251.6 ms
2	$\{t_4, t_7, t_{12}, t_{13}\}, \{t_2, t_{15}\}$	125.8 ms
3	$\{t_2\}, \{t_4\}, \{t_7\}, \{t_{12}\}, \{t_{13}\}, \{t_{15}\}$	1984 ms

Now, we assume device *N2* on the top path is scheduled for maintenance, e.g., a firmware upgrade. Without any further information about the devices and without advanced monitoring, a naive approach could be to execute the tasks $t_x, \forall x \in \{2, 4, 7, 12, 13, 15\}$ in parallel to create a new VLAN tunnel between the SDN domains to steer the traffic through device *H*. Note that $t_x, \forall x \in \{2, 4\}$, $t_x, \forall x \in \{7, 12\}$ and $t_x, \forall x \in \{13, 15\}$ cannot be executed in parallel as they require changes to the same device. A more advanced approach could first execute $t_x, \forall x \in \{4, 7, 12, 13\}$ to configure a new tunnel and afterward, when the device acknowledged the command, change the steering by executing $t_x, \forall x \in \{2, 15\}$.

Table 4.3 lists the overall blocking for different task execution orders. The overall blocking duration is derived from the task execution order and the per-device measurements presented in Chapter 3.1. We assume that each set is executed in the order given in the table and that the tasks in a set are executed in parallel. The orders with ID 1 and 2 are the worst and best case task execution orders for parallel task execution, respectively. Order 3 represents the worst case for sequential task execution, when the management application waits for confirmation before continuing to the next task. From the table we conclude that a planned task execution decreases the traffic interruptions on average by factor 16. Although the absolute interruption time of up to 2 seconds seems negligible for low frequencies of configuration changes, we argue that a future autonomous decision entity on top of the NSAL makes use of the interface with a higher frequency than a static VLAN tunnel set-up. For example, this can be for load-balancing reasons and, therefore, can result in frequent interruptions.

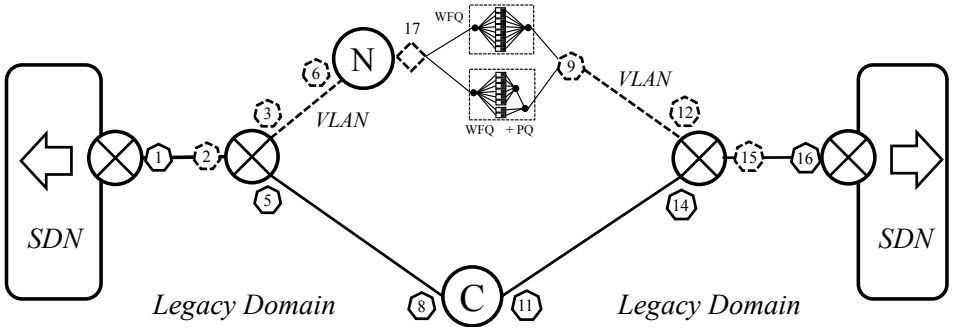


Figure 4.7: Set-up for the second use case consisting of two SDN domains, two named switches, NEC and CISCO. Switch NEC supports an alternative QoS processing pipeline on the output interface as shown in Figure 4.5

4.5.2 UC2: QoS Discovery and On-demand Monitoring

In the second use case UC2, we show how a management application uses the NSAL to first discover low-latency options and second, configure low-latency tunneling through a traditional network. Figure 4.7 depicts the test-bed for the second use case. The network consists of an SDN and traditional domain. Two (physical) paths through the traditional domain are connecting the two SDN domains. One path traverses the NEC switch on top and the second path traverses a simple traditional switch with VLAN configuration options on bottom. The NEC allows to switch to alternative packet processing pipelines as hinted in the upper part of the figure and illustrated in detail in Figure 4.5.

In order to provide low-latency virtual networks, the management application first has to discover the *extended* network graph through the NSAL. The discovery takes a path of nodes and edges from the high-level topology and returns an extended path consisting of the switch components on the path, including the alternative processing selectors and QoS schedulers. In the extended graph, it can calculate all shortest paths where priority queuing is available and based on the calculated paths, determine the alternative path selectors to configure. If the path selector is configured without further analysis, i.e., without calling *estimate* of the task to be executed, the virtual networks are interrupted for about 6 seconds as the measurements in Chapter 3.2 showed.

In the following, we show how the domain knowledge of the device database provided by the NSAL allows the management application to predict the interruption, monitor it on-demand, and reroute the traffic accordingly. We denote $t_x, e \in \{2, 3, ..15\}$ as the atomic task to configure the interfaces $\{2, 3, .., 15\}$ and the alternative path selector $\{17\}$. A management application aware of the prediction provided by the NSAL, first gathers a set of tasks required for changing to an alternative processing pipeline, denoted as T, with $T := \{t_{17}\}$ in this case. Second, it calculates the combined blocking delay by $t^b_{max} := max(\{t^b_x\}, \forall x \in T\})$ on the path. If the t^b_{max} is larger than a defined threshold t^b_{thress} of acceptable interruption, it decides to re-route the

```
1  import nmdsl as nm
2  import networkx as nx
3
4  topo = nm.topology()
5  sp = nx.all_shortest_paths(topo,source=1,target=16)
6
7  for s in sp:
8    et = nm.ext_topo(s)
9
10   pq = [e for e in et if e.type() == "PriorityQueuing"]
11
12   if not len(pq): continue
13   break
14
15 pq = pq[0] # only one pq scheduler here
16
17 if not pq.is_selected():
18
19   if pq.select().estimate()['t_b'] > 5:
20       nm.execute([e.set_vlan(12) for e in sp[1][1:-1]])
21
22   pq.select().execute()
23   pq.contained_in().wait_for('intf_status=="up"',
24                              freq = 4)
25   nm.execute([e.set_vlan(10) for e in sp[0][1:-1]])
```

Listing 4.1: Implementation of use case 2 in the proposed DSL

traffic before the change of the processing pipeline. For example, the interruptions observed in Section 3.2 can be expressed by $estimate(t_{15})$, which returns an average t_b of 6 seconds.

Next, we introduce the implementation of this use case in the Python-based DSL. We assume the reader to be familiar with the general Python3 syntax. Listing 4.1 gives the implementation in source code and the subsequent enumeration summarizes the steps with corresponding line numbers in square brackets. Please note that the listing is over-fitted to the scenario due to space constraints, i.e., additional loops and checks are required to make it work under different scenarios and environments.

Step 1 [4] Retrieve high-level topology

Step 2 [5] Calculate shortest paths

Step 3 [8] Request extended graph for shortest paths

Step 4 [15] Select path with priority scheduler

Step 5 [19] Predict t^b for alternative processing path

Step 6 [20] Re-route traffic to second shortest path

Step 7 [22] Execute conf. job for alternative processing

Step 8 [23] Monitor interface, wait until available again

Step 9 [25] Re-route traffic to first shortest path again

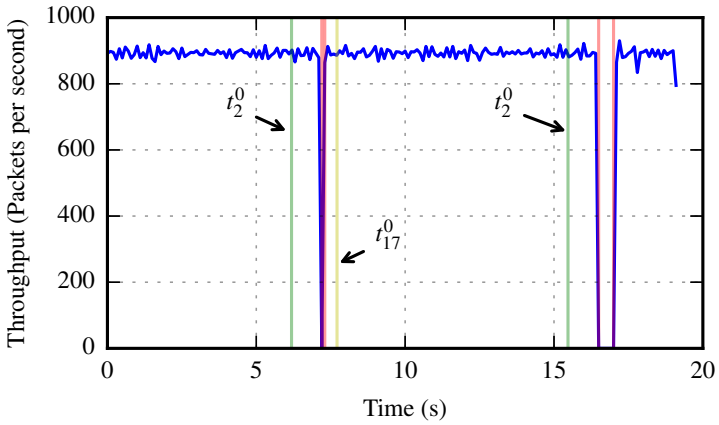


Figure 4.8: Use case 2 implemented in our testbed. The short traffic interruptions are caused by rerouting of the traffic by changing the VLAN configuration. t_2^0 and t_{17}^0 are the times when the management application created the configuration tasks and before the task is executed by the agents.

First, we retrieve the high-level topology as a graph of connected interfaces. This requires one call to the NSAL. Second, we calculate the path options between the two domains and afterward, in Step 3, we query the NSAL to retrieve the extended graph including the interface processing pipelines for each relevant path, i.e., the upper and lower one in this use case. Subsequently, in Step 4, we traverse the extended graph to search for priority schedulers. Next, we check if the scheduler is on a selected path segment by calling *is_selected* on the priority queuing switch component. As the path segment is not selected, we create an anonymous task object and retrieve t^b by calling *estimate*. Note that this does not execute the task. Afterward, as $t^b > 5s$, we use VLAN tunneling to create a linear VLAN broadcasting domain (excluding the SDN edge interfaces) to steer the traffic to the bottom path. This implicitly creates a list of configuration jobs, which are forwarded to the configuration agents. In Step 7, we execute the configuration job to select the priority scheduling. This creates one configuration job, which is executed on the NEC switch and results in an interface restart, thus, a blocking time. In Step 8, we create a monitoring job to check every 250 ms for the interface status and return when the interfaces is available again. In the final step, we change the VLAN configuration to steer the traffic again to the default upper path.

Figure 4.8 depicts a stream of packets traversing the set-up presented in Figure 4.7. The figure shows how the packets are affected by the described configuration changes. The traffic is interrupted two times, but only for a brief duration and the interruptions correspond to the measured t^b values in Chapter 3.1. This demonstrates how the interplay of discovery, management, and monitoring enables an autonomous management application on top of the proposed NSAL can correctly predict and reduce traffic interruptions.

4.6 Summary and Discussion

In the following, we summarize the chapter, relate the architecture to the aforementioned challenges and discuss open questions. In order to facilitate innovation in the management-plane of modern network equipment, a device- and vendor-neutral unified NSAL for discovery, monitoring and configuration of device resources is required. Such a unified abstraction is a first step towards combining network control with network management in a programmable manner. Thus it unifies the feature set of network control with the feature set of management actions. Additionally, the abstraction layer provides means to capture domain knowledge from human experts and human-centered manuals, such as device-specific limitations and non-inferable side-effects of management actions. For example by making port-based QoS schedulers available through the programmable abstraction layer, the number of possible QoS options compared to the OpenFlow and OFCONF data model can be increased.

The chapter proposes an architecture with an extended network topology that provides a hardware detail-level that includes even the processing pipelines of devices. Via a northbound interface, management applications can run networking tasks through a vendor- and device-neutral abstraction layer. By example of Panopticon, a strategy for OpenFlow networking in mixed-SDN/traditional networks, we show how a management application can implement the strategy through the NSAL. Furthermore, the architecture provides mechanisms to estimate traffic interruptions due to the triggered management actions, e.g., due to the changes of the QoS scheduler configurations. By example, we show how a smart task scheduling based on the timings of the management actions can mitigate service interruptions in a QoS use case.

In summary, challenge **C2.1**, *different control and management data-models*, is tackled by providing detailed device models which combine control and management aspects of a device. Challenge **C2.2**, *the level abstraction*, is tackled by choosing a level of abstraction which allows rapid development and semi-autonomous decisions. Challenge **C2.3**, *different reconfiguration times*, is addressed by providing an API for northbound applications to query beforehand the execution times of different reconfiguration tasks and thus gives the northbound application the possibility to plan accordingly. Challenge **C2.4**, *non-inferable reconfiguration side-effects*, is tackled by the same API as discussed for challenge **C2.3**. The API also offers the opportunity for northbound applications to query possible data-plane interruptions as side-effects of the reconfiguration. Challenge **C2.5**, *on-demand and long-term monitoring of heterogeneous devices*, is approached by integrating monitoring in the detailed device models. Monitoring can either be expressed as a task as short-term and high-frequency monitoring or as long-term monitoring through a permanent configuration setting. Challenge **C2.6** is discussed in detail in the subsequent Chapters 5 and 6.

Chapter 5

Theoretical Performance Limits of an NSAL for Hybrid Networks

Chapter 3 presented measurement results highlighting the differences in flexibility between Software-Defined Networking (SDN) and traditional devices in terms of reconfiguration speed. The measurements show an up to hundredfold difference of reconfiguration times between SDN and traditional devices. Hence, traditional devices are not designed for frequent reconfigurations and are slow to react to reconfiguration commands by the Network Services Abstraction Layer (NSAL) management module. SDN-capable devices on the other hand are fast to reconfigure through the simple match-action rules pushed by the NSAL control module. In this chapter we focus on the question how the differences in reconfiguration speed constrain the maximum achievable reconfiguration rate, denoted as λ_{max}, of the NSAL. Furthermore, we investigate a collection of real-world topologies and shed light on the achievable reconfiguration rate in various stages of the SDN migration process, depending on the size and structure of the real-world topologies.

We first introduce a methodology which allows to evaluate the impact of these differences on the maximum reconfiguration rate of the whole network for different SDN deployment stages. Second, we introduce a metric, the potential P, which describes how much a topology benefits from SDN deployment. Third, we investigate a large number of real world topologies and the gain in terms of flexibility that can be achieved by deploying SDN-capable devices incrementally. The results show that even a small number of slow devices can severely constrain the maximum reconfiguration rate. Furthermore, even with the majority of the network nodes being replaced by SDN-capable devices, the maximum reconfiguration rate increases in the best case only up to five times compared to the all-traditional network.

The contribution of this chapter is threefold. First, we introduce an analytical method based on queuing theory to quantify and compare the maximum reconfiguration rate λ_{max} based on

the ratio of SDN and traditional devices in the network. Second, we propose an intuitive metric to compare different network topologies in terms of their benefit through SDN deployment in terms of flexibility. Third, we investigate a large number of real world topologies and the gain in terms of flexibility that can be achieved by increasing SDN deployment.

The chapter is structured as follows. Section 5.1 defines the problem and challenges of determining the flexibility in terms of maximum reconfiguration rate of a hybrid network in detail. Section 5.2 introduces the relevant background. Section 5.3 defines the general system model and Section 5.4 introduces the potential P of a topology. Section 5.5 surveys a large number of real world topologies based on the introduced methods and metric. Section 5.6 summarizes the chapter and discuss the findings.

The content of this chapter is based on the theoretical framework introduced in [15] and the measurements results presented in [12]. [15] introduces the scenario, defines the methodology, proposes the potential P and conducts the survey using a set of real-world topologies. [12] introduces measurements regarding reconfiguration timings of different traditional and SDN-enabled switching platforms.

[15] C. Sieber, R. Durner, and W. Kellerer. "How fast can you reconfigure your partially deployed SDN network?" In: *IFIP Networking Conference*. 9 pages. Stockholm, Sweden, 2017, p. 9. DOI: 10.23919/IFIPNetworking.2017.8264845.

[12] C. Sieber, A. Blenk, A. Basta, D. Hock, and W. Kellerer. "Towards a Programmable Management Plane for SDN and Legacy Networks." In: *IEEE Conference on Network Softwarization (NetSoft)*. 9 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502428.

5.1 Challenges and Problem Definition

Figure 5.1 depicts the NSAL as introduced in the previous chapter. The figure shows a network consisting of traditional (Ⓛ) and SDN-enabled devices (Ⓢ). The network is controlled by the unified control and management layer which provides applications northbound, e.g., an orchestrator, control over the network. For example, the orchestrator could be OpenStack [181], a cloud computing framework. The orchestrator uses the abstractions provided by the NSAL to query the network topology and to trigger reconfigurations. We assume that these reconfigurations are requested with a certain rate λ by the orchestrator and implemented by the NSAL in the network on one or multiple devices. A problem arises if the global rate of reconfiguration is over a certain threshold the network can handle, denoted as λ_{max}. Hence, if $\lambda > \lambda_{max}$ the system becomes overloaded and reconfiguration times increase.

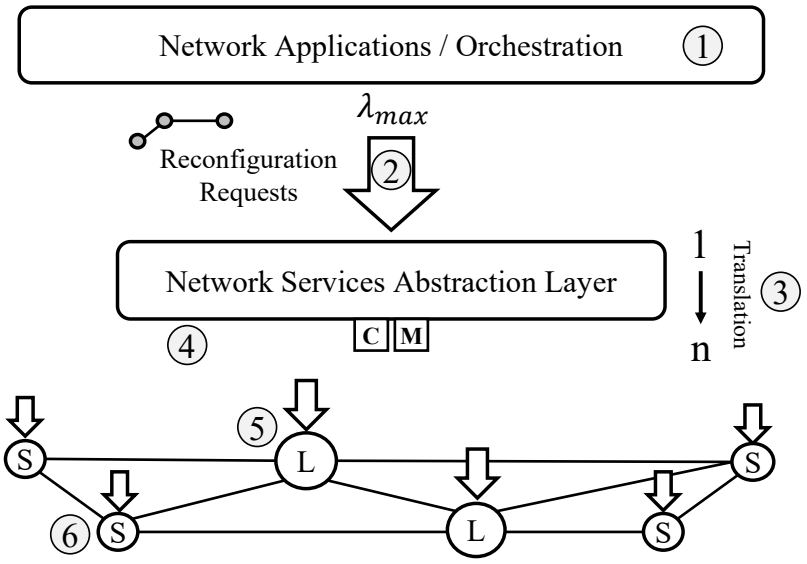


Figure 5.1: Overview of the scenario and investigated challenges. Reconfigurations are done by network control applications or orchestration systems (①) on top of the NSAL with a rate of λ (②). Each reconfiguration request translates to multiple reconfiguration tasks (③). Reconfiguration tasks are non-uniformly distributed to the devices in the network (④). Some devices react slowly, e.g., traditional devices, but receive a big fraction of the tasks (⑤).

5.1.1 C3.1: Determining Maximum Global Reconfiguration Rate

The NSAL is a platform for northbound control applications for use cases ranging from load balancing, access control, Quality of Service (QoS) configuration to traffic accounting, each with a certain demand on reconfiguration latency and rate. A use case becomes infeasible if the demand in reconfiguration rate and latency exceeds the available rate and the control- or management-planes of devices become overloaded with reconfigurations. Overloaded devices result in reconfiguration queue buildup at the NSAL and increased control delay. Challenge C3.1 is how to determine λ_{max} for a given topology and SDN deployment stage. The importance of λ_{max} comes from the fact that λ_{max} determines the feasibility of network management use cases. Use cases which may be possible on fully deployed SDN networks, such as per-flow routing, become infeasible with partially deployed networks where slow traditional devices can not keep up with a reconfiguration per flow arrival.

5.1.2 C3.2: Quantifying Benefit of SDN Upgrade for a Network

From the survey in Chapter 2.6 we conclude that the (partial) deployment of SDN-capable devices is mostly driven by use cases such as monitoring, load balancing or access control, besides the promised ease of management. A deployment strategy therefore faces the question at one point in time: How much is the benefit from upgrading more traditional devices to

SDN-capable devices? Challenge C3.2 is therefore how to quantify the benefit of continued SDN upgrade for a network in terms of maximum reconfiguration rate.

5.1.3 C3.3: Quantifying Average Benefit for Real-World Topologies

The structure of evolved real-world provider and enterprise networks differs from synthetic and well-structured topologies found in data-centers and simulations. It is therefore important to evaluate the benefit of incremental SDN upgrade for realistic topologies. Challenge C3.3 is the investigation of the distribution of the benefit in realistic upgrade scenarios. The distribution allows to deduce important insights into SDN deployment gains with respect to reconfiguration rates. That way, popular SDN use cases can be evaluated for their feasibility in realistic incremental deployment scenarios.

In the subsequent sections we first introduce the background, methodology, metrics and notations for describing the flexibility of partial SDN deployments. Afterwards we introduce the system model. Based on the system model, we show how the maximum global reconfiguration rate of a network topology can be calculated. Besides the graph properties of the topology, the placement of the traditional and SDN devices in the partial deployments is important for the global reconfiguration rate. We discuss a best case, worst case and random placement of traditional and SDN devices in the network. Afterwards, we discuss the maximum feasible reconfiguration rate of all possible placements in general and of a specific topology in particular as an example. Then, we introduce the potential P, an intuitive metric for comparing different topologies in terms of their expected gain in flexibility when deploying SDN. The last part of the chapter investigates a large collection of real-world topologies for their suitability for incremental migration to SDN.

5.2 Background

To the best of our knowledge, the literature does not tackle the problem of different reconfiguration speeds in hybrid networks so far. However, there are publications related to orthogonal topics which we describe in the following. One orthogonal research angle is to reduce the number of necessary reconfigurations. This can be done for SDN using wild-card rules. Authors of [52, 164] aim to reduce the control plane load by limiting the network state view of the centralized controller. As a side effect, this also reduces the necessary reconfiguration rate and mitigates the issues we try to solve here. Differences in the time to update switches can also cause congestion and inconsistencies in a network in general. Authors of [97] provide an algorithm that schedules network updates efficiently. The focus of the work is on optimizing

rule ordering for network updates which are non-atomic in order to improve reconfiguration speed. However, the global maximum reconfiguration rate is not evaluated.

Authors of [83] evaluate gradual deployments of SDN devices in terms of throughput. They review real world topologies, where some percentage of the devices are replaced. They observe that for traffic engineering a relatively small deployment of about 20 % can reduce congestion and maximum link usage in real Internet Service Provider (ISP) and enterprise topologies effectively. Authors of [93, 116] provide a traffic model of OpenFlow. However, the authors focus on modeling reactive SDNs, while we aim to model the maximum achievable reconfiguration rate independent of the reactive or proactive use case, and especially for mixed SDN/traditional networks.

5.3 System Model

Figure 5.2 shows the arrival process at the NSAL with the global reconfiguration rate λ_g. Table 5.1 summarizes the notation. The reconfigurations arrive through the NSAL in a probabilistic manner following a Poisson process. The inter-arrival times are negative exponential distributed. This mimics the combined request stream from different network applications such as orchestrators, load balancers or firewalls on top of the NSAL. The NSAL translates the reconfigurations to configuration commands or OpenFlow messages for each device and forwards them to the devices. Different assumptions can be made in the way how the commands are completed on the device. We assume a First-In First-Out (FIFO) queue of all the commands for each node. For traditional devices this might not be an actual queue on the device, but rather a blocking Transmission Control Protocol (TCP) management connection which requires the NSAL to store the commands until the device is free again to accept the next command. Regarding SDN devices, related work shows that the configuration times are limited by the interaction between Central Processing Unit (CPU) and Application-Specific Integrated Circuit (ASIC) [80]. As a result, the commands are queued by the switch software running on the CPU, e.g. the OpenFlow agent.

The measurements in Chapter 3 show that the command execution times on the devices depend on multiple influence factors. For our model we assume a common deterministic reconfiguration time for all traditional devices and likewise for all SDN devices (h^L and h^S). One device is therefore modeled as an *M/D/1* system following Kendall's notation. Multiple reconfigurations can be performed in the network at the same time, if they require reconfigurations of disjoint sets of devices. But if multiple reconfigurations are required to a single device, the configuration commands of the device are enqueued.

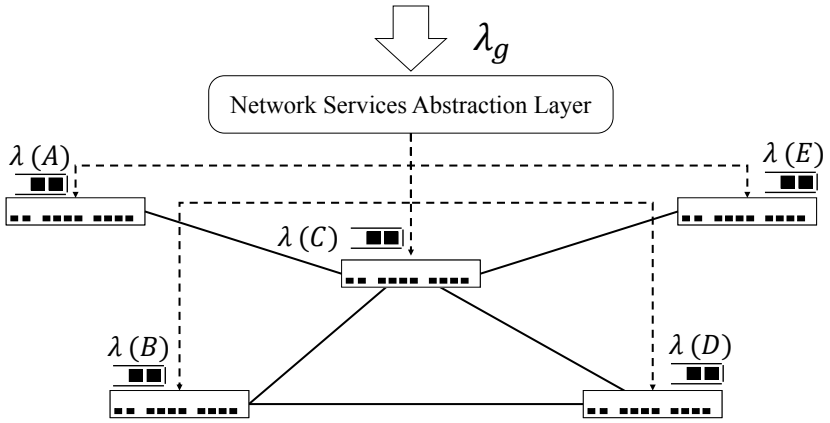


Figure 5.2: Abstract reconfiguration model with one configuration queue for each device showing local and global configuration arrival rates

Table 5.1: Key variables and notations used in the chapter.

Notation	Description
λ_g	Global arrival rate
$\lambda(v)$	Arrival rate at device v
V	All nodes/devices in the topology
$h(v)$	Deterministic configuration time of device v in seconds
$\|V\|$	Number of devices in the topology
$\sigma(s,t\|v)$	Boolean indicating path from device s to t includes v
$c_B(v)$	Centrality of device v
h^L, h^S	Reconfiguration time of traditional and SDN devices

The maximum achievable reconfiguration rate can be computed using queuing theory. Each device v has its deterministic reconfiguration time $h(v)$. Not every global reconfiguration includes every device, so only a fraction of λ_g has to be processed by one device which leads to a local arrival rate $\lambda(v)$. The fraction of the global reconfigurations processed by a single device can be chosen in different ways. For example, the fraction for each device can be deduced from historic data. Or, it can be estimated based on the intended use case of the partial SDN-deployed network at hand. We choose the fractions based on the transition use case of setting up Virtual Local Area Network (VLAN) tunnels in the network. If the tunnel endpoints are chosen randomly from all devices in the network and all tunnels are setup using the shortest path, the probability of device v being part of one tunnel is exactly the betweenness centrality $c_B(v)$. Besides the use case of tunnel configuration, the betweenness centrality follows the intuition that a more central device is part of more reconfigurations in the network. $c_B(v)$ of a specific device v is defined in [67] and specifies the fraction of all shortest paths which include the specific device:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{5.1}$$

Here $\sigma(s,t|v)$ equals one if the path from device s to device t includes v otherwise zero. The local configuration rate can then be computed as:

$$\lambda(v) = c_B(v) \cdot \lambda_g \tag{5.2}$$

For a given topology and configuration times, it must hold true:

$$\forall v : \lambda(v) \cdot h(v) < 1 \tag{5.3}$$

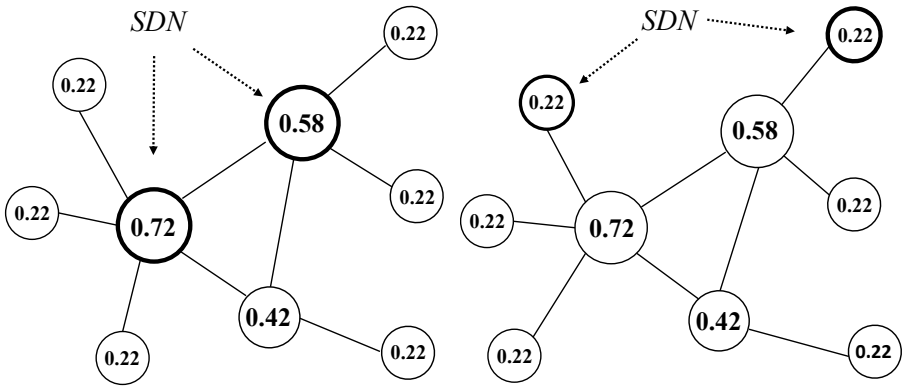
or else the configuration queue grows unrestricted. This leads to a maximum global configuration rate of a given network:

$$\lambda_{g,max} = \max_{v \in V} \left(\frac{1}{c_B(v) \cdot h(v)} \right) \tag{5.4}$$

5.3.1 Network Realizations

The measurements show that SDN devices support a higher reconfiguration rate compared to traditional devices. However, to determine λ_{max} it is not enough to just consider the number of deployed SDN devices, but also their placement in the topology, i.e., which of the traditional devices in the current network are replaced by faster SDN devices. For a given deployment ratio and topology, the SDN devices can be placed in different ways. We call the resulting networks different realizations of the same topology. For example, for a device count of 9 in a topology and a deployment ratio of 22 %, there are $\binom{9}{2} = 36$ possible network realizations, including the best and worst case realizations. Two possible realizations, a best case and a worst case realization, are shown in Figure 5.3. In the best case, the fast SDN devices are placed most central (Fig. 5.3a). In the worst case, they are placed at the edge of the network (Fig. 5.3b) . Best case and worst case result in largely different feasible configuration rates. For a deployment ratio of 22 % and $h^L = 650\,ms$ and $h^S = 1\,ms$, the best case realization has a maximum rate of $1389\,s^{-1}$, while in the worst case the network can only process 2.14 reconfigurations per second.

The maximum possible reconfiguration rates of all possible network realizations can be summarized as follows: We assume that the reconfiguration bottleneck is always a traditional node. This is true for most topologies if the SDN reconfiguration rate is much faster than the traditional reconfiguration rate. Therefore, regarding the maximum global reconfiguration rate, the most central non-SDN device constraints the reconfiguration rate.



(a) Best Case (b) Worst Case

Figure 5.3: Best case (left) and one worst case realization (right) of a topology with 9 nodes and a deployment ratio of $\frac{2}{9}$. The annotations in the nodes denote in-betweenness centralities.

The number of network realizations with a specific device as bottleneck can be deduced from counting the placement combinations. The maximum reconfiguration rate of one device, if the device is traditional, is given with:

$$\lambda^L_{max,v} = \frac{1}{h^L \cdot c_B(v)} \tag{5.5}$$

We sort all rates and index the resulting set Λ with index x:

$$\Lambda_{max} = \{\lambda^L_{max,v}(x) \forall v \mid \lambda^L_{max,v}(x) \leq \lambda^L_{max,w}(x+1)\} \tag{5.6}$$

With this we can compute the fraction of all possible topology realizations which are constrained by the maximum reconfiguration rate at index x of Λ_{max} with l traditional devices out of $|V|$ total devices in the topology:

$$\frac{\binom{|V|-x}{l-x}}{\binom{|V|}{l}} \tag{5.7}$$

By applying the formula for all $\lambda_{max,v}$ in the set Λ_{max}, we get the maximum reconfiguration rate distribution of all realizations. Next, we discuss how the deployment ratio relates to λ_{max} and apply the described placement counting to an example topology.

5.3.2 Feasibility of Reconfiguration Rates

We define a reconfiguration rate λ_g feasible for a given SDN deployment ratio and network realization if the rate λ_g is less than λ_{max}. Therefore, when only considering the SDN deployment ratio, there exists a range of feasible configuration rates. Depending on the deployment ratio, we can identify three feasibility regions for a given network topology and given reconfiguration times (h^L and h^S). Figure 5.4 qualitatively illustrates the three feasibility regions I, II and III. The feasibility regions are defined as follows.

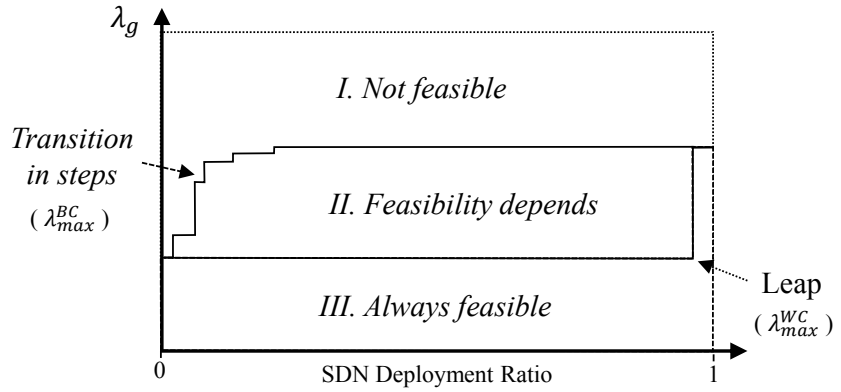


Figure 5.4: Qualitative feasibility regions depending on the SDN deployment ratio and maximum possible reconfiguration rate over all possible network realizations. For the best case λ^{BC}_{max}, the maximum reconfiguration rate increases in steps with increasing deployment ratio. For the worst case λ^{BC}_{max}, the rate stays low for higher deployment ratios until all devices are replaced by SDN devices.

I All reconfiguration rates in the region above the best case realization are infeasible, independent of the SDN and traditional device placement.

II The reconfiguration rate is feasible if the different device types are placed such that $\lambda_g < \lambda_{max}$ holds true.

III Even for the worst case realization the requested reconfiguration rate is always feasible.

For a network without any SDN devices, there is only one network realization and therefore the worst case rate λ^{WC}_{max} equals the best case rate λ^{BC}_{max} and there is no feasibility region II. With growing SDN deployment more and more central switches can be SDN-enabled and the best case rate grows quickly until all devices in the network's core are replaced by SDN switches and the growths saturates. The border between region II and III, where the worst case realizations are located, has a contrary behavior and first stays constant until it leaps to λ^{BC}_{max} when 100 % deployment is reached. At 100 %, there is only one possible realization again. Next, we take a closer look at feasibility region II for a real-world topology. The goal is to better understand the expected benefit of SDN deployment, if the deployment strategy is not optimal, e.g., a semi-random upgrade strategy. We evaluate the wide area network shown in Figure 5.5, the AT&T MPLS topology, taken from the Internet Topology Zoo [186].

For the evaluation of the expected gain we use the configuration delays of the NEC device, as the NEC is the most recent carrier-grade hardware among the measured devices with traditional and SDN mode. Hence we set $h^L = 650\,ms$ and $h^S = 1\,ms$. The topology has a device count of 25 and there are 2^{25} possible network realizations in total.

Figure 5.6 visualizes the maximum reconfiguration rate for all possible network realizations of the AT&T MPLS topology. The x-axis denotes the SDN deployment ratio. The y-axis

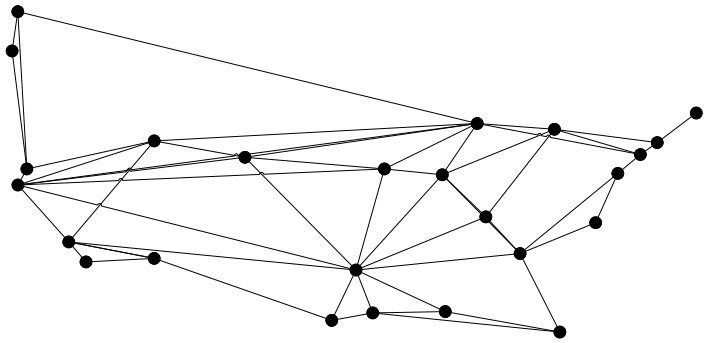


Figure 5.5: AT&T MPLS topology (Internet Topology Zoo [186])

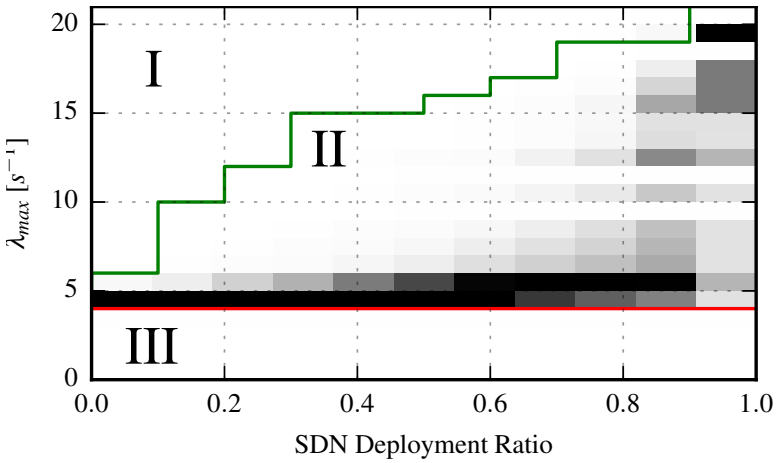


Figure 5.6: Analytical solution space for the AT&T topology for deployment ratios between 0 and 1 in steps of 0.1. λ_{max} is quantized to steps of 1. Color density indicates where the majority of network realizations are located. Green and red show the best case and the worst case realizations, respectively.

denotes the maximum global reconfiguration rate. The red line on the bottom indicates the worst case network realization. The green line indicates the maximum reconfiguration rate for the best case. We can observe that for this topology the border between feasibility area II and III is constant at $\lambda_{max} = 4.16\,s^{-1}$. The I/II border grows steadily for an increasing SDN deployment ratio.

The figure also shows the number of possible network realizations in area II using density indications. For the density indications we quantize λ_{max} and the SDN deployment ratio to steps of 1 and 0.1, respectively. It can be observed that until a ratio of ≈ 0.8 is reached, most deployments are close to the worst case with respect to λ_{max}. This is because a single slow device at a central position can constrain the global reconfiguration rate and the probability for this is high in a random SDN deployment.

From this it follows that the transition strategy has to carefully select which devices to upgrade, as a random replacement of traditional devices with SDN devices will most likely not benefit the flexibility of the network. This also holds true for faster traditional devices, i.e.,

for smaller values of h^L. Smaller values of h^L result in an increase in reconfiguration rate of the I-II and II-III borders. However, there is still a high density of slow reconfiguration rates close to the II-III border for $h^L >> h^S$.

5.4 Potential P

In order to evaluate and compare topologies in terms of their expected increase in global reconfiguration rate when deploying SDN, we propose the *potential* P of a topology. The feasibility regions show that centrality is a key influence factor of the potential flexibility gain. Therefore, we define P as independent of the configuration times of the devices and focus on the centrality c_B of each node. P is defined as the mean relative positive squared deviation from the median of the node centralities. C_{med} is defined as the median of all node centralities, $C_{med} = median\{c_B(v), \forall v \in V\}$. C_+ is the set of all node centralities greater than the median, $C_+ = \{h_B(v), \forall v \in V | h_B(v) > C_{med}\}$.

$$P = \frac{\sum_c^{\forall c \in C_+} (\frac{c - C_{med}}{C_{med}})^2}{|C_+|} \tag{5.8}$$

With the deviation from the median, the potential P puts an emphasis on the beginning of the migration phase with 0 % to 50 % of the devices being replaced by SDN devices. In general, the metric favors topologies with a high number of nodes with a high centrality and decreases for topologies without central nodes.

5.5 Topology Investigation

In the following, we evaluate the achievable gain in terms of global reconfiguration rate when deploying SDN in real-world networks. We do so by applying the methodology to the Internet topology zoo. The Internet topology zoo provides a database of about 250 publicly available real-world topologies ranging from small testbed installations to large global-scale backbone networks. We filter the topology zoo by discarding topologies which are not connected or not unique in terms of node centralities. After filtering, the topology set for the evaluation contains 81 topologies with a node count between 4 and 74 and an average of 22 nodes per topology.

In the best case, to increase the maximum reconfiguration rate of the network, a network operator would choose to upgrade devices to SDN which are central or receive the most reconfiguration requests and are slow to reconfigure at present. Therefore, we define the best case (BC) gain as the ratio between the achievable reconfiguration rate of a topology with

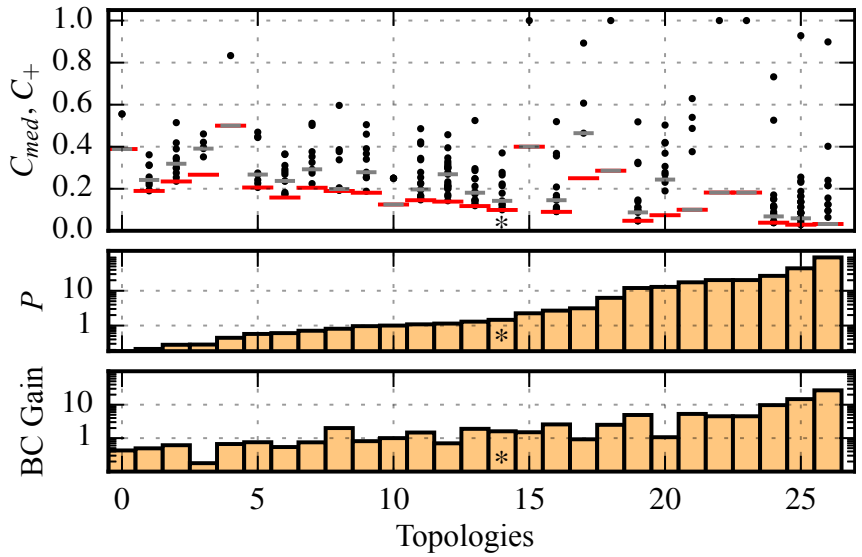


Figure 5.7: Centrality distribution, potential and best-case gain for a deployment ratio of 25 % of a random subset of the Internet topology zoo. C_{med} is shown as red horizontal line. C_+ as black dots. The bigger the distance between maximum and median is, the bigger is the potential for increasing reconfiguration rate and BC gain by deploying SDN.

and without SDN devices at the nodes in the topology graph with the highest centralities. For example, a BC gain of 1 refers to a doubling of the maximum achievable reconfiguration rate.

Next, we first discuss a random subset of the topologies in greater detail. Afterwards we summarize the findings over all topologies. At the end, we discuss the relationship of the gain and the potential P based on the topology set. Reconfiguration times are set to $h^L = 650ms$ and $h^S = 1ms$.

Figure 5.7 illustrates the BC gain, the potential P and the centralities for 26 of the 81 topologies. The BC gain is given for a deployment ratio of 25 %. P and BC gain are shown on a logarithmic scale and the topologies are sorted by P. The upper part of the figure shows the median of the centralities C_{med} with a wide red bar. The shorter gray bar shows the 75 % quartile, i.e., the most utilized traditional node in a 25 % SDN deployment. The dots indicate the centralities greater than the median (C_+) of each topology. The AT&T Multiprotocol Label Switching (MPLS) topology from Figure 5.6 is located at position 14 and marked with a star.

Two conclusions can be drawn from the figure. First, the potential P gives a good estimate of how well topologies can profit from SDN deployment compared to each other. However there are exceptions like topology 3, 17 and 20 which can profit far less from the SDN deployment than P suggests. This is due to that the difference between topology 3's most central node (upper most dot) and the 75 % quartile (gray short bar) is smaller in comparison to topology 2 and 4. Hence, topology 3 can not profit as much from an 25 % SDN deployment as topology 2 and 4. Second, the achievable best case gain for 25 % deployment varies greatly

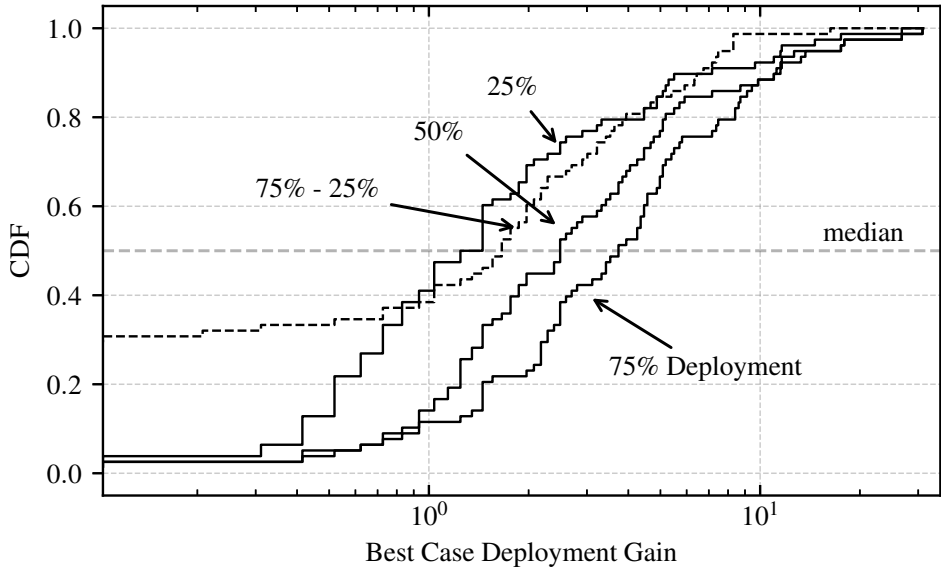


Figure 5.8: CDF of the best case deployment gain over 81 real world topologies for deployment ratios of 25 %, 50 %, 75 % and the difference in gain between 75 % and 25 %. A deployment gain of 1 translates to a doubling of the maximum global reconfiguration rate of the network.

between different topologies. For topology 3 we observe a gain of only 0.18, whereas topology 26 exhibits a gain of about 27. The AT&T MPLS topology is with a gain of 1.6 close to the median of 1.5 of all shown topologies. On average, there is a gain of 3.4.

Next, we evaluate the achievable gain for all topologies in the topology set for 25 %, 50 % and 75 % SDN deployment. Figure 5.8 shows the Cumulative Distribution Function (CDF) of the best case deployment gain on a logarithmic scale. "*75 % - 25 %*" denotes the difference between the 25 % and 75 % gain for each topology. The points where 50 % of the topologies have less than a specific gain are highlighted by the grey semi-transparent vertical line. The line intersects the CDFs at a gain of 1.37, 1.67, 2.57 and 3.76 from the left to the right.

The figures shows that for half of the real-world topologies, upgrading half of the nodes to SDN, results in the best case in an up to three to four times higher achievable reconfiguration rate. If only one forth of the devices are upgraded, the gain drops to two to three times in the best case. If three forth of the devices are upgraded, the achievable reconfiguration rate increases to four to five times. The CDF for the difference between 75 % and 25 % deployment answers the question of how much gain can be expected when continuing SDN deployment from 25 % to 75 % of the devices. The results show that for half of the topologies, an continuation in the upgrade process increases the reconfiguration rate up to two to three times, similar to the upgrade from all-traditional to 25 %.

Next, we discuss the correlation between the best case gain and the potential P based on the investigated topology set. Table 5.2 summarizes the Pearson linear-correlation between the

	h^L/h^S				
	5	**10**	**100**	**650**	**1000**
25%	0.66	0.85	0.94	0.94	0.94
50%	0.48	0.71	0.80	0.80	0.80
75%	0.35	0.56	0.74	0.74	0.74

Table 5.2: Pearson correlation between potential P and BC gain for 25 %, 50 % and 75 % SDN deployment and four different ratios of h^L/h^S with $h^L = 1ms$.

best case gain and the potential P for three different deployment ratios and five different ratios between the configuration times of the traditional and SDN devices (h^L/h^S). In general, we observe that the linear correlation is stronger for higher values of the ratio h^L/h^S. This comes from the fact that P expects the slowest SDN device to be faster than the fastest traditional device, which is less likely for lower ratios of h^L and h^S. Furthermore, the correlation decreases for higher deployment ratios. This is not surprising as P is defined to favor lower deployment ratios. The Spearman rank-correlation (not shown in the table) is ≥ 0.90 for all combinations of 25 % and 50 % deployment and all configuration time ratios shown in the table and ranges between 0.35 and 0.75 for 75 % SDN deployment. From this it follows that the potential P is suitable for comparing the gain of topologies in magnitude and rank.

5.6 Summary and Discussion

In this section we summarize the chapter and discuss the presented methodology, metric and the results of the topology investigation. Based on the results of the measurements in Chapter 3 we know that the evaluated traditional devices have highly varying reconfiguration times from 3 ms to 20 ms and up to 650 ms. For the system model we assume the worst case and define a traditional reconfiguration time of 650 ms and a SDN reconfiguration time of 1 ms. This is a best case choice from the perspective of the migration process. That way an upgraded device can support a 650 times higher reconfiguration rate and thus a potential reconfiguration bottleneck in the network is eliminated.

The following three challenges are investigated by this chapter. First, determining the maximum global reconfiguration rate of a topology (C3.1). Second, quantifying the benefit of SDN upgrade for a network (C3.2). Third, quantifying the distribution of the gain with real-world topologies (C3.3). To this end, we introduce a system model and argue that configuration tasks are formulated centrally, e.g., by a network management application on top of the NSAL, and afterwards placed in device specific queues. Based on previous work on SDN migration, we introduce a scenario where each device receives a fraction of the global reconfigurations generated by the abstraction layer. This fraction defines the reconfiguration work-load of

a device and is based on the device's in-betweenness centrality. However, the presented methodology is not limited to the in-betweenness centrality as workload metric for a device. Other options could be to determine the workload of a device based on historic data. Afterwards, we introduce the concepts of feasibility and maximum achievable global reconfiguration rate. For a specific topology in a specific migration stage, a global reconfiguration rate is feasible if no single device is overloaded by its fraction of the reconfigurations.

Subsequently, we discuss how the distribution of SDN and traditional devices in the network influences the maximum achievable rate. Here we see that there is a best case strategy which replaces first the devices with the highest workload and a worst case strategy to replace first all edge devices with low workload. Additionally, we present how the maximum reconfiguration rate of all possible upgrade strategies can be calculated. Based on an example we deduce that a random upgrade strategy results much more likely in a maximum reconfiguration rate close to the worst case than to the best case. This stems from the fact that a single slow device can severely impact the global reconfiguration rate and that the number of devices with low centrality is much higher compared to the number of devices with high centrality. At the end of the section we introduce the metric P which captures how well a given topology responds to the best case migration strategy. As we see in the evaluation, the potential P is suitable for capturing the best case gain well up to a 50 % deployment ratio.

We then apply the methodology to a large number of real-world topologies from the Internet topology zoo. We evaluate the expected best case gain for deployment ratios of 25 %, 50 % and 75 %. The results show that for 50 % of the topologies an SDN deployment ratio of 25 % increases the global reconfiguration rate up to two to three times. Furthermore, we see a similar increase when continuing the migration from 25 % to 75 %. Hence, when three out of four devices are upgraded, we observe the global reconfiguration rate being up to 5 times compared to the all-traditional case. In general it is surprising that the 650-fold increase in terms of possible reconfiguration rate of the SDN devices results in a relatively low increase in the global reconfiguration rate even in the best case.

An operator can apply the following three step approach to determine the flexibility of his network. In a first step, the operator measures the reconfiguration times of the devices ($h(v)$) in the network depending on the use cases at hand, e.g. for VLAN tagging. As a second step, the operator determines for each node in this network the fraction of reconfiguration it receives, either by monitoring the currently deployed network, or by approximating the future transition scenario through other means, such as a simulation. Subsequently, the operator can apply the presented methodology to determine a suitable upgrade strategy with respect to the desired maximum global reconfiguration rate.

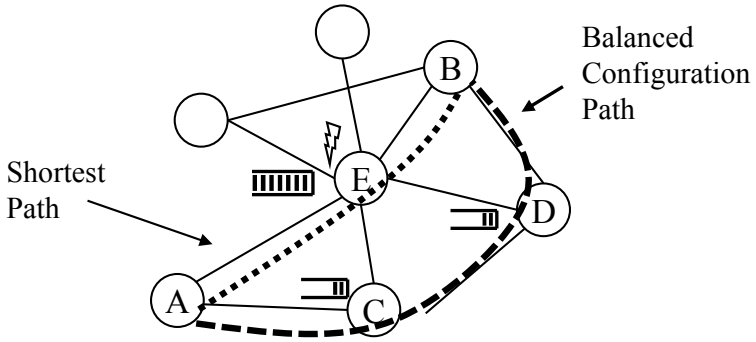


Figure 5.9: The design of a reconfiguration load-balancer could be a potential further research direction. A device at a central location can be overloaded by reconfigurations (E). However, the path from A to B going through C and D might be underutilization both in terms of data traffic and reconfiguration load on the devices. Reconfigurations could be balanced by restricting the path choices for the data routing.

One future research direction is the design of a reconfiguration load balancer. By distributing the reconfiguration load between the devices, the maximum global reconfiguration rate can be increased. However, this requires a careful design of a combined routing and reconfiguration balancing algorithm. In the example in Figure 5.9, the device E is overloaded in terms of reconfigurations. If link and control utilization allows it, the reconfigurations could be assigned to devices C and D instead. That way, the reconfiguration load on device E could be reduced and large queues of reconfiguration tasks prevented. This would not only increase the maximum possible rate, but also the reconfiguration delay and thus enable use cases for the NSAL which require frequent reconfigurations.

Chapter 6

Performance Modeling of a Software-Based NSAL at Runtime

The Network Services Abstraction Layer (NSAL) processes all reconfigurations and control messages to and from the network. This makes it a critical part of the infrastructure with high requirements on availability and performance. Therefore, understanding the performance of the NSAL at run-time is of importance. The chapter at hand investigates the application of machine-learning for the online performance modeling of an NSAL at run-time. Figure 6.1 illustrates the scenario and methodology. For reliability and horizontal scalability, the logically centralized NSAL (①) is physically distributed. The NSAL consists of one or more instances (③) which share (⑥) the reconfiguration load (λ) generated by the northbound control applications (②). The NSAL instances must not be over-utilized to prevent reconfiguration delay and should not be under-utilized to not waste resources. The central questions of this chapter are: which model describes the performance of an NSAL under resource constraints best? and how can this model be trained at runtime? And if there is a change in available resources, how can this be detected and the model consequently be adapted to the new environment? An accurate estimation of the maximum reconfiguration rate of a specific instance, denoted as the reconfiguration *budget* (⑤/⑧), allows a load-balancer (⑥) to optimally distribute the load between the instances. The challenges how to distribute the workload and how many instances are required are out of scope of this chapter.

The contribution of this chapter is as follows. We first introduce the novel control message load generator *hvbench* for dynamic and scalable generation of reconfigurations. Second, we introduce multiple NSAL performance models and select, based on experiments, the model which generalizes the best for different NSAL implementations and hardware platforms. Third, we propose an online machine learning pipeline (⑦) to train (④) the performance models and refresh the models in case of concept drift due to fluctuations of available resources.

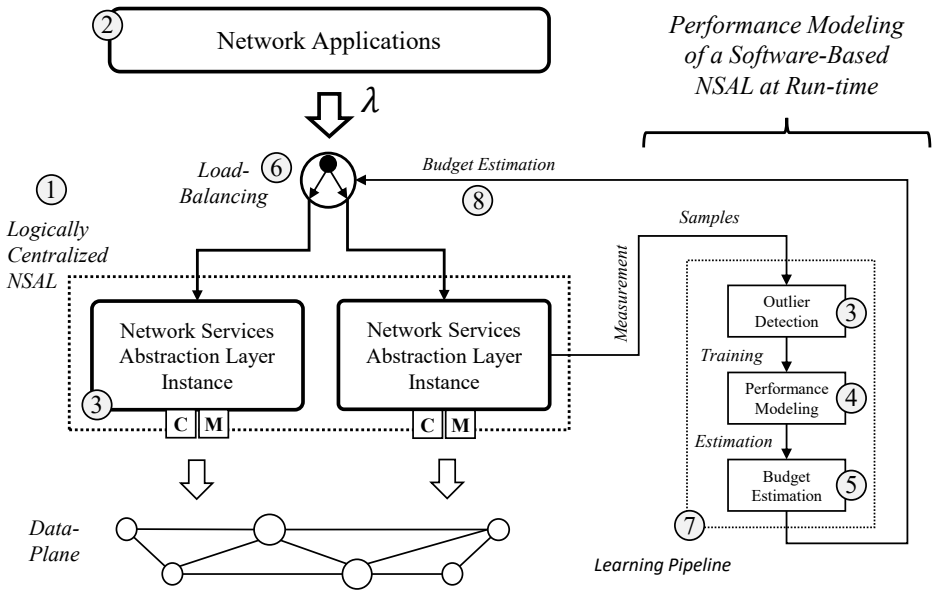


Figure 6.1: Performance modeling of a software-based NSAL at run-time. This chapter defines and evaluates a machine-learning pipeline (⑦) for the maximum reconfiguration rate estimation (⑧) of an NSAL instance. A load-balancer (⑥) distributes the reconfiguration load λ between the logically centralized (①), but physically distributed, NSAL instances (③). The budget estimation can be used by the load-balancer to decide on the required number of NSAL instances and the workload for each NSAL instance.

The concept drift is detected using outlier detection (③). Forth, we evaluate the pipeline by exploring the parameter space and give guidelines for different deployment scenarios. The proposed pipeline can be deployed as part of an autonomous load-balancing/orchestration layer which keeps track of current usage and capacity of running NSAL instances. This allows load-balancing of northbound control applications based on their control message rate and that way prevent over- and underutilization of the NSAL instances.

This chapter is structured as follows. Section 6.1 first defines the problem and challenges in detail. Section 6.2 then gives the background of this area of research. In Section 6.3 the formalized NSAL system model is presented. In Section 6.6 the proposed learning pipeline, the performance models and the chosen machine learning techniques are introduced. Section 6.7 discusses the experimental evaluation methodology and Section 6.8 presents the results of the evaluation. Section 6.9 concludes this chapter by deducing deployment guidelines from the findings and give an outlook on future work in this area.

The content of this chapter is based on the results presented in the following publications. [11] proposes and evaluates three models for the performance of an NSAL. From the three proposed models, the best performing model is chosen and implemented in the learning pipeline. The online learning pipeline is presented and evaluated in [18]. The dynamic load generator tool is presented in [13] and made available as an open source project.

[18] C. Sieber, A. Obermair, and W. Kellerer. "Online learning and adaptation of network hypervisor performance models." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 9 pages. Lisbon, Portugal, 2017. DOI: 10.23919/INM.2017.7987462.

[11] C. Sieber, A. Basta, A. Blenk, and W. Kellerer. "Online Resource Mapping for SDN Network Hypervisors using Machine Learning." In: *2nd IEEE Conference on Network Softwarization (NetSoft)*. 5 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502447.

[13] C. Sieber, A. Blenk, A. Basta, and W. Kellerer. "hvbench: An open and scalable SDN network hypervisor benchmark." In: *IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services*. 4 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502475.

6.1 Challenges and Problem Definition

The problem can be divided into the three distinct challenges: Finding an accurate NSAL performance model, training of the model and detecting concept drift with subsequent model refresh. Next, we discuss the three challenges in detail.

6.1.1 C5.1: Accuracy of the NSAL Performance Model

A logically centralized, but physically distributed, NSAL requires accurate budget estimation of each distributed NSAL instance to prevent under- or overutilization of instances. Budget denotes here the maximum rate of messages the NSAL can process. Each instance may vary in deployed software, software revision or underlying hardware, e.g., number of Central Processing Units (CPUs), or which type of hardware virtualization is deployed, e.g., *KVM* or *XEN*. Hence, it is of importance that the model generalizes well and provides high estimation accuracy under different deployment scenarios. Challenge C5.1 is therefore the process of developing and comparing different NSAL performance models.

6.1.2 C5.2: Training of the Model at Run-time

Hardware virtualization became ubiquitous in data-centers and allows convenient and secure co-location of software and whole operation systems on the same physical hardware. Resources, such as number of CPUs or Random Access Memory (RAM), can dynamically be added and removed to satisfy increased or decreased demand. However, for accurate estimations of the rate budget of the NSAL, the model parameters have to be fitted to the underlying platform.

One way to solve this are offline benchmarks of every NSAL instance in every deployed environment. However, this approach is not scalable to a larger number of platforms and NSAL implementations. Challenge C5.2 is therefore the online fitting of the model parameters to the underlying platform. To this end, a learning pipeline has to be designed which receives and filters measurement samples and fits the model parameters. Additionally, hyper-parameter of the pipeline, such as the samples weighting, have to be investigated for optimal estimation accuracy in different environments.

6.1.3 C5.3: Detecting Concept Drift and Refreshing the Model

Concept drift describes the decrease of the model's estimation accuracy over time. We focus on dynamic resource assignments as a cause of concept drift. There are multiple reasons for dynamic resource assignments in virtualized compute environments. For example, a dynamic resource increase can happen with vertical auto-scaling. There, the infrastructure increases the assigned resources when an application hits a predefined threshold. A temporary decrease in assigned resources can happen for example when higher priority tasks are scheduled. Virtual machine migration techniques are also known for decreasing resources to speed up the incremental state transfer between two physical machines. Other causes of concept drift can be changes of the operational parameters of the system, such as network transport tuning or CPU scheduling parameters.

Challenge C5.3 is therefore to not only learn an accurate NSAL performance model but also to recognize when the NSAL is subject to an increase or decrease in available resources, which can result in concept drift. Furthermore, false positive detections should be avoided while at the same time being sensitive enough to adapt the model fast to changes in available resources. The challenge is not trivial as there is a lack of insights into the underlying physical platform. From the perspective of the NSAL instance, the parameters of the hardware do not change but instead the hardware just exhibits faster or slower processing.

6.2 Background

To the best of the knowledge of the author there exists no other works in the area of online learning of NSAL performance at run-time. Often models are trained on offline samples or samples gathered of isolated application instances. We discuss first the state of the art regarding performance of NSAL implementations in general. Furthermore, we introduce the alternatives to the proposed *hvbench* load generator. Afterwards, we introduce the state of the art in the area of application resource demand estimation in cloud environments in general and of abstraction layers in particular.

General NSAL Performance / NSAL Benchmarking

Software-Defined Networking (SDN) network hypervisors such as OpenVirtX (OVX) [141] and FlowVisor (FV) [144] abstract a given SDN network and allow multiple SDN controllers to share this network. SDN network hypervisors are NSALs but with a focus on fully deployed SDNs-based networks and a focus on the control protocol OpenFlow. Thus, the state of art in the area of network hypervisors is relevant here. In [26, 29, 91, 103], the authors discuss and survey SDN network virtualization in general and the network hypervisor placement problem (HPP) specifically. HPP targets the amount of needed hypervisor instances and the placement of the hypervisor instances inside the network. In [30, 161], the control plane latency and monitoring overhead of different SDN network hypervisor architectures is evaluated. But these approaches are not applicable to our problem as per-message resource consumption and hardware details are not available. Furthermore, performance models based on offline benchmarks can not adapt at run-time to changes in the available computing resources or to new hardware platforms.

Resource Usage Modeling / Performance Prediction

A related area of research is the performance prediction and classification of applications through online machine learning from the perspective of the infrastructure provider.

In [159], Weingärtner et al. present a survey on forecasting and profiling of cloud applications for cloud resource management. The authors show the importance of cloud resource modeling and argue that cloud dynamism is one of the biggest challenges in this area. Two mechanisms in our proposed pipeline deal with this cloud dynamism. First, the gradual adaptation of the performance models at run-time. Second, the concept drift detection which triggers a re-training of the performance models. [55] use application profiling for dynamic scalability and contention prediction in public infrastructure. As with the other works, the models are training offline for a specific hardware platform without considering changes to at run-time. In [120], the authors propose and compare different approaches for per-application resource usage prediction. In [74, 126, 143], the authors propose machine learning pipelines to provide medium-term resource demand predictions and elastic resource scaling. In [70, 109, 160], the authors show that machine learning techniques can accurately predict future database query resource requirements based on previously monitored queries. [90] provides an extensive survey on anomaly detection in cloud environments in general. But anomaly detection on performance models is not considered in [90].

This chapter extends the state of the art by tackling the dynamic budget estimation problem from the point of view of an orchestrator with limited control and information about the underlying virtual resources and the resources assigned to the NSAL. Furthermore, a complete

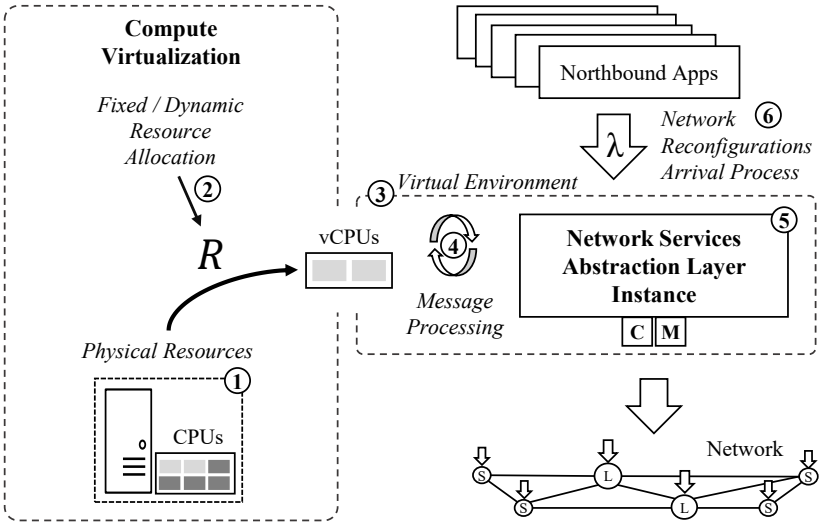


Figure 6.2: The figure illustrates the relationship between compute virtualization and the NSAL. A resource scheduler on the physical machine (①) allocates R physical resources (②) to the NSAL instance executed in a virtual environment (③), which uses its virtual resources to process and/or forward control messages (④) from/to the northbound applications and the network. The resource allocation (②) can be dynamic and be adapted at any time.

methodology is developed from the collection and filtering of samples, up to the budget estimation and detection of concept drift.

6.3 System Model

Next, we formalize the relationship between the physical resources, e.g., CPU, the virtualized NSAL instance and the reconfiguration rate. Figure 6.2 depicts the abstract system model. Table 6.1 summarizes the notation. The system model extends the system model from previous chapters by focusing on the underlying compute virtualization and processing resources. The figure shows that from the perspective of the compute on the left, the NSAL instance (⑤) is a process consisting of one or multiple threads executed inside a virtual environment (③), e.g., *KVM*. The virtual environment with the NSAL inside is assigned resources R (②) by the host's resource scheduler based on a fixed or dynamic scheduling strategy. For example, out of six physical CPUs, the scheduler assigns the NSAL two CPUs, so called *virtual* CPUs or Virtual Central Processing Units (vCPUs), which the NSAL is allowed to utilize each up to 50 %. We assume a dynamic scheduling strategy where the resource scheduler assigns resources dynamically between an upper limit R_{max} and a lower guaranteed limit of R_{min}. ΔR denotes a change of resource allocation. We consider compute resources, i.e., the CPU, as the only limited resource.

Table 6.1: Notation & Variables

Nomenclature	
$fm, ot(ot = \{fr, er, sp, sf, po\})$	Reconfiguration types.
λ	Total arrival rate of reconf.
$\lambda_{ot} = w_{ot} \cdot \lambda$	Arrival rate of *ot* reconf.
$\lambda_{fm} = w_{fm} \cdot \lambda$	Arrival rate of fm reconf.
$w_{ot}, w_{fm}, w_{ot} + w_{fm} = 1$	Reconf. type distribution in λ.
$R, R_{min}, R_{max}, \Delta R$	Resources assigned to NSAL.
B	Total reconf. budget of the NSAL.
$\rho, 0 \leq \rho \leq 1$	Instantaneous resource utilization.
$\rho_{max}, \rho_{max} \leq 1$	Max allowed system utilization.
ρ or $\rho(\lambda)$	Performance model.

From the perspective of the network control, the NSAL consists of one or multiple NSAL instances in between the network's control- and data-plane. The abstraction layer allows multiple network applications to share the control of the physical network. In OpenFlow, a network application can control the network by adding or updating forwarding rules in the devices using flow modification messages. Furthermore, it can request the current state of the network, such as throughput, by sending statistic request messages. Each of the reconfiguration or state requests has to pass through a NSAL instance, which uses its assigned resources to process the messages (④). We assume a negative exponential (*Poisson*) message arrival process with an average rate λ (⑥).

Some message types are computational more complex than others, e.g., simple echo requests compared to reconfigurations. For our evaluation, we consider the OpenFlow message types *feature request* (*fr*), *echo request* (*er*), *port stats request* (*sp*), *flow stats request* (*sf*), *packet out* (*po*) and *flow modification* (*fm*). During our experiments with OVX and FV, we noticed that flow modifications exhibit a much larger cost per message than each of the other message types. Hence, for the remainder of this chapter we summarize $\{fr, er, sp, sf, po\}$ as *other* (*ot*) and denote the combined message rate as λ_{ot}.

6.4 The *hvbench* Load Generator

There is a lack of scalable and realistic NSAL control message generators capable of emulating dynamic load scenarios. Hence, we present an extensible and distributed NSAL load generator framework based on flexible statistical request generators. The request generators are able to generate Poisson arrivals of custom distributions of control message types. The framework can be scaled out horizontally to multiple compute nodes that are centrally controlled and reconfigured at run-time.

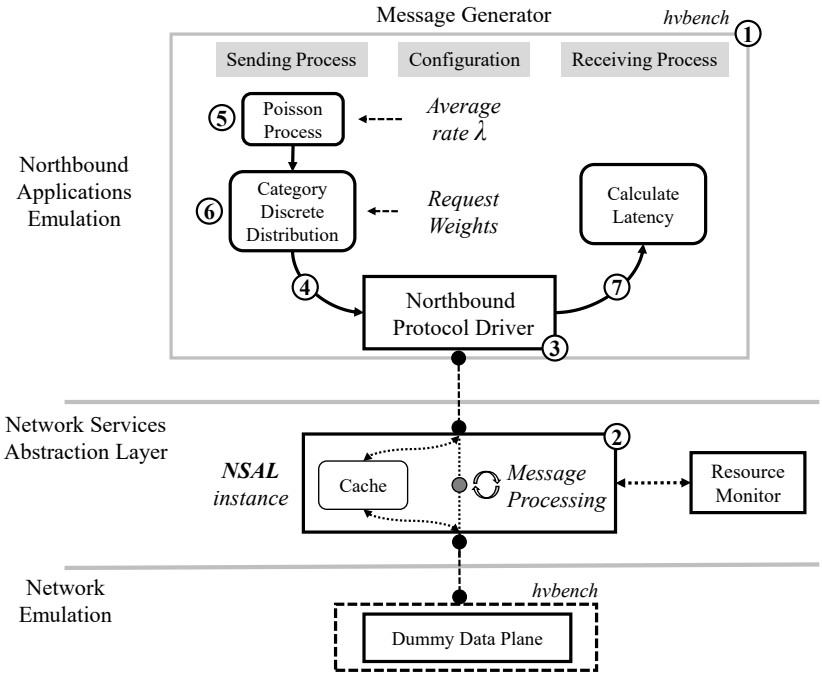


Figure 6.3: Detailed view of a *hvbench* load generator with a single northbound application including the message generator, one data plane node and the NSAL.

The architecture consists of four main modules. A configuration and directory service, distributed *hvbench* instances, a high-throughput messaging bus and host monitors. A command-line front-end and Python bindings simplify the usage of the distributed architecture. *hvbench* instances are containers for multiple, independent, control applications and emulated forwarding devices. When the distributed *hvbench* instances are started, the instances register at the configuration and directory service. We use *etcd* [175] as configuration directory service for its simplicity and support of change notifications. From this service, they receive the run-time configuration, e.g., number of control applications per instance, and report their host configuration, e.g., the host's Internet Protocol (IP) address. The configuration for a specific instance is identified by an Identifier (ID) given to each instances on start-up. Additionally, *etcd* pushes run-time configuration changes, e.g., an update to the average message rate, to the *hvbench* instances and the instances in return report their utilization to the directory service.

A message bus distributes injected data plane packets received from the NSAL between the data plane nodes (*packet out*). This allows topology discovery via Link Layer Discovery Protocol (LLDP) packets, as used for example by FV and OVX. We use Apache *kafka* [169] as message bus because of its low latency properties and scalable architecture. For handling the control connections and message parsing, denoted as *OF driver* for the control applications and *OF agent* for the data-plane node, we use a customized version of libfluid [168]. The host monitor uses a database of known implementation names to identify and monitor NSAL

processes. From the process it collects performance metrics such as CPU utilization as reported by the underlying Linux kernel.

Figure 6.3 gives a detailed overview of a set-up consisting of one *hvbench* instance (①) and one NSAL instance (②). A northbound application is described by the used northbound protocol driver (③), e.g., OpenFlow, and the used request generator (④). As of now, two types of request generators are implemented for a northbound application. A constant message generator generates requests of one specific type with a specific rate, e.g., 1000 feature requests per second. A probabilistic generator generates different requests based on a set of weights and an average rate λ. At first, the sending process in the probabilistic generator chooses the point in time of the next message to send based on a Poisson process with exponential distributed inter-arrival times and an average rate of λ (⑤). Next, the type of the requests to be generated is determined by a categorical (discrete) distribution based on a set of weights (⑥). The average rate and the weight of each request type can be adjusted at runtime.

In the receiving process of the northbound application, we measure the latency of reconfiguration messages (⑦). We use messages with replies such as echo requests to calculate the round-trip time. Measured one-way latency depends on the quality of clock synchronization in cases the network emulation and the northbound application are not located on the same physical machine. The data-plane nodes emulate the OpenFlow agents and answer messages either with random values, e.g., port statistics, or with pre-configured values, e.g., feature requests.

6.5 Performance Model

In the following we evaluate three different performance model candidates for the learning pipeline. A performance model has to a) make accurate estimations of the reconfiguration budget, b) converge fast on unknown or changed hardware platforms, and c) avoid overestimation of the available budget.

We define $\rho(\lambda)$ as the (injective) relationship between the message arrival rate λ and the resulting utilization of the NSAL instance. For example, $\rho(10000) = 1$ describes a resource usage of 100 % of the resources R assigned to the NSAL at a message rate of 10000 messages per second. We denote ρ^{-1} as the inverse of ρ, which translates resource usage to λ. The message rate an instance can process at specific maximum utilization ρ_{max} is denoted as budget B:

$$B = \rho^{-1}(\rho_{max}) \tag{6.1}$$

We set $\rho_{max} = 0.90$ as target utilization for the estimation in the evaluation. A target of 90 % allows for inaccuracies in the prediction and leaves headroom for an unexpected increase

in the control message rate. Thus a target of 90 % minimizes the risk of over-utilization and the consequential increased control delay. Accordingly, the budget is defined as $B = \rho^{-1}(0.90)$. Next we discuss the model candidates for ρ in detail.

6.5.1 Model Candidates and Online Fitting Method

We define three performance models for the NSAL CPU utilization ρ. The first two models are linear models with one or multiple dependent predictors. The third model is an exponential model. The requirements to a model are twofold. First, the model has to fit to the ground truth, i.e., to the samples gathered offline. Second, the model must be able to estimate the control message rate budget on potentially unknown hardware platforms not seen during training. Furthermore, the estimation has to converge to a low error state even if only unevenly distributed training samples are available, e.g., in the case where only a low utilization was observed so far.

Linear Model

Equation 6.2 defines the linear model ρ_{lin} where ρ depends on the total average messages per second λ and on the model coefficients β_1 and β_2. β_1 represents the processing cost per one message and assumes that the processing cost for all message types is equal. This model is based on the intuition that the CPU utilization of the NSAL depends solely on the messages to be processed and that each message has a fixed processing cost.

$$\rho_{lin}(\lambda) := \beta_1 \cdot \lambda + \beta_2 \tag{6.2}$$

Multiple Linear Model

The multiple linear model extends the simple linear model and introduces a breakdown of the message rate into control message types. Equation 6.3 describes the model ρ_{lin2} where ρ depends on the message rate λ_o and cost β_o for each of the message types ($\forall o \in O$). Hence, this model is able to capture the difference in processing costs of each message type.

$$\rho_{lin2}(\lambda) := (\sum_{o \in O} \beta_o \cdot \lambda_o) + \beta_I \tag{6.3}$$

Exponential Model

The third model (ρ_{exp}) in Equation 6.4 assumes an exponential relationship between the message rate λ and ρ. $\beta_2, \beta_I, \beta_b$ are constant offsets. The model is based on the intuition that processing of batches of control messages becomes more efficient with increased message rate. For example it is more efficient to collect multiple data packets from the networking

interface at once than single packets, known as *batching* [171]. Hence, the model describes an decreasing cost per message with higher message rates.

$$\rho_{exp}(\lambda) := \beta_c \cdot (1 - e^{-\beta_a \cdot \lambda}) + \beta_b \tag{6.4}$$

Iterative Fitting Process and Initial State

Next, we describe the iterative fitting process to estimate the model coefficients β_0, β_1, β_o, β_a, β_b and β_c during run-time. Subsequently, we denote the CPU utilization estimated by the models at point in time t as ρ^t and ρ^t_{meas} as the measured CPU utilization at time t. β^t describes the value of β at time t in the learning process. Based on an initial training, we define the following initial state for the model coefficients β. In the initial training, ground truth of the utilization at varying message rates and on different hardware platforms was collected. Afterwards, the coefficients of the proposed models where fitted based on the collected data. Thus the coefficients represent the average over multiple platforms. For ρ_{lin}, $\beta_1 = 0.29 \times 10^{-4}$ and $\beta_2 = 0.1$. For ρ_{lin2}, $\beta_o := 1 \times 10^{-5}, \forall o \in O$ and $\beta_I := 0.1$. For ρ_{exp}, $\beta_a = 0.26 \times 10^{-4}$, $\beta_b = 0.201$, and $\beta_c = 1.004103$.

The online learning is an iterative process structured in three phases. In the first phase, new samples are collected. The resource monitor continuously gathers a new sample each second ($l = l + 1$). Second, fitting the model function(s) to the collected samples. Third, the models are updated based on the fitting results. The process is continuously repeated when a new measurement sample is available.

We restrict the adaptation of β per iteration to 10 % ($\beta^{t+1} \in [\beta^t \cdot 0.9, \beta^t \cdot 1.1]$) to make the process more robust against short-term fluctuations or fitting errors. For fitting the models, we use weighted Orthogonal Distance Regression (ODR) as introduced by Boogs et al. in [31]. ODR considers measurement errors in the input, e.g., clock drift or inaccuracy in the sending process of *hvbench* or in the resource monitor, and output dimension, e.g., model discrepancy.

6.5.2 Testbed Configurations

In order to test how well the proposed performance models generalize to different hardware platforms and virtualized environments, we perform a study with three different configurations. Table 6.2 gives details on the configurations. The first configuration *C1* is a virtual machine based on *VirtualBox* which has two cores of an Intel i7-4470 CPU at 3.40 GHz assigned to it. For the second configuration *C2*, we reduce configuration *C1* to one CPU core. Configuration *C3* is a physical platform with 2 cores of a i7-4790 CPU at 3.0 GHz.

Table 6.2: Hardware Configurations

C1:	Virtual, 2 x i7-4770 CPU 3.40GHz
C2:	Virtual, 1 x i7-4770 CPU 3.40GHz
C3:	Physical, 2 x i7-4790 CPU 3.60GHz

6.5.3 Model Accuracy and Convergence Time

Next, we evaluate the estimation accuracy of the three models with the three testbed configurations (Table 6.2) and the two implementations OVX and FV. Furthermore, the software switch OpenvSwitch (OVS) as a data-plane implementation is evaluated to test how well the model generalizes to other use cases. Figure 6.4 shows the convergence times of the models in a scenario where the reconfiguration rate is increased by 200 messages per second every 5 s, starting at 0 messages per second. The convergence time is the duration until the model can estimate the reconfiguration rate budget within 5 % CPU utilization. Hence, the figure illustrates how suitable and how fast the models are for the estimation of unseen utilization values. The lower axis of the figures depicts all combinations of hardware configurations and NSALs. The left axis gives the convergence time in seconds. The whiskers indicate the range of the values, the box indicates the 25 % and 75 % quartiles, respectively, and the line the median of the values.

From the figure we conclude that ρ^{exp} can estimate the CPU usage of the NSALs at about 200 s into the experiment for most combinations. Furthermore, the linear model, which does not distinguish the request types, outperforms the second linear model, which considers the types of the requests. The results also show that there is a difference between the different configurations. While the linear models are good in predicting the usage for OVS in the virtual environments, for the hardware environment *C3*, the prediction quality is significant lower. For FV on the physical hardware, the ρ^{lin2} and the exponential model show a long and highly varying convergence time compared to the linear model ρ^{lin}.

6.5.4 Over- and Underestimation

Next, we take a closer look at whether the three models under- or overestimate the CPU utilization and we quantify the under- or overestimation for each model for one configuration. Overestimating the future CPU utilization can result in degrading the overall performance, e.g., slow message processing or message rejection, and it could also result in an inefficient resource utilization due to over-dimensioning. On the other hand, underestimation can result in an overload of the NSAL instance by accepting messages exceeding the CPU capacity. The under- or overestimation are calculated as follows:

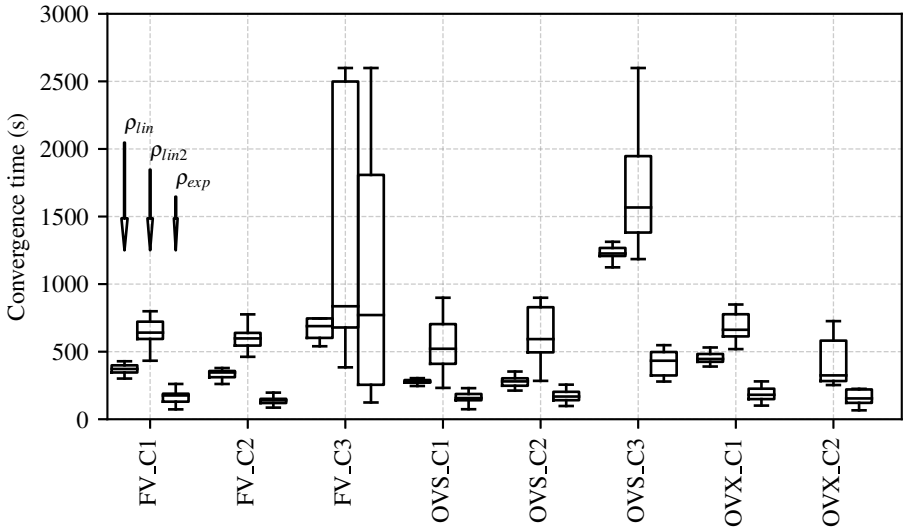


Figure 6.4: Experiment time t where the Mean Squared Error (MSE) between the model and the measured CPU time is and stays lower than threshold 0.05. Lower values are better. ρ_{exp} outperforms the ρ_{lin} and ρ_{lin2} models for most configurations.

$$X_m^t := \{\forall \alpha \in \{t, t+1, .., l\} | \rho_{meas}^{\alpha} - \rho_m^t(r^{\alpha}) > 0\} \tag{6.5}$$

$$\phi_m^t := \frac{\sum_{x \in X_m^t} (\rho_{meas}^{\alpha} - \rho_m^t(r^{\alpha}))}{|X_m^t|} \tag{6.6}$$

$$Y_m^t := \{\forall \alpha \in \{t, t+1, .., l\} | \rho_{meas}^{\alpha} - \rho_m^t(r^{\alpha}) < 0\} \tag{6.7}$$

$$\psi_m^t := \frac{\sum_{x \in X_m^t} (\rho_{meas}^{\alpha} - \rho_m^t(r^{\alpha}))}{|X_m^t|} \tag{6.8}$$

X_m^t in Equation 6.5 defines the points in time where the model $\rho_m^t, m \in M$ at experiment time t underestimates the future CPU utilization $\rho_{meas}^x, \forall x : x > t$. In Equation 6.6, ϕ_m^t defines the mean absolute underestimation of model m starting from time t till the end of experiment time l. Y_m^t and ψ_m^t in Equations 6.7 and 6.8 define the mean absolute overestimation, analogous to X and ϕ for underestimation.

Figure 6.5 depicts ϕ and ψ for OVX on hardware configuration *C1* over experiment time t. Overestimation ψ is shown on the top of the figure, i.e., positive values > 0. Underestimation ϕ is shown on the lower part of the figure, i.e., negative values < 0. From the figure we conclude that ρ_{lin} and ρ_{exp} both underestimate the utilization in this configuration. ρ_{lin2} exhibits mostly overestimation. In terms of convergence speed, both, ρ_{lin} and ρ_{exp}, decrease the underestimation at a similar rate starting from experiment time $t = 100\,\text{s}$. For the

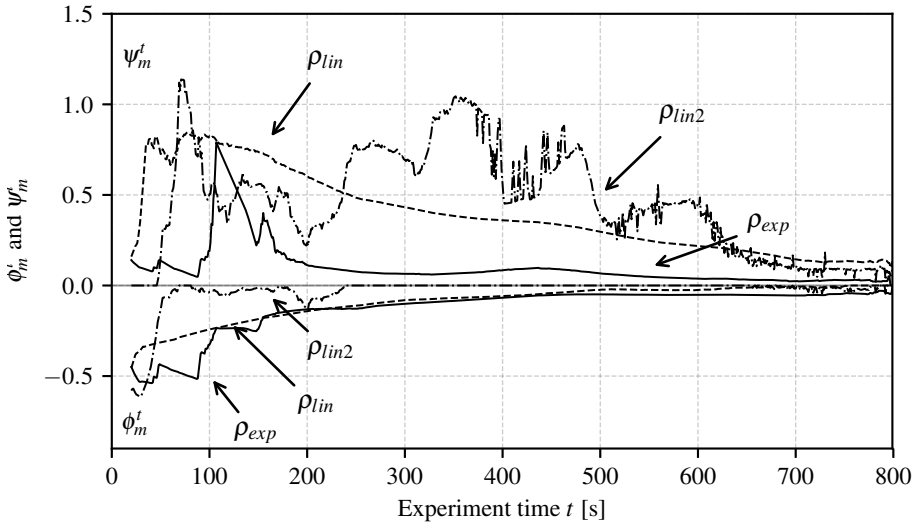


Figure 6.5: Mean absolute over- and underestimation of the models over time for an OVX instance running on hardware configuration *C1*. The top part of the figure depicts the overestimation, the bottom part the underestimation. Overestimation can lead to inefficient resource usage, while underestimation can overload the NSAL, thus effecting all northbound applications.

overestimation case, the figure shows that ρ_{lin} and ρ_{lin2} exhibit high overestimation compared to ρ_{exp}. In addition to high overestimation, ρ_{lin2} shows an oscillating behavior. ρ_{exp} can decrease the amount of overestimation soon into the experiment and can keep it stable at a low level starting from $t > 130\,\text{s}$. Although the results of the figure can not be generalized to all hardware configurations, the results give important indications on the under- and overestimation characteristics of the three models.

6.5.5 Model Selection

In the preceding sections we compared different NSAL performance models for ρ. The results show that a negative exponential performance model can describe the performance of an NSAL accurately. Hence, we choose the following negative exponential model for the learning pipeline:

$$\rho(\lambda) := \Theta_c \cdot (1 - e^{-\Theta_a \cdot \lambda}) + \Theta_b \tag{6.9}$$

While in general the model yields good results, it exhibits unstable behavior in cases where a majority of the samples report a low utilization, e.g., $\leq 15\,\%$. This comes from the fact that there are many possible regression results for the coefficients Θ_b and Θ_c which fit well to low utilization samples but do not allow accurate estimation of the budget at ρ_{max}. Hence, based on training experiments, we fix Θ_b to 0 and Θ_c to 1.6. Despite the fixed coefficients, the model can still adapt through the coefficient Θ_a.

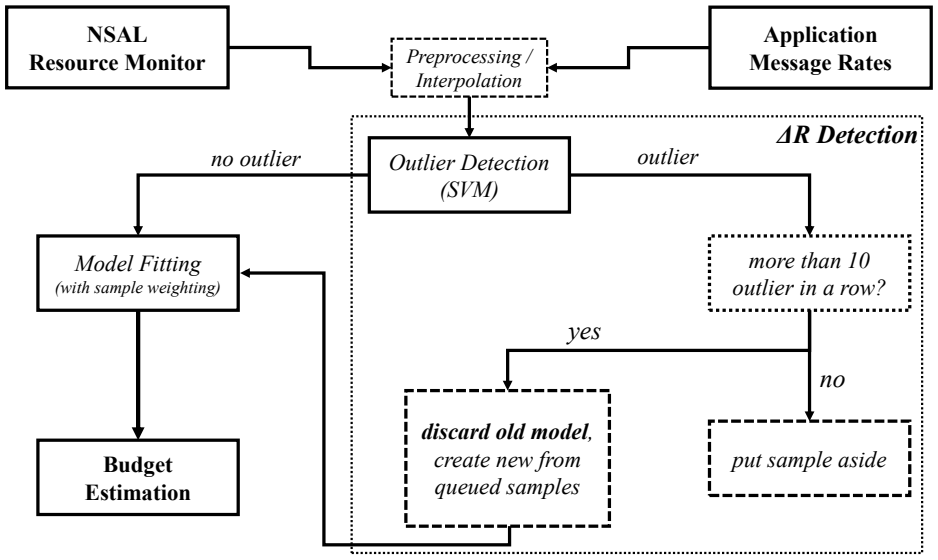


Figure 6.6: Proposed NSAL performance learning pipeline. Input parameters are the resource utilization and the northbound application message rates. Output is the estimation of the current message rate budget. Outlier detection is used to invalidate the current model in case of significant changes in available resources ΔR.

6.6 Proposed Learning Pipeline

In the following, we introduce the proposed learning pipeline. The objective of the learning pipeline is to estimate the maximum performance in terms of reconfiguration message rate λ the NSAL instance can process with the currently assigned resources R. This maximum rate is denoted as the message budget B. Furthermore, it must detect and adapt the reported message budget to resources fluctuations ΔR. This adaption can be gradual, by decreasing the importance of older samples compared to more recent samples. This approach does not require to detect ΔR. Or the adaptation can be rapid, by implementing ΔR-detection, which detects ΔR and, if necessary, discards the current model.

We first give a general overview of the elements in the learning pipeline, including the ΔR-detection based on an Support Vector Machine (SVM). Afterwards, we discuss the implementation of the performance model based on the overall message rate λ from the previous section and how we adapt it to the pipeline. Subsequently, we discuss the sample weighting function for the gradual adaptation. At the end of this section we discuss an extended performance model which distinguishes different message types (λ_{fm}, λ_{ot}).

Figure 6.6 presents the proposed pipeline. There are two input data sources. One is the NSAL resource monitor which measures the exact amount of cumulative CPU time used by the NSAL. The second one is the cumulative message counters provided by the NSAL which denote how many messages are processed by the NSAL per message type and per application.

Resource usage and message counter values are provided unsynchronized with a frequency of 1 Hz each. The pipeline consists of the three high-level steps: i) Collecting and preprocessing of the monitoring samples, ii) ΔR-detection to check for concept drift and if necessary discard the current model and iii) the model fitting and budget estimation.

First Step: Collecting and Preprocessing Monitoring Samples

The samples are then processed as follows. The first step is combining the asynchronously collected samples of cumulative resource usage $\Sigma\rho$ and message counters $\Sigma\lambda$ by linear interpolation. For this, two cumulative resource usage samples before and after the point in time t of a message rate sample $s^t_{\Sigma\lambda}$ are taken, $s^{<t}_{\Sigma\rho}$ and $s^{>t}_{\Sigma\rho}$. Afterwards, the cumulative resource usage sample $s^t_{\Sigma\rho}$ for the message rate sample $s^t_{\Sigma\lambda}$ is linearly interpolated. As a result, we have a sample $(s^t_{\Sigma\lambda}, s^t_{\Sigma\rho})$ of the message counter with the approximate *cumulative* resource usage for those messages at time t. Finally, the instantaneous resource usage s_ρ for the message rate at t, i.e., s_λ, is calculated by element-wise subtraction $(s^t_{\Sigma\lambda}, s^t_{\Sigma\rho}) - (s^{t-1}_{\Sigma\lambda}, s^{t-1}_{\Sigma\rho}) = (s_\lambda, s_\rho)$.

Second Step: ΔR-detection and Model Invalidation

In the second step, the ΔR-detection checks if the current performance model is still valid and not invalidated by a change in resources ΔR. To do so, it calculates the difference between the sample (s_λ, s_ρ) and the current performance model $\rho()$ by $\rho(s_\lambda) - s_\rho$. On this error, it applies a one-class SVM [117] for outlier detection. The SVM is configured with a Radial Basis Function (RBF) kernel, $\gamma = 0.1$ as kernel parameter and we consider 10 % of the training data as outliers ($\nu = 0.1$). Then, if none of the samples in the last $T_{thres} = 10$ seconds fit to the current model, we assume a big resource change ΔR and invalidate the current model. If no resource change is detected, the sample is added to the previous samples of the current model and the model is re-trained with updated sample weights where old samples become less important than the newer samples. A maximum of 120 samples, equal to the last 120 s, are stored and older samples are discarded. The idea behind this is, that if there is a small ΔR which can not be detected by the ΔR-detection, we instead adapt the model over time using the sample weighting function and the limited sample memory of 120 s.

Third Step: Model Fitting and Budget Estimation

The training of the model uses ODR [31] which considers errors in the data in both dimensions, i.e., in the measurement of the resource usage and the message counters. In the following we introduce the performance model p, the sample weighting function $w(t)$ for gradual adaptation, the extended performance model p_{ext} and the budget B in detail.

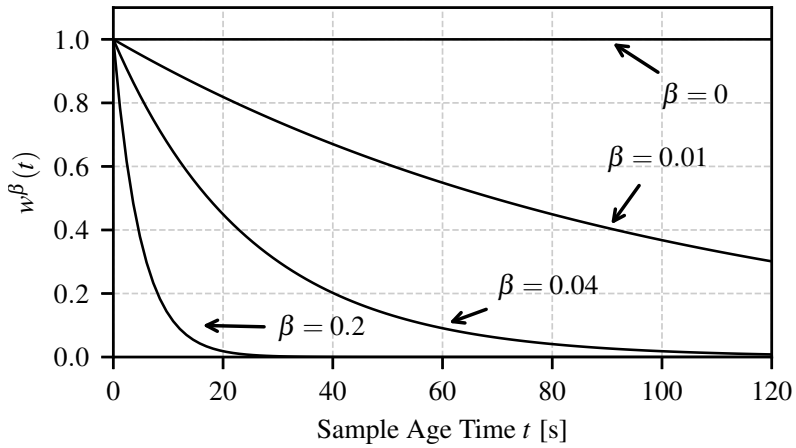


Figure 6.7: Sample weighting function $w(t)$ (Eq. 6.10) describes the important of each sample in the regression depending on the age t of the sample in seconds. Parameter β describes how fast the weights of the samples decrease. The figure illustrates w for $\beta = [0.0, 0.2, 0.04, 0.01]$. For $\beta = 0$, w becomes $w(t) = 1$, which makes the weight of a sample independent of its age. For $\beta = 0.2$, the weight of a sample becomes half after five seconds.

6.6.1 Gradual Adaptation and Sample Weighting Function

If the resources R assigned to the NSAL instance do change, but the ΔR-detection does not detect it, the performance model has to incrementally adapt over time. We perform the gradual adaptation of the model by reducing the importance of each sample in the regression based on its age. In Eq. 6.10 we define a function $w(t)$ which describes a negative exponential decay process. t denotes the age of the sample in seconds. The negative exponential decay process was chosen based on preliminary experiments.

$$w^\beta(t) := e^{-t \cdot \beta} \tag{6.10}$$

Figure 6.7 illustrates w for four different values of β. For $\beta = 0$, w becomes $w(t) = 1$, which makes the weight of a sample independent of its age and all 120 samples are equally important in the regression. For $\beta = 0.2$, the importance of a sample decreases fast so that after approximatively 5 seconds, a sample has half of the weight of a new sample. We evaluate the impact of the weight function on the adaptation for 15 different values between 0.0 and 0.2 ($\beta = [0.0, 0.015, .., 0.2]$).

6.6.2 Extended Performance Model

The performance model discussed so far only considers the relationship between the total message rate λ and the resulting utilization ρ. Hence, in cases when it is desired to differentiate

the cost of different message types, the simple exponential model is not sufficient. Next, we introduce an extended exponential model which considers λ_{fm} and λ_{ot} separately.

The following equations define the extended performance model. The model is made up of separate equations for λ_{fm} (Eq. 6.11) and λ_{ot} (Eq. 6.12) with each consisting of two parts, a linear part and a negative exponential part. The two parts are added together using the parameters A, B, C and D:

$$\rho_{ext,fm}(\lambda_{fm}) = A \cdot (\Theta_A \cdot \lambda_{fm}) + B \cdot \Theta_B \cdot (1 - e^{-\Theta_C \cdot \lambda_{fm}}) \tag{6.11}$$

$$\rho_{ext,ot}(\lambda_{ot}) = C \cdot (\Theta_D \cdot \lambda_{ot}) + D \cdot \Theta_E \cdot (1 - e^{-\Theta_F \cdot \lambda_{ot}}) \tag{6.12}$$

Equation 6.13 describes the total utilization as sum of the utilizations induced by λ_{fm} and λ_{ot} message rates:

$$\rho_{ext}(\lambda_{fm}, \lambda_{ot}) = \rho_{ext,fm}(\lambda_{fm}) + \rho_{ext,ot}(\lambda_{ot}) \tag{6.13}$$

The parameters A, B, C and D allow for fine-tuning of the performance model between linear and negative exponential behavior. From preliminary experiments for this work we concluded that this gives the best possible result. As depending on the platform, NSAL and message type (λ_{fm} or λ_{ot}), the performance behavior varies between linear and negative exponential behavior. $B = 0.7$ and $D = 0.7$ provided the best balance between linear and negative exponential model in our experimental environments. Therefore we set $B = 0.7$ and $D = 0.7$ in the evaluation. Following the constraints $A + B = 1$ *and* $C + D = 1$ we set $A = 0.3$ and $C = 0.3$.

To learn the parameters Θ_A, Θ_B, Θ_C and Θ_D from the samples, we use the two equations Eq. 6.14 and Eq. 6.15. Two independent ODR regressions are used to learn the coefficients of both equations.

$$\rho_{ext,lin}(\lambda_{fm}, \lambda_{ot}) = (\Theta_A \cdot \lambda_{fm}) + (\Theta_D \cdot \lambda_{ot}) \tag{6.14}$$

$$\rho_{ext,exp}(\lambda_{fm}, \lambda_{ot}) = \Theta_B \cdot (1 - e^{-\Theta_C \cdot \lambda_{fm}}) + \Theta_E \cdot (1 - e^{-\Theta_F \cdot \lambda_{ot}}) \tag{6.15}$$

6.7 Evaluation Methodology

In the following we present the methodology used for the evaluation of the proposed pipeline. We first present a general overview of the experimental set-up. Afterwards, we discuss how

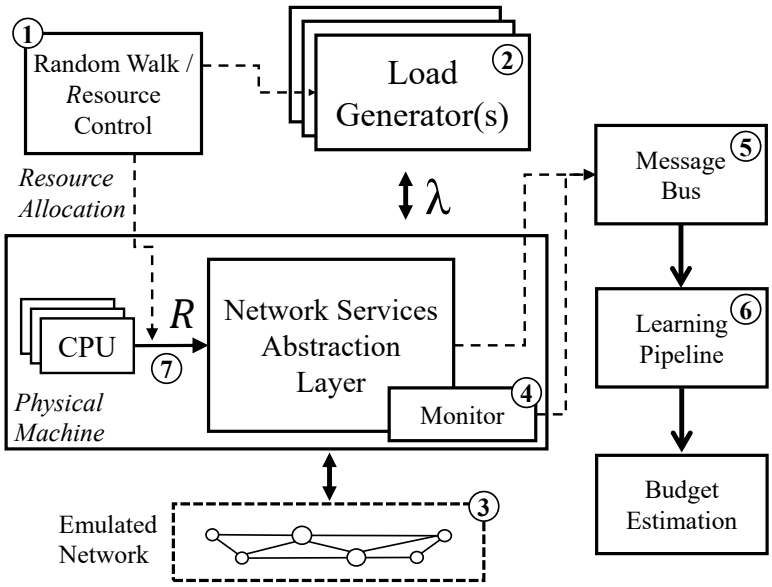


Figure 6.8: Experimental set-up consisting of the load generator *hvbench* (②), a simple emulated network (③), the NSAL resource monitor *monitor* (④) and a message bus (⑤). The message bus distributes the measurement samples to the learning pipeline (⑥), which in turn outputs the model for the budget estimation.

we evaluate the accuracy of the budget estimation. At the end of this section, we show the message arrival process we use to emulate northbound applications and their reconfiguration or statistics messages.

6.7.1 Experimental Set-up

Figure 6.8 describes the experimental set-up consisting of the load generator *hvbench* [13] (②), a simple emulated network (③), the resource monitor [13] (④) and a message bus (⑤) (implemented with *kafka* [169]), which distributes the measurement samples to the learning pipeline (⑥). The pipeline in turn outputs the model for the budget estimation. A control component adjusts the overall message rates and message type mixes based on a random walk process (①). Furthermore, it adjusts the assignment of R to the NSAL instance (⑦). For the set-up, the resource R assignment is implemented by scaling the CPU frequency of the NSAL host through the Linux CPU governor. For example, the experiment is started with assigning the maximum possible resources $R_{max} = 3.2\ GHz$ to the NSAL process. After 60 s, R is reduced to $\frac{R_{max}}{2} = 1.6\ GHz$. For this, the *performance* CPU governor has to be activated so that the Linux kernel always scales the CPU to the maximum configured CPU frequency.

During the experiment, the *monitor* component queries the total time the NSAL instance used the CPU ($s^t_{\Sigma\rho}$) with an accuracy of 10 ms and sends it to the message bus. Furthermore, the simple emulated network answers all messages it receives from the NSAL with static,

pre-configured, responses. Additionally, *hvbench* reports the message counters ($s^t_{\Sigma\lambda}$) to the message bus. The following hardware and software set-up is used for the evaluation presented in this paper: *Intel(R) Core(TM) i5-3470* CPU with a maximum frequency of 3.20 GHz, 8 GB RAM and *Ubuntu 14.04.4 LTS*. FV is used as NSAL in version 1.4.0, *hvbench* as load generator and *hvmonitor* monitor in version 0.1.0.

6.7.2 Budget Estimation Error

The budget estimation error ϵ is defined as the difference between the true budget, denoted as *ground truth*, and the estimated budget. We use offline benchmarks to determine an approximation of the true budget of a specific NSAL on a specific hardware platform. For this, we linearly increase the message rate λ in small steps until the monitor component reports an average utilization of ρ_{max}. Based on the results of the offline benchmark, we can define the relative estimation error ϵ as: $\epsilon = \frac{|GroundTruth - B|}{B}$

6.7.3 Load Generation Process

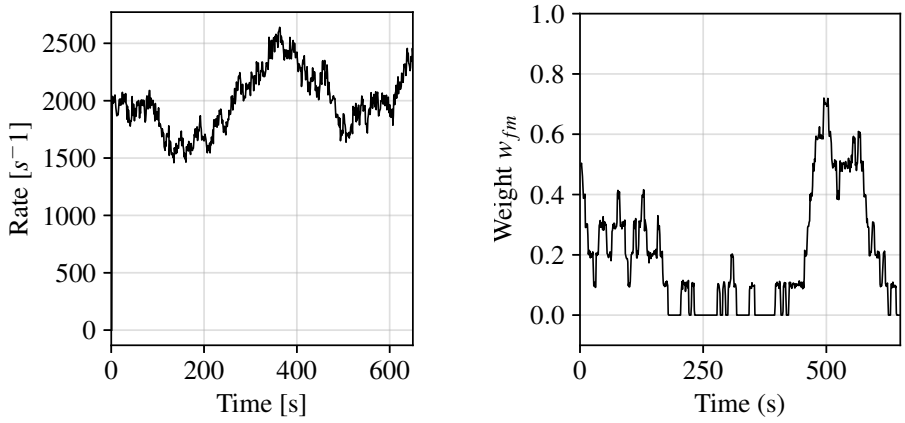
We configure *hvbench* with a Poisson message arrival process and we use two random walk processes to model the change of λ and (w_{ot}, w_{fm}) over time. At the beginning of the experiment run we set λ to $\rho^{-1}(0.5)$ based on the ground truth. Then, each 5 s we use random number generator to decide to keep λ constant with a chance of 40 % or change it with a chance 60 % according to the following rule:

$$\lambda = \begin{cases} \lambda \cdot 0.9 & \text{if } \rho(\lambda) > 0.5 \\ \lambda \cdot 1.1 & \text{if } \rho(\lambda) < 0.5 \end{cases}$$

When the random walk process of the message type distribution is activated, we adjust (w_{ot}, w_{fm}) also every 5 s. With a chance of each $\frac{1}{3}$, we either decrease or increase w_{fm} by 10 % of the previous value. With a chance of $\frac{1}{3}$, we keep w_{fm} constant. w_{ot} is updated accordingly $(w_{ot} + w_{fm} = 1)$. An example of the generated rates and weights of the random walk process is given in Figure 6.9. At $t = 0$, we start with $\lambda_T = 2000 s^{-1}$. Until about 200 s into the experiment, the rate decreases. From 200 s the message rate increases up to 400 s into the experiment with a peak message rate of about 2600.

6.8 Evaluation

Next we evaluate the effectiveness of the proposed pipeline by using the methodology defined in the previous section. The main performance metrics here are the relative budget estimation



(a) Message Rate (λ) (b) Message Weight (w_{fm})

Figure 6.9: Example load generation process for a) the message rate and b) the message type weights. At $t = 0$ the process is initiated with a rate equal to 50 % NSAL utilization and with a message weight of $w_{fm} = 0.5$. Every 5 s a new value for the rate and weights is generated by the random walk process.

error ϵ and the model convergence time after a change in resources. If not otherwise stated, the experiments were conducted in the described test-bed (Section 6.7). For space reasons we focus on the results for FV as NSAL implementation. The results for OVX do not considerable differ from the presented ones.

6.8.1 Budget Estimation Error without ΔR-Detection

We fist discuss the budget estimation error ϵ for different combinations of β and constant available resources R. We evaluate β in the range of $[0, 0.2]$ with a step size of 0.015 and R in the range of $[1.6, 3.2]$ with a step size of 0.1. The result of a combination of β and R is presented as median over 20 runs. For each run, after a warm-up phase of 120 s, a sample is taken each second for a period of 180 s and the error averaged over all samples. We use the random walk process described in Section 6.7.3 for the message rate λ. w_{ot} and w_{fm} we keep constant at $w_{ot} = 0.5$ and $w_{fm} = 0.5$.

Figure 6.10 illustrates the mean estimation error over the evaluated parameter space of β and R. The results show that on average the estimation error is 5.8 % with a standard deviation of 2.5 %. Furthermore, we conclude from the figure that the estimation error depends on R. For example, while for $R = 1.6$ we observe ϵ to be 3.8 % on average, ϵ for $R = 3.0$ is on average 9.5 %. The lowest error can be observed for $R = 1.6$ with $\epsilon = 2.6$ % Additionally, the figure suggests a minor correlation between the estimation error and parameter β. However, the maximum (Pearson) correlation we observe is for $R = 2.6$ with 0.18.

Two main conclusions can be drawn from the figure. First, we observe only an insignificant influence of β in the evaluated parameter range. Therefore, β can be chosen freely in the

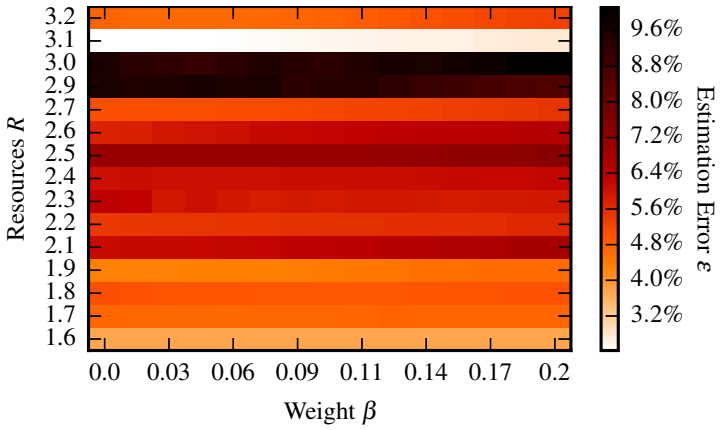


Figure 6.10: Mean estimation error for resource values $R = [1.6, 3.2]$ and sample weighting factors $\beta = [0, 0.2]$ for constant allocated resources during the experiment. $R = 1.6$ exhibits the lowest error with $\epsilon = 2.6\,\%$. $R = 3.0$ with $\epsilon = 9.5\,\%$ the highest. No correlation between e and β can be observed.

evaluated range in cases when R is constant. This gives freedom in the choice of a β value. Second, the error depends on R and ranges between 2.6 % and 9.5 %. Next, we discuss the influence of parameter β on the convergence time in scenarios where R varies over time.

6.8.2 Convergence Time after ΔR-Detection

In compute environments with dynamic assigned resources R, it is important for the learning pipeline to quickly adapt to changes of R. We focus on the use case of a sudden decrease of R and measure the time period between the decrease and the point in time the model reaches a relative estimation error ϵ of less than 10 % again. First, we illustrate the convergence time by example for a gradual adaptation through sample weighting, afterwards we explore the parameter space with and without ΔR-detection.

Figure 6.11 illustrates the convergence time by example for four different values of the sample weighting factor β, with a sudden decrease in CPU frequency of 1.4 GHz ($\Delta R = 1.4$) and a relative error threshold of 10 % ($\epsilon = 0.1$). A training phase with a duration of 125 s is omitted in the figure. At 175 s into the experiment, the available resources are decreased from 3.0 GHz to 1.6 GHz. Two observations can be made from the figure. First, up to 175 s into the experiment all four model instances can estimate the available budget with a high accuracy of $\epsilon \leq 0.04$. Second, the choice of β has a significant influence on the convergence time. For $\beta = 0.2$, which only considers recent samples, the model adapts rapidly (7 s) to the new resources. However, as few samples have a strong influence on the regression, we observe a higher variance in the figure. For $\beta = 0.0$ it takes 99 s to reach the low-error threshold and no variance is visible.

Convergence Time without ΔR-Detection

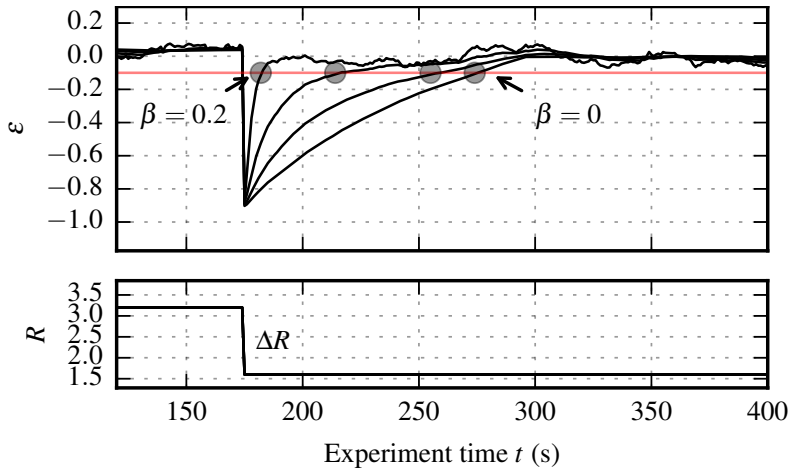


Figure 6.11: Convergence time after a resource change $\Delta R = 1.5$ for the sample weighting factors $\beta = [0.0, 0.0143, 0.0429, 0.2]$. The horizontal (red) line marks an low error threshold of $\epsilon = -0.1$. Circles mark the time when the model reaches the error threshold. Higher values of β result in fast convergence, but also in instability in terms of estimation accuracy.

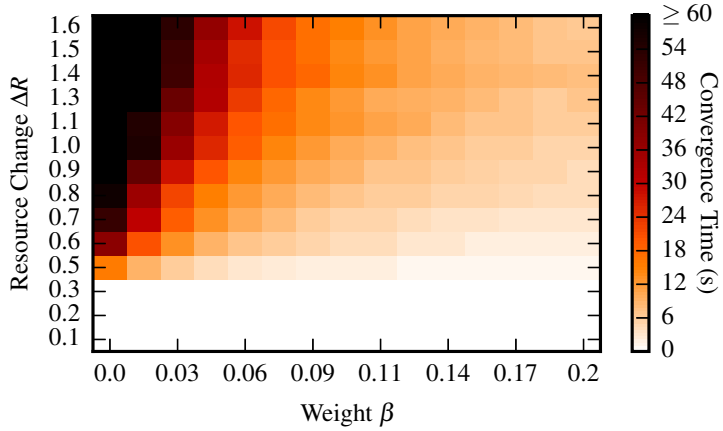


Figure 6.12: Evaluation of convergence time **without** ΔR**-detection** for different ΔR and β. β and ΔR are divided in 15 equally-spaced values. Convergence time increases with decreasing values of β and increasing values of ΔR.

Next, we evaluate the convergence time *without* ΔR-detection for different values of the sample weight parameter β and the amplitude of resource change ΔR. Figure 6.12 depicts the relationship between $\beta = [0, 0.2]$ and $\Delta R = [0.1, 1.6]$ with a step size of 0.015 and 0.1, respectively, and the convergence time. Each value for a combination of ΔR and β is presented as the median of 20 runs. Note that all values of a convergence time $\geq 60s$ are displayed in black. The figure shows that with increasing β, i.e., with a faster decay of the weight of a sample, the model converges to a state of low estimation error faster for most combinations of ΔR and β. Furthermore, the figure illustrates that the model converges slower if the amplitude of the resources change ΔR is larger. For small ΔR, i.e., $\Delta R = \{0.5, 0.6\}$, a low error threshold

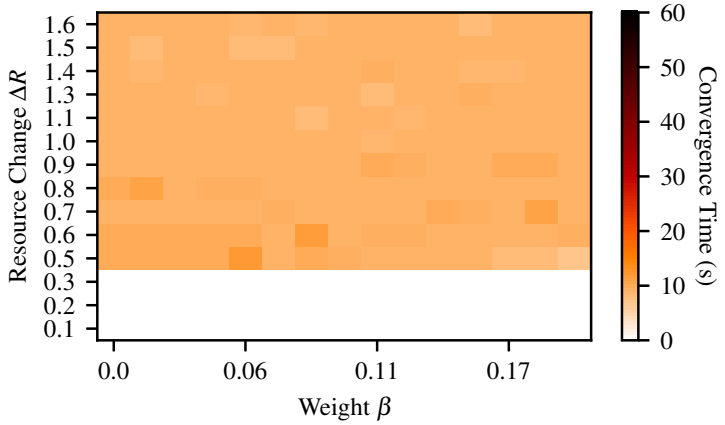


Figure 6.13: Evaluation of convergence time **with ΔR-detection** for different ΔR and β. Standard deviation over all combinations decreases from 21.2 s to 3.9 s. Convergence times are homogeneous with minor variations for almost all parameters. Hence, with ΔR-detection, a wider range of values for β can be selected while maintaining a low convergence time.

is reached in less than 4 s. While a combination of large values of ΔR and small values of β can result in an average convergence time of up to 95 s.

Convergence Time with ΔR-Detection

The findings of Figure 6.12 motivate the need for a sophisticated ΔR-detection to accelerate convergence in cases of large ΔR and low β. With activated ΔR-detection, if a change is detected, the sample buffer is flushed and the model re-trained with the outlier samples and any subsequent samples as described in Section 6.6. Figure 6.13 depicts the relationship between β, ΔR and the convergence time *with* ΔR-detection. Figure 6.14 illustrates the difference between Figure 6.12 and 6.13. Compared to the case without ΔR-detection, we observe a decrease in convergence time for 46 % of the investigated combinations of β and ΔR, mostly in the upper-left triangle of the figure. On average, the decrease is 16.6 s. Furthermore, the standard deviation over all combinations decreases from 21.2 s to 3.9 s From the figure we conclude that with ΔR-detection, a wider range of values for β can be selected while maintaining a low convergence time after ΔR.

However, the figure also illustrates that the convergence time can increase, especially for high values of β and low ΔR found in the lower-right triangle of the figure. On average, the increase is 3 s. This is due to the way outliers are treated in the pipeline. After an outlier is detected, the outlier sample is stored in a separate buffer and not used for the regression. When we observe ten outliers in a row, we signal a detected ΔR and discard the current performance model and use the outlier buffer to learn a new model. But if we do not observe ten outliers in a row, the samples in the outlier buffer are discarded. Hence, there are less samples available to learn from in cases where an *undetected* change in resources happened and this increases

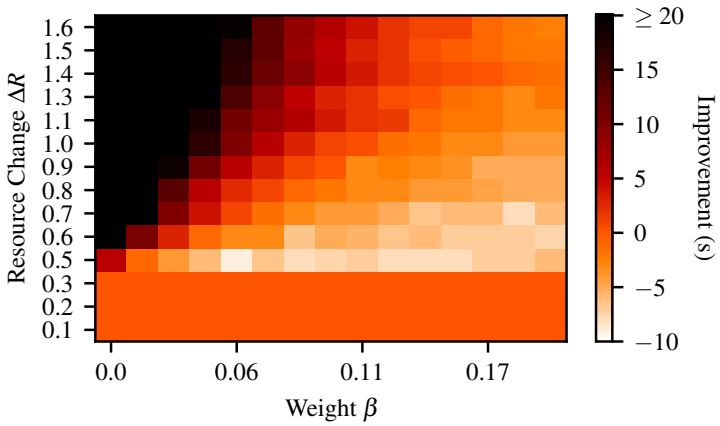


Figure 6.14: Difference between Figure 6.12 and Figure 6.13. Evaluation of the improvement of convergence time **with ΔR-detection** for different ΔR and β. On average, the decrease is convergence time is 16.6 s. A degradation of on average 3 s can be observed for some parameter combinations.

the convergence time. This effect could be mitigated by increasing the sensitivity of the SVM. However, increasing the sensitivity increases the chance of falsely detected resources changes and unnecessary re-training.

6.8.3 Extended Performance Model

The extended performance model enables the estimation of the message budget per message type. In the following section we evaluate the accuracy of the extended performance model. For the evaluation of the extended model we use two random walk processes for the load generator, one random walk process which controls the overall mean message rate (λ) and one which controls the allocation of how many messages of each type are send (λ_{fm} and λ_{ot}). The allocated resources R are constant. Nevertheless, ΔR-detection is turned on to simulate the full proposed pipeline. Each experiment run consists of a warm-up phase of 400 s where no model training was performed. After the warm-up phase, 600 samples are collected at 1 Hz and used in the training of the model. Hence, for the evaluation we take a snapshot of the extended model coefficients at 1000 s into the experiment. Offline benchmarks provide accurate ground truth. β is fixed to 0.0.

Figure 6.15 presents the results. The (blue) markers illustrate offline benchmarks, i.e. the ground truth. For the offline benchmark we selected 11 allocations of λ_{fm} and λ_{ot} with $(\lambda_{fm}, \lambda_{ot}) \in [(a \cdot \lambda, (1-a) \cdot \lambda) | \forall a \in [1, 0.9, .., 0]]$, denoted as Υ. Each allocation is presented as a distinct marker. For example, the upper left dot represents a composition of $\lambda_{fm} \approx 1 \cdot 115000 = 115000$ and $\lambda_{ot} = (1-1) \cdot 115000 = 0$. The confidence intervals of the offline benchmarks are omitted as they would not be visible on the presented scale. The links between the (blue) markers are for better readability and do not represent measurements.

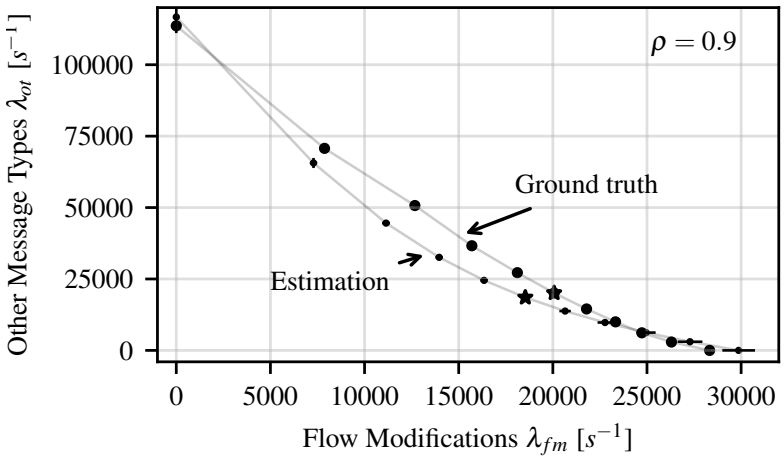


Figure 6.15: Budget estimation accuracy of different message type distributions using the extended model. The blue dots depict the ground truth from offline benchmarks. Red dots show the estimation results of 21 random experiment runs. The two stars highlight the 0.5 allocation, i.e., $\lambda_{fm} = \lambda_{ot}$.

The estimation results of 21 random experiment runs per allocation are shown in red. For the estimation we use the coefficients of the extended model at 1000 s into the experiment run to approximate $\rho_{ext}^{-1}(0.9)$ for the allocations of λ_{fm} and λ_{ot} defined in Υ. The confidence intervals for the estimations are indicated with error bars in the direction of λ_{fm} and λ_{ot}. For most evaluated allocations we observe stable estimation results and therefore most of the confidence intervals are hardly visible. From the figure we conclude that for allocations with a share of ≥ 0.5 in favor of λ_{fm} ($\lambda_{ot} \leq 20.000$ to $\lambda_{fm} \geq 20.000$), the extended model can accurately estimate the message budget. For shares ≥ 0.5 in favor of λ_{ot}, it becomes less accurate and varies between underestimating, e.g., at $\lambda_{ot} = 40.000$ to $\lambda_{fm} = 15.000$, and overestimating the utilization, e.g., where λ_{fm} is close to 0. Confidence intervals increase if either λ_{fm} or λ_{ot} dominates λ, but in general are small.

6.9 Summary and Deployment Guidelines

The evaluation results show that the choice of model, pipeline configuration and parameters can be optimized depending on the use case. Next, we summarize the results and discuss how to achieve optimal budget estimation results for different use cases.

6.9.1 Summary

In Section 6.8.1 we discussed the general budget estimation error with constant resource assignment. The results show that the sample weighting parameter β has no significant effect on the estimation error in the evaluated parameter range and that the error ranges between

Table 6.3: Deployment Guidelines

Use Cases ↓	Static R, Stable distribution	Dynamic R and/or unstable distribution
Overall budget sufficient	Simple model, $\beta \geq 0.1$	Normal model with SVM $\beta \leq 0.1$
Budget estimation per message type required	Simple & ext. model, accept cost per type inaccuracy	If unstable distribution, extended model. If stable distribution, simple & ext. model.

2.6 % and 9.5 %. In Section 6.8.2 we evaluated the convergence time after a change in allocated resource with and without ΔR-detection. For this we measure the time it takes for the pipeline to accurately estimate the current budget after ΔR. Without ΔR-detection, for minor ΔR and a large value of β, we observe a fast convergence, whereas large changes and a low values of β show low convergence time. With ΔR-detection, the results exhibit a homogeneous convergence time in the evaluated parameter range. On the one side, we conclude that the ΔR-detection speeds up the convergence time considerable after a large change in resources allocation and low β. On the other side, the ΔR-detection slows down the convergence time for minor changes. The evaluation of the extended model in Section 6.8.3 shows that accurate estimation of the resource consumption per message type is possible. In particular, the results show that the asynchronously collected message counters and resource usage samples are sufficient for the regression to learn the model coefficients in scenarios where the message type distribution is not constant.

6.9.2 Deployment Guidelines

Next, we discuss the following questions: when do I choose the extended model and which values do I use for the parameters? Based on the evaluation, we identify three criteria which dictate the choice of model and parameters. First, is it enough to know the overall budget or do I need the budget per message type? Second, do I expect the available resources R to be constant or are they likely to change frequently? Third, is the message type distribution of the incoming messages mostly stable or is it likely to be unstable? Table 6.3 gives guidelines based on these three criteria.

Overall budget sufficient, R static, stable type distribution

If the overall budget is sufficient and R is mostly static with a stable message type distribution, then the simple model with $\beta \geq 0.1$ is sufficient. However, note that a large value of β increases the influence of measurement noise on the estimation accuracy.

Overall budget sufficient, R dynamic, unstable type distribution

If the overall budget is sufficient, but R is likely to change and/or the distribution of message type is unstable, use the normal model with SVM, but select $\beta \leq 0.1$.

Budget per message type, R dynamic, stable type distribution

If you require budget estimation per message type and R is either static or dynamic and you observe a stable message type distribution, then deploying the simple and extended model in parallel is the best choice. However, the stable message type distribution will not contain enough information to accurately learn the cost per message type.

Budget per message type, R dynamic, unstable type distribution

If you require budget estimation per message type and R is either static or dynamic and the message type distribution is changing over time, then the extended model gives accurate estimation results per message type. However, note that in our experiments the extended models in general converged slower than the simple model. Furthermore, note that for best estimation results the trade-off parameters between linear and negative exponential behavior might require adjustments to the systems at hand. For example, during our experiments we noticed that in environments with power saving settings turned on, the relationship between λ and ρ tends to be less linear and more negative exponential.

In general, the estimation accuracy also depends on the range of samples seen by the learning process. The estimation becomes better if many samples close to 90 % utilization exist. If the samples are dominated mostly by low utilization samples, e.g., $\rho \leq 10\,\%$, the estimation will be worse. This can present a challenge where very large message budgets are allocated to control applications at the same time. However, we expect that in a realistic deployment the budget allocated to one control application is far less than the overall budget and as our previous study shows [11], samples with an utilization of about $20\,\% - 25\,\%$ are already well suited for estimation using the negative exponential model.

In summary, in this chapter we propose and evaluate an online machine learning pipeline for the capacity estimation of NSAL instances in dynamic cloud environments. The evaluation shows that the learned performance model provides accurate estimations of the message rate budget at run-time. Furthermore, a reduction or increase of available resources assigned to the abstraction layer is detected by the pipeline and the estimations are adapted accordingly. The proposed pipeline is an important step towards autonomous scaling and load-balancing of NSAL instances. Future work in this area should investigate the convergence time of the extended model and evaluate the trade-off between SVM sensitivity and convergence time in more detail.

Chapter 7

Application-Aware Resource Allocation through the NSAL

The presented Network Services Abstraction Layer (NSAL) enables the device and vendor-neutral control and Quality of Service (QoS)-configuration of forwarding devices. In the preceding chapters we first discussed how to design an NSAL for enterprise networks. Afterwards, we discussed the control-plane performance of such an NSAL from a theoretical and deployment perspective. However, there is an important aspect missing in the NSAL design so far: The applications. At the end of every service transported over an enterprise network stands an application and a human user behind the application. From web browsing to Voice-over-IP (VoIP) and video streaming applications, a user is involved and expects a certain Quality of Experience (QoE) with the service. To account for the application and user behind the transported data, the NSAL is extended in this chapter with application-awareness.

Application-awareness it closely related to the problem of resource allocation. Resource allocation is the question of how to distribute the available network resources to satisfy every application. Hence, the NSAL is extended with both, application-awareness and resource allocation. Achieving efficient and application-aware multi-application resource allocation in enterprise networks is challenging when applications can send packets into the network at will. But enterprise networks are not bound to the same net neutrality laws which govern most parts of the public Internet and access to an enterprise network is not limited to approved devices. The network operator is in full control of the applications deployed in the network and on the end-hosts. This is driven by security concerns (malware, leakage of sensitive documents/data) and the need for performance guarantees for mission-critical applications. This means that end-hosts are restricted to a small set of applications, potentially depending on the role of the employee, and that the communication of each application is monitored. HTTP(S) traffic is passed through a proxy to perform Deep Packet Inspection (DPI) to identify

sensitive documents being uploaded on an external website or malware being accidentally downloaded.

In Chapter 3.3 we evaluated policing on the forwarding devices as a means to restrict applications. The results show that traffic policing or shaping on forwarding devices in the network leads to queuing and packet loss, which introduces delay and decreases transmission efficiency. Furthermore, it destabilizes the congestion control algorithms which results in under-utilization of the allocated throughput. Therefore we propose a methodology for multi-application resource allocation in enterprise networks based on delay-constrained central routing configured through the NSAL and fine-grained per-application pacing at the end-hosts. Pacing refers to the method of restricting the amount of data an application is allowed to send into the network by implementing local back-pressure to the application sockets and introducing artificial delay between packets. Moving application pacing from the forwarding devices to the end hosts, e.g., to user PCs, servers, smartphones and tablets, is scalable, increases transmission efficiency, reduces the required complexity of forwarding devices and allows cost-efficient high link utilizations [105, 137].

Figure 7.1 gives an overview over the extended NSAL. The graph model is extended with applications (Ⓐ), pacing components (🄿) and virtual switches (Ⓥ). The NSAL itself is extended by a module for application-awareness (①) which provides 🄿, Ⓐ and Ⓥ to northbound control applications. Based on the extensions, it is possible to implement application-aware control applications (②) on top of the extended NSAL. The control applications are able to query the active applications in the network (③). The novel control application can then decide on pacing rates for each application and delay-constrained routing policies. The decisions are based on the provided application list and available resources provided by the NSAL abstraction, e.g., available bandwidth and network topology.

The chapter is structured as follows. First, the problems and challenges are discussed in detail (Section 7.1). Afterwards, Section 7.2 takes a closer look at the related work in this area of research. Section 7.3 subsequently introduces the extensions required to the NSAL's graph-based abstraction model. Then, throughput- and delay-dependent utility functions for five application classes are defined (Section 7.4). Compared to other works, the utility functions are based on actual subjective studies and measurements of the applications. Afterwards, the utility bandwidth- and delay-aware allocation problem is formulated as a 2-step Mixed-Integer Linear Program (MILP). The first step maximizes the minimum utility in the network (*max-min-fairness*), while the second step maximizes the sum of all utilities for a constrained minimum utility (Section 7.5). Application mixes with over 100 parallel application of 5 common use cases are evaluated in a proof-of-concept set-up (Section 7.6). By evaluating the measurements, it is shown how the NSAL with pacing can improve QoS metrics such as delay

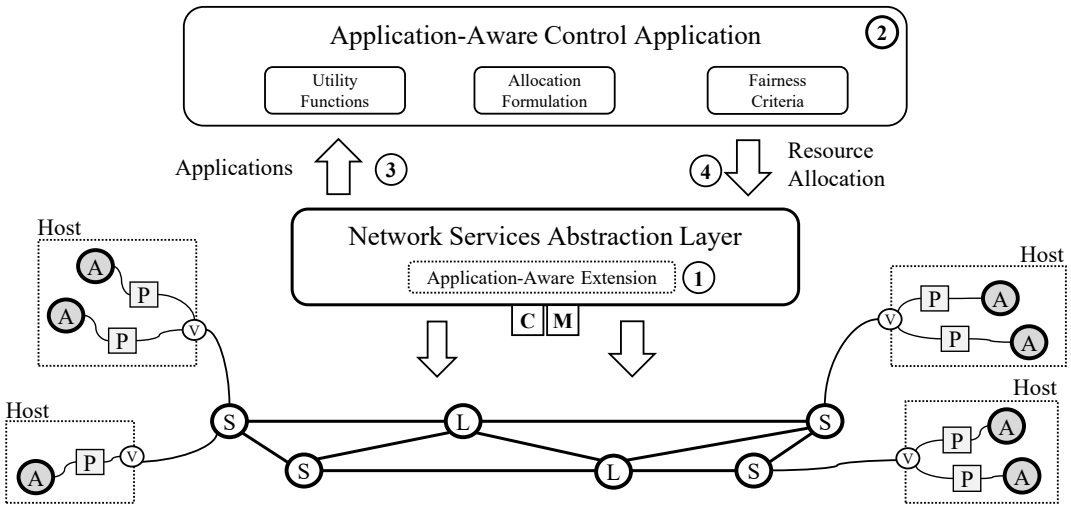


Figure 7.1: Extension of the NSAL with application-awareness. The topology abstraction is extended with applications ((A)), pacers ([P]) and virtual switches ((V)) on the hosts. An extension to the abstraction layer ((1)) allows discovery of applications ((3)) and centrally-controlled pacing ((4)) at end-hosts for novel application-aware control applications ((2)).

and packet loss and significantly increase inter-application fairness compared to a best-effort scenarios (Section 7.7).

The content of this chapter describes the results presented in the following publications. The majority of the results are based on [3]. The paper [3] discusses dependable application-aware resource allocation in managed networks through end-host pacing. In [4, 2] results regarding the user-experience of video streaming are presented. The results are a part of the selection and evaluation process of the video streaming use cases and Key Performance Indicators (KPIs) in this chapter.

[3] C. Sieber, S. Schwarzmann, A. Blenk, T. Zinner, and W. Kellerer. "Scalable Application- and User-aware Resource Allocation in Enterprise Networks Using End-host Pacing." In: Under minor revision for *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)* 34 pages (2018). arXiv: 1811.02367.

[2] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia. "Identifying QoE Optimal Adaptation of HTTP Adaptive Streaming Based on Subjective Studies." In: *ELSEVIER Computer Networks* 81.23 pages (2015). DOI: 10.1016/j.comnet.2015.02.015.

[4] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. "Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming." In: *Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*. 6 pages. Singapore, Sept. 2014. DOI: 10.1109/QoMEX.2014.6982305.

7.1 Challenges and Problem Definition

This chapter investigates the following three main challenges for an application- and user-aware NSAL design and provides the definition of the problems. The challenges can be summarized to *defining*, *determining* and *allocating* resource shares:

- **Challenge C7.1**: How to *define* resource shares in terms of QoE considering the possible variety of application classes and their demands?
- **Challenge C7.2**: How to *determine* the shares under resource constraints?
- **Challenge C7.3**: How to *allocate* each application a share of the network resources through the abstractions offered by the NSAL?

7.1.1 C7.1: Defining User-Level Resource Shares

Ultimately, a user of an application does not care about what share of the resources is allocated to her/him as long as her/his user experience, or QoE, with the application is positive. Challenge C7.1 is how to define a resource allocation based on user experience. This is challenging as the experience of a user with an application can only by determined through subjective studies. For example by implementing options for live feedback into the used applications or by conducting laboratory or crowd-sourcing studies. The experience of a user with an application is commonly measured on the Mean Opinion Score (MOS) scale [167], which describes the experience of a user as 1 (*bad*) to 5 (*excellent*). From the subjective studies, a model has to be derived which can describe the experience of the user in terms of tangible KPIs. The KPIs can be technical metrics such as response time or web page load time, but the model may also include parameters of the environment or the user itself, such as screen size or user expectation.

We address this challenge by defining the user experience as a per-application utility function of throughput and delay demand. For this we rely on user experience models from the literature, selected KPIs and automated measurements. We measure the KPIs of each investigated application under different emulated throughput and delay restrictions. Afterwards, we apply the models from the literature on the measured KPIs and derive the per-application utility function.

7.1.2 C7.2: Determining Resource Shares under Resource Constraints

Based on the utility functions defined in challenge C7.1 and the available network resources, a feasible and fair allocation of network shares has to be determined. A formulation of the

allocation problem must consider the 2-dimensional utility functions, the network topology, the available bandwidth on each link, the queuing, processing and propagation delay at each hop and a fairness criteria. The utility functions are defined in C7.1, the network topology, available bandwidth and processing and propagation delay can be provided by the NSAL topology abstraction. Multiple fairness criteria exist in the literature, e.g., proportional fairness, weighted proportional fairness, balanced fairness or max-min fairness [32]. Which fairness criteria to apply depends on the specific deployment scenario. The queuing delay at each hop depends on the inter-arrival times of the arriving packets. However, for practical purposes the delay can be modeled with the rates of the traversing flows and the link utilization [33]. The allocation problem formulation has to balance the target link utilization in order to full-fill the delay demands as given by the utility functions. While the min-max utility proportional fair bandwidth allocation problem is well studied in literature, the problem combination of bandwidth allocation and delay-aware routing for arbitrary utility functions is less studied.

We address challenge C7.2 by formulating the allocation problem as a MILP and solve the problem for the max-min fairness criteria. By jointly optimizing the utility and network resources usage, a share in terms of max-min utility can be determined given a set of applications, utility functions and constrained network resources.

7.1.3 C7.3: Allocating Resource Shares in the Network

The main problems with implementing QoS mechanisms for resource allocation in the network are as follows. a) Buffers in forwarding devices are expensive and there is only a limited number of queues to configure per egress interface, typically about 8 [189]. This is insufficient to implement a sophisticated strategy to distinguish hundreds of active applications of multiple classes in a network. Furthermore, congestion can then happen inside an application class which then degrades the performance of all applications of the class. b) Policing interacts badly with transport-level congestion avoidance algorithms resulting in lost packets. Lost packets cause retransmissions and decrease transmission efficiency [66]. c) Heterogeneous enterprise networks with diverse forwarding devices from different vendors are complex and error-prune to manage. Furthermore there are no common QoS abstractions across switching hardware vendors. Hence deploying a single QoS strategy across devices might not be possible, especially if not all devices support the required features. d) Encryption or header field ambiguity can prevent the correct identification of application classes in the network.

For challenge C7.3 we address the resource allocation problem by implementing centrally-controlled pacing by the NSAL of individual applications at the end-hosts, combined with delay-aware routing in the network. Packet pacing at the end-hosts ensures that a stream of

Table 7.1: Overview of related works targeting multi-application QoE-awareness.

	C7.1) Define	C7.2) Determine	C7.3) Allocate
[73]	Mapping KPIs to QoE	Not specified	Generic control concepts
[149]	Mapping network QoS to QoE	Particle swarm optimization (PSO) based algorithm	Applying the proposed algorithm to resource block allocation technique in LTE
[136]	Mapping network QoS to QoE	Game theoretic approach	Radio resource management applying proposed game theoretic approach
[113]	Mapping network QoS and KPIs to QoE	Optimization based on multi-choice knapsack problem (MCKP)	Carrier scheduling applying proposed optimization algorithm
[62]	Mapping of network bandwidth to QoE	Solving multi-objective optimization problem	Joint subcarrier and power allocation scheme
[64, 65]	Application feedback instead of utility functions	Not specified	Admission Control, bandwidth guarantees
[138]	Hypothetical utility functions mapping network QoS to QoE	Proposed algorithm optimizing bandwidth allocation	WFQ scheduling with QoE-optimized weights
[71]	Mapping screen resolution and bitrate to Structural Similarity Index (SSIM)	Branch and bound algorithm to find optimal set of video bitrates	Video bitrate guidance for heterogeneous clients
[105]	Mapping bandwidth to an arbitrary fair share	Novel Multi-Path Fair Allocation (MFAA) algorithm	Enforced via pacing at the hosts

packets conforms to a specified data-rate by adding artificial delays during the sending process in-between consecutive packets. Pacing prevents packet loss by smoothing out packet bursts and allows for shallow buffers in intermediate forwarding nodes. Shallow buffers reduce queuing delay and avoid expensive switch buffer space. Applications can reliably determine their available goodput and it is unnecessary to probe the throughput by loss-based congestion control mechanisms. Furthermore, pacing at the end-host allows for implementation of effective backpressure to the applications producing the data, reducing the amount of buffered data in the network stack. Pacing at the end-hosts can scale to thousands of traffic classes [137], congestion in the network can be avoided by central management of the available resources [105] and application flows can be identified at the source. Recent works show that bandwidth allocation to applications can be implemented holistically at global scale, enabling high percentages of link utilization [105]. Sender congestion control and QoS in the network are downgraded both to failsafe solutions and supportive roles in the overall QoS strategy, e.g., in cases the central control fails or embedded devices can not be modified.

7.2 Background

Several efforts have been made towards QoE-awareness in multi-application scenarios. Some relevant approaches are summarized in Table 7.1. The table columns represents the three

challenges introduced beforehand: *define*, *determine*, and *allocate*. Many related works on multi-application QoE-aware network design are associated to the mobile domain [62, 73, 113, 136, 149].

Several KPIs are proposed to be monitored at network elements in the architecture of [73], including packet loss rate, throughput, and Round Trip Time (RTT). At the clients, network-related parameters, e.g., delay, and application-based metrics including web page download time or video buffer and bitrate can be measured. The collected KPIs metrics are used to estimate the per-application QoE using models from literature. One of the presented use-cases considers an optimization based on estimated QoE values. To do so, the authors list a variety of parameters that can be configured along the protocol stack in order to control QoE. The work does not provide a specific algorithm for determining the required resources, nor does it propose a designated method for allocating them. Instead, the authors outline several possible control actions like bandwidth limiting or QoE-aware capacity planning. As the utility functions applied only rely on KPIs, the QoE can only be controlled in a qualitative manner, meaning enhancing and degrading the QoE, but not controlling it so to achieve e.g. a specific MOS value.

[149] proposes a novel algorithm for resource block allocation in Long Term Evolution (LTE) systems to maximize QoE whilst preserving fairness among users. The authors also use existing models to estimate user QoE, but adapt the models so to express the MOS solely from network-parameters like delay or packet error probability. Based on these models, the authors present a resource block allocation algorithm that is based on Particle Swarm Optimization (PSO). Another QoE-aware resource scheduling algorithm for mobile networks is based on a game-theoretic approach [136]. The QoE is estimated for various applications using models from literature that map network parameters to MOS. The users' data flows cooperate with each other in a proactive manner and jointly optimize the QoE in a game theoretic based manner. Instead of using the conventional throughput maximizing algorithm in radio resource management of orthogonal frequency-division multiple access (OFDMA), the authors propose to implement their scheduling algorithm which aims on maximizing the fairness among heterogeneous users.

The approach described in [113] targets QoE-awareness in mobile LTE-Advanced networks. In the QoE modeling step, both network QoS and application KPIs are used for estimating the user perceived quality for different types of application. The QoE estimates and available bandwidth are inputs to the resource scheduling algorithm, which solves a multi-choice knapsack problem (MCKP) that maximizes the sum of all users' MOS values. The component carriers are dynamically scheduled according to the network traffic load by this QoE-aware scheme.

A further approach towards QoE-driven resource allocation in wireless networks is [62]. The authors apply utility functions which express MOS for various applications as functions of different network QoS parameters. Thereby, they assume a packet error probability of 0, a packet loss rate of 0, and fix frame rate in the case of video streaming applications. Using these simplified utility functions, the authors propose a solution to a multi-objective optimization problem which aims at maximizing MOS. As network resource control mechanisms, the authors apply an efficient allocation of subcarriers among the active users.

The concept of Participatory Networking is proposed in [64, 65]. It describes an Application Programming Interface (API) that can be used by applications, end-hosts, and devices to interact with the network. A centralized controller is authorized to delegate read and write access to the network participants. Using the write access, applications, users or end-hosts can reconfigure the network according to their needs and can provide knowledge to the network, e.g., their future traffic demands. Hence, no utility functions that map KPIs or network QoS to QoE are needed, as the application instances directly communicate their requirements to this controller.

[138] applies hypothetical piecewise linear functions that map bandwidth to QoE and propose a new scheduler for fair and efficiently bandwidth allocation in shared networks. Using these utility functions, they optimize the bandwidth per flow so to have a fair utility over all active applications. According to the bandwidth shares, the weighted fair scheduler allocates respective weights to the flows. Simulation results show that the minimum utility can be increased significantly, while maintaining the same average utility in most of the cases, compared to a conventional max-min-fairness approach.

[71] presents an Software-Defined Networking (SDN)-based framework to support a fair video QoE allocation for all clients within a shared network domain. The utility function maps a client's device resolution and bitrate to SSIM [158]. Considering the current network capacity, a controller decides about the bitrate for each video client, so to provide a similar quality to each of them. The bitrates are communicated to the streaming clients, which in turn request the respective quality layer from the video content server.

BwE[105] introduces a global hierarchical top-down bandwidth allocation schema used in Google's internal network for distributed computing tasks. Bandwidth allocation is done via a function mapping bandwidth to a "relative priority on an arbitrary, dimensionless measure of available fair share capacity." [105] The BwE reference is important as it shows that global and large-scale bandwidth allocation is indeed possible in production environments. But how to derive an allocation for end-user applications and how they benefit from it, is not discussed in BwE. In contrast, this chapter at hand focuses on end-user applications and the interplay with, and possibilities for, network control to guarantee a specific user experience to the end users through the NSAL.

The presented strategies are all steps towards QoE-awareness in multi-application systems. Some of the works rely on state of the art control mechanisms, but propose novel resource scheduling or allocation techniques. However, the applied utility functions often depend on features which cannot be influenced in a direct manner. As a result, those approaches allow for a qualitative, less targeted QoE control compared to the proposed solution. For example, a low video quality implies a low MOS value. Providing more bandwidth will enhance the playback quality and increase MOS, but it is not possible to quantify the impact of providing a certain amount of bandwidth on the MOS scale. This chapter presents an approach that allows to quantitatively map the network QoS parameters bandwidth and delay to MOS (Challenge C7.1). Furthermore, we propose to apply pacing, which allows us to control both, the bandwidth allocated to a flow and the end-to-end delay (Challenge 7.3). Having the utility functions only relying on controllable parameters allows for a targeted, fine granular QoE optimization (Challenge 7.2). In summary, the proposed extensions to the NSAL abstraction in combination with the novel application control opportunities northbound of the NSAL enable true multi-application QoE control in the network.

7.3 Extensions to the NSAL Abstraction

There are three extensions to the NSAL graph-model required to implement the proposed solutions to the challenges C7.1 to C7.3. Figure 7.2 illustrates the additional components which extend the control of the NSAL beyond the network to the end-hosts. The term applications refers to client- and server-applications alike. Both types, client and server applications, contribute to network congestion. Thus client applications, e.g., web browsers, and server applications, e.g., Hypertext Transfer Protocol (HTTP) servers, have to be known to the NSAL. Each application registers at startup with the NSAL through a local agent at the end-host. The technical implementation of this is out of scope of this work.

Next, we describe the additional components and their interfaces. First there is the *application* component (Ⓐ). The application has as a property which class the application belongs to, e.g., a video streaming client. Furthermore, the application component provides the *intent* of the communication, e.g., which video is watched. Intents are defined in detail in Section 7.4. The application is connected to a *pacing* component (🅿). The pacing component allows to configure an egress data-rate and a maximum queue size. When the maximum queue size is reached, the application is blocked from sending more data (*backpressure*), but no packets are dropped. The pacer is connected to a *software switch* component (Ⓥ). The software switch forwards the data from the applications to the physical interface of the host. Furthermore, the virtual switch provides in- and egress counters of each application to

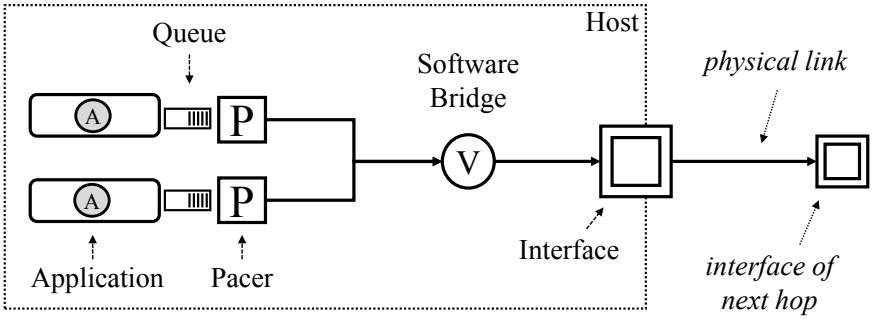


Figure 7.2: Extensions to the NSAL graph-model for application-awareness. The NSAL graph-model is extended with the three components application (Ⓐ), pacer (🄿) and virtual switch (Ⓥ). The host's interfaces were already part of the design of the NSAL in Chapter 4. Component Ⓐ allows identification of applications. Component 🄿 allows to set application egress rate. Component Ⓥ provides in- and egress packet and Byte counters for each application.

the NSAL. That way the control applications can monitor the sending and receiving rate of each application. At the end-hosts, the association between flows and applications is straight forward. In the network this is often not possible due to encryption or header field ambiguity.

7.4 Utility Function Definition

Comparing the performance of different applications with conceptual different KPIs requires mapping functions to a common scale. We denote the scale as *user-aware utility scale* and we define it with a dimensionless quantity in the range of [1, 5]. The utility functions then describe the relationship between the amount of resources allocated to an application and the resulting experience of the user with the application. In the following section, we define the utility functions for selected classes of applications and intents. First, we present the considered application classes, intents, and KPIs of the deployed implementations. Second, we discuss the selected user experience models from the literature. Third, we define the utility functions based on measurements and the user experience models.

We consider five application classes: Web browsing, file download, video streaming, remote terminal work, and VoIP (Table 7.2). *Web browsing* covers a wide range of use cases, as modern web standards facilitate the move from proprietary and platform-dependent software to responsive web applications running in the browser. *File download* is the batch-transfer of data the user is waiting for, such as an email attachment. Use cases for adaptive *video streaming* in the enterprise range from announcements to training videos, such as on-boarding lectures for new employees. In particular major announcements are taxing for the infrastructure when viewed by a large fraction of the staff in a short time-frame. *Remote terminal work* by secure shell access allows administrators to access the terminals of servers, hosts, and switches from

Table 7.2: Applications, Intents and Key Performance Indicators

Class	Application	Intent(s)	Shorthand(s)	Evaluated KPI(s)
Web Browsing	Firefox, selenium [184]	*science_lab*	*WEB*	Page Load Time
File Download	Python *requests*	*emailattach*	*DL*	Download Time
Video Streaming	TAPAS [56]	*bbb*, *bbb_live*	*VoD*, *Live*	Quality(+ Switches, Stalls)
Remote Terminal	SSHv2, paramiko [182]	*sshadmin*	*SSH*	Response Time
Voice-over-IP	D-ITG [34]	*g729.1*	*VoIP*	Delay, Loss, (+ Jitter)

anywhere. The application class *VoIP* includes office phones, conferencing by software or in the browser, and VoIP applications on smartphones. We denote the combination between an application class and intent as application *type* and use the types *WEB*, *DL*, *VoIP*, *Live*, *SSH* and *VoIP* as shorthands for the investigated combinations of application classes and intents.

7.4.1 Applications, Intents and KPIs

Next we discuss the implementations, KPIs, and intents per application class in detail. KPIs in parentheses in Table 7.2 are not inputs for the user experience models, but are part of the evaluation in this paper.

7.4.1.1 Remote Terminal Work

For remote terminal work we define the intent of an administrator typing commands over a Secure Shell (SSH) connection. An automated SSH client enters commands and measures the duration until the output of the command appears in the terminal. Only commands which require minimal processing on the server-side, e.g., *uptime* and *date*, are entered. The SSH connection is established before the start of the experiment. *OpenSSH 7.2* is used as server implementation on *Ubuntu 16.04.4 LTS* systems. Client-side automation is implemented using *paramiko* [182].

7.4.1.2 File Downloads / Web Browsing

File download is the batch transfer of a chunk of data over one Transmission Control Protocol (TCP) connection. As intent we define *emailattach*, a file with random content and a size of 10 MB, which is placed on a HTTP server for download. In an enterprise environment this intent could represent the maximum size of email attachments. The download is implemented using a short Python script and the *requests* library. As KPI, the script measures the duration from when the GET request is sent, up to the last received Byte.

Web browsing is implemented using the open-source browser Firefox in version 58.0.2, automated with *selenium* [184]. The settings are left to the default state and the cache is cleared after every page view. The number of parallel connections is limited to six per server

and HTTP pipelining is not supported anymore by recent Firefox versions. The connections are configured to be persistent between requests. The browser interface is disabled (headless mode) and no page rendering is performed in the experiments to minimize the influence of system load and deployed testbed hardware.

This is a scenario where a limited number of browser-based business applications are used frequently and/or all web browsing sessions are tunneled through an enterprise proxy. With proxies, connections can be persistent even when requesting content from different domains. General web browsing, where multiple domains are involved without proxy, is not represented well by assuming persistent connections. This is due to the fact that connection establishment can significantly influence the page load time for longer transport delays. We define the KPI for one web browsing request as the duration from the initial GET request to the time all embedded resources are received (*page load time*). For web browsing we define the intent *science_lab*. The science_lab template [187] is an example web-site with 22 objects with a total size of about 1.3 MB.

7.4.1.3 Adaptive Video Streaming

HTTP adaptive video streaming is implemented using the TAPAS [56] Dynamic Adaptive Streaming over HTTP (DASH) player. The *conventional* [112] bit-rate adaptation strategy is selected. We consider one video view as one request and select the average quality level of all downloaded segments as KPI. We define the intent *bbb* for on-demand video streaming. For this intent, we encode the open-source movie Big Buck Bunny in six quality levels with average bit-rates of 486 Kbps, 944 Kbps, 1389 Kbps, 1847 Kbps, 2291 Kbps, and 2750 Kbps. Only the first 60 s of the movie are selected and segmented into 15 chunks of 4 s each. The playback buffer is configured with a maximum size of 60 s

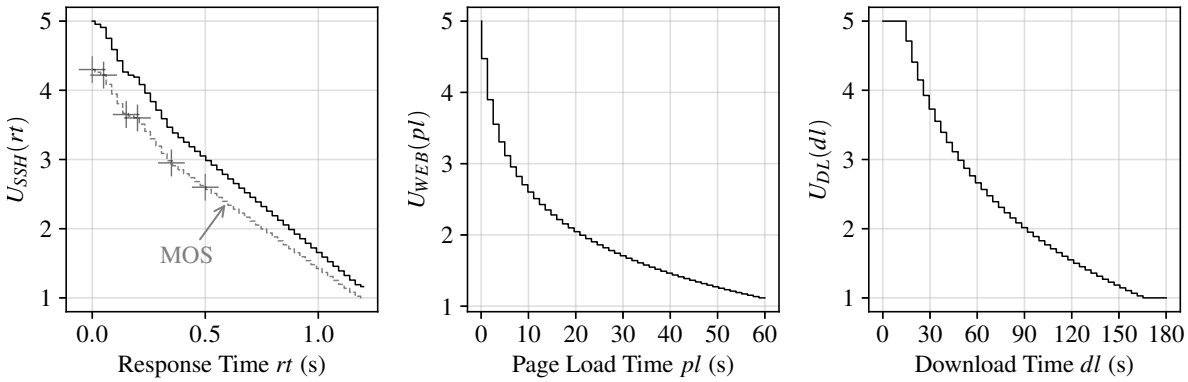
Additionally, we define the live-streaming intent *bbb_live* where the chunk size is reduced to 1 s and the buffer is limited to 10 s. Due to encoding overhead for the shorter chunk duration, the bit-rates increase to 572 Kbps, 1103 Kbps, 1625 Kbps, 2145 Kbps, 2660 Kbps, and 3172 Kbps.

7.4.1.4 Voice-over-IP

We emulate VoIP traffic using the Distributed Internet Traffic Generator (*D-ITG*) by Botta et al. [34]. *D-ITG* reproduces the inter departure-times and packet sizes of VoIP traffic and measures the KPIs jitter, packet loss, and delay of the resulting User Datagram Protocol (UDP) packet stream. We define the intent *G.729.1* for VoIP and configure *D-ITG* to emulate Real-Time Transport Protocol (RTP) VoIP calls with the audio codec G.729.1. In this configuration, a

Table 7.3: Subjective Models

Class	Model KPI(s)	Subjective Model
Web Browsing	Page Load Time (*pl*)	Egger et al. [58]
File Download	Download Time (*dl*)	Egger et al. [58]
Video Streaming	Average Quality (*ql*)	*custom*
Remote Terminal	Response Time (*rt*)	Casas et al. [43]
Voice-over-IP	Delay (*delay*), Loss (*loss*)	Sun et al. [146]



(a) Terminal Work ($M^{(SSH)}$) (b) Web Browsing ($M^{(WEB)}$) (c) File Download (10M) ($M^{(DL)}$)

Figure 7.3: Utility from application KPIs (M: KPIs $\mapsto$ Utility) derived from subjective study results scaled to range $[1, 5]$. $M^{(SSH)}$ is derived from subjective study [43, Fig. 5 (a)] by Casas et al. Plus signs indicate the MOS data points as collected by the authors in the study. Web and file download utility models are derived from subjective user studies in [58] by Egger et al.

constant bit-rate stream with 50 packets per second is generated with a packet size of about 20 Bytes, resulting in data-rate of ≈ 8 Kbps.

7.4.2 Utility from KPIs

We define the current utility value of an application as an estimation of the instantaneous satisfaction of a user with the interaction with the application. The relationship between KPIs and user experience has to be determined through subjective studies, either directly by conducting dedicated laboratory, field, or crowd-sourcing studies, or indirectly by measuring user-relevant success metrics such as task completion times. We denote this relationship as M: KPI $\mapsto$ Utility. In case there is a suitable QoE MOS [167] model available for the application, we take a scaled version of the MOS model as the utility function (Table 7.3). The MOS scale describes the experience of a user with the application on a scale of one to five where the scale is labeled with {Bad, Poor, Fair, Good, Excellent}. However, the range of some user experience models does not reach up to 5.0 (Excellent). In those cases, we define the utility function by scaling up the experience model to $[1, 5]$. If no model is available, we define the utility based on hand-picked application KPIs.

QoE is an active area of research and holistic models do not exist yet for most applications. There could be alternatives or more complex models available for the selected user experience models. Furthermore, custom enterprise applications might require custom user experience studies. In any case, the presented system design and findings of this chapter are independent of the concrete deployed user experience models. Therefore, the selected models in this work should be seen as rough approximations of the true underlying user experience.

7.4.2.1 Remote Terminal Work

We piece-wise interpolate a utility model for remote typing from the results presented in [43, Fig. 5(a)]. There, Casas et al. study the QoE of remote desktop services for different use cases. For the investigated *typing* use case, the test subjects where asked to type a short text on a text processor in a remote desktop session. The higher the delay in the network, the longer the user has to wait until his actions, e.g., typing a character or deleting character, appear on the screen. The delay until the actions result in visual feedback is denoted as response time and we choose it as the KPI for remote terminal work. Figure 7.3a illustrates the piece-wise interpolated model based on the presented opinion scores in [43]. The authors only investigated response time values up to 0.5 s. We linearly extrapolate the results up to 1.2 s where the utility reaches 1. We define M as $M^{(\mathrm{SSH})}(t) := MOS^{(\mathrm{SSH})}(t) - 1) \cdot \frac{4}{3.3} + 1$ to project the MOS values to a utility range of $[1, 5]$.

7.4.2.2 Web Browsing / File Downloads

Egger et al. [58] propose models for the user experience of web browsing and file downloads based on subjective user studies. The web browsing model uses the page load time (pl) as KPI. For the file download, the download time (dl) of a 10 MB file is used as KPI. The MOS value for web browsing is proposed as $MOS_{WEB}(pl) := -0.88 \cdot ln(pl) + 4.72$. For the file download, $MOS_{DL}(dl) := -1.68 \cdot ln(dl) + 9.61$.

Figure 7.3b illustrates the web browsing model. The figure highlights the severe impact of the page load time on the user experience in web browsing. After only 2.2 s waiting time, the MOS is already down from 5 (*Excellent*) to 4 (*Good*). With additional 4.6 s waiting time, the MOS decreases to 3 (*Fair*). After a total waiting time of 20 s, the score ranges between *Poor* and *Bad*. For web downloads (Figure 7.3c), the users are more willing to accept longer waiting times. For example it takes a waiting time of 28 s for the opinion score to decrease to 4. We use the $MOS^{(\mathrm{DL})}$ model as proposed by the authors as M with $M^{(\mathrm{DL})}(t) := MOS^{(\mathrm{DL})}(t)$. $M^{(\mathrm{WEB})}$ we define as $M^{(\mathrm{WEB})}(t) := (MOS^{(\mathrm{WEB})}(t) - 1) \cdot \frac{4}{3.6} + 1$.

7.4.2.3 Adaptive Video Streaming

The user experience during an adaptive video streaming session depends on factors such as average presented quality, number and amplitude of quality switches, frequency and duration of stalling events, device's screen size, viewing environment, user expectation, encoding, adaptation strategy, and content type [4]. To the best of our knowledge there is no holistic model for the user experience of adaptive streaming available at the moment. One option for enterprises is to create custom models, for example for onboarding videos for new employees.

Studies show the average quality as a dominant influence factor [2] for the user QoE. For the evaluate we therefore assign a utility value to a streaming application based on the observed average quality $q^{(\text{avg})}$ and the maximum and minimum quality level, $q^{(\text{max})}$ and $q^{(\text{min})}$. The utility value is then determined by $M^{(\text{HAS})}(q^{(\text{avg})}) := \frac{q^{(\text{avg})}-q^{(\text{min})}}{q^{(\text{max})}-q^{(\text{min})}} \cdot 4 + 1$.

7.4.2.4 Voice-over-IP

Sun et al. [146] propose a model for the MOS of VoIP depending on the used audio codec and a user's interactivity, i.e., whether the user is only listening or also conferencing. The MOS value is presented as polynomial equation with constants a to j and with packet loss ratio and delay as input parameters (Eq. 7.1). The constants depend on the used codec. We configure *D-ITG* to emulate G.729. The MOS function $MOS_{VoIP}(loss, delay)$ is then described by Eq. 10 and Table II in [146]. Equation 7.1 gives the resulting equation with the specific constants for the codec. We define M accordingly as $M^{(\text{VoIP})}(\text{loss}, \text{delay}) := MOS^{(\text{VoIP})}(\text{loss}, \text{delay}) - 1) \cdot \frac{4}{2.65} + 1$.

$$\begin{aligned} MOS^{(\text{VoIP})}(\text{loss}, \text{delay}) = 3.61 - 0.13 \cdot loss + 1.22 \cdot 10^{-3} \cdot delay + 3.76 \cdot 10^{-3} \cdot loss^2 - \\ 2.29 \cdot 10^{-5} \cdot delay^2 + 4.71 \cdot 10^{-6} \cdot loss \cdot delay - \\ 5.16 \cdot 10^{-5} \cdot loss^3 + 2.54 \cdot 10^{-8} \cdot delay^3 + \\ 1.28 \cdot 10^{-7} \cdot loss \cdot delay^2 - 4.43 \cdot 10^{-8} \cdot loss^2 \cdot delay \end{aligned} \tag{7.1}$$

7.4.3 Utility Functions

The utility function U_a : (Throughput [Kbps], Delay [ms]) $\mapsto [1, 5]$ approximates the QoE-aware utility for a specific application type a for a unidirectional pacing rates and maximum delay threshold using the utility model. Hence, the function solves the problem of linking network resource demands with the resulting user experience. The hereinafter described methodology for constructing the utility functions can be applied in an automated fashion to any enterprise application and its intents.

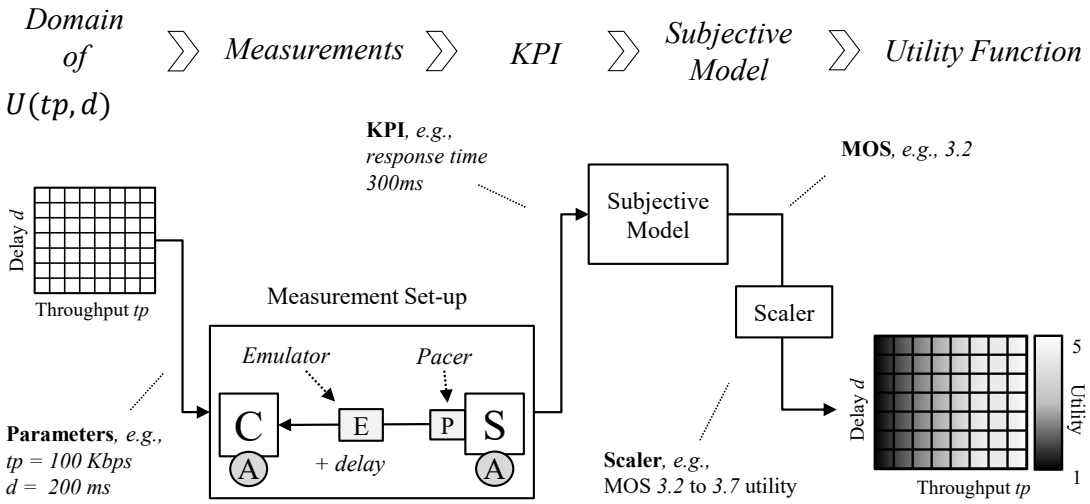


Figure 7.4: Utility functions for *(class, intent)* are generated by first defining a measurement domain in terms of throughput and delay. Second, the domain is quantized and the application KPIs are measured in an emulated network environment using the quantized parameters for throughput and delay. Third, user experience models are used to derive the utility for the measured parameters.

Figure 7.4 illustrates the process of constructing the utility functions. A set-up measures the utility of each application and intent for different pacing rates and delays in an isolated environment. Two hosts (Host S and Host C) with the applications (Ⓐ) are connected through a network emulator. On the emulator, Linux *netem* is adding delay to all packets passing through it. Host S is running the server endpoint of the application, e.g., in case of web browsing a HTTP web server. The client endpoint is assigned to Host C, e.g., the web browser. Host S egress traffic is paced using the *cfg* queuing discipline (Section 7.6.2). From the measurements we derive the 2-dimensional utility functions. Note that to account for asymmetric data-rates in a conversation, which is the case for the most server-client traffic such as web traffic, the two directions of a conversation have to be described by different utility functions. For the sake of simplicity, we consider only one direction per conversation as constrained and only present the server-to-client utility functions. For the throughput, we measure *DL* in the range of $[100, 5000]$ Kbps, *WEB* in the range of $[100, 12000]$ Kbps, *VoD* and *Live* in the range of $[750, 5000]$ Kbps and *VoIP* and *SSH* in the range of $[100, 500]$ Kbps. For the delay, we measure *WEB*, *DL*, *VoD*, *Live* in the range of $[0, 240]$ ms and *VoIP* and *SSH* in the range of $[0, 500]$ ms. The maximum pacing rate per intent is set so that further increasing the pacing rate does not improve the utility for any delay demand.

Figure 7.5 presents the measurement results for the utility of the applications depending on delay and throughput. The intersections of the grid indicate the quantization as used by the resource allocation problem formulation. The figure shows that *DL*, *WEB*, and *VoD* are highly dependent on the throughput and only a minor dependency on delay is visible. *Live* depends

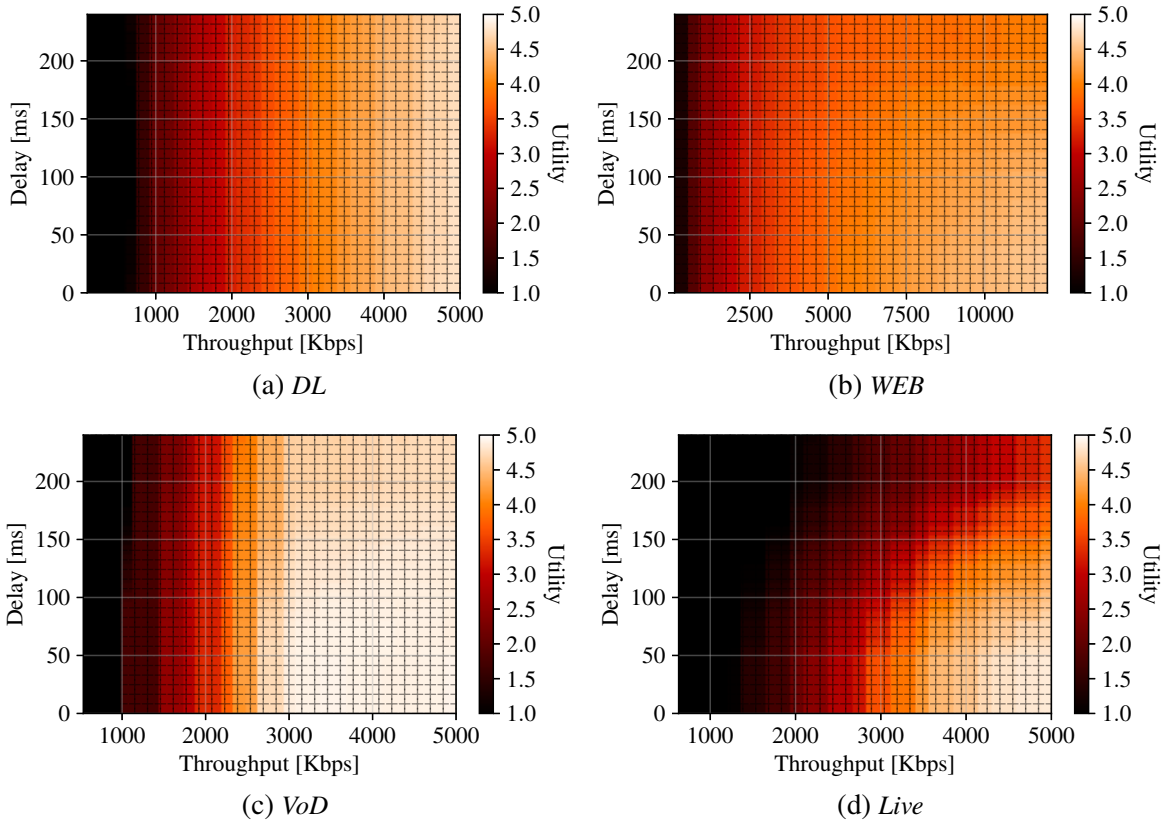


Figure 7.5: Utility functions U_a: (Throughput, Delay) $\mapsto [1, 5]$ which map throughput and delay to utility for the application classes file download, web browsing and video streaming and intents defined in Table 7.2.

on delay and throughput. *SSH* (not shown) depends solely on the delay. For *DL* (Fig. 7.5a), the impact of the delay is limited to the TCP handshake, the file request and acknowledgements packets. The impact is insignificant compared to the download time and not visible on the figure. For *VoD*, the impact of delay depends additionally on the number and playtime duration of video segments and the adaptation strategy. As illustrated by Figure 7.5c, the influence of delay for the intent *VoD* is minor. For *Live* there is a clear influence of delay on the utility (Fig. 7.5d). For *SSH*, the delay is the important influence factor, as every typed character triggers an outgoing packet and requires an immediate response packet. As we use persistent HTTP connections for web browsing, there is no influence of the delay on the *WEB* utility due to the TCP handshake. The influence of the delay is limited to the requests of the Hyper Text Markup Language (HTML) index object and the embedded resources (Fig. 7.5b).

The maximum utility values an application can reach in the measurements are determined by implementation-specific factors and the domain and range of the utility function. For example *WEB* is limited by the browser processing time and *VoD*/*Live* depend on the behavior of the adaptation algorithm. *SSH* can reach the highest utility of 5 with 100 Kbps throughput

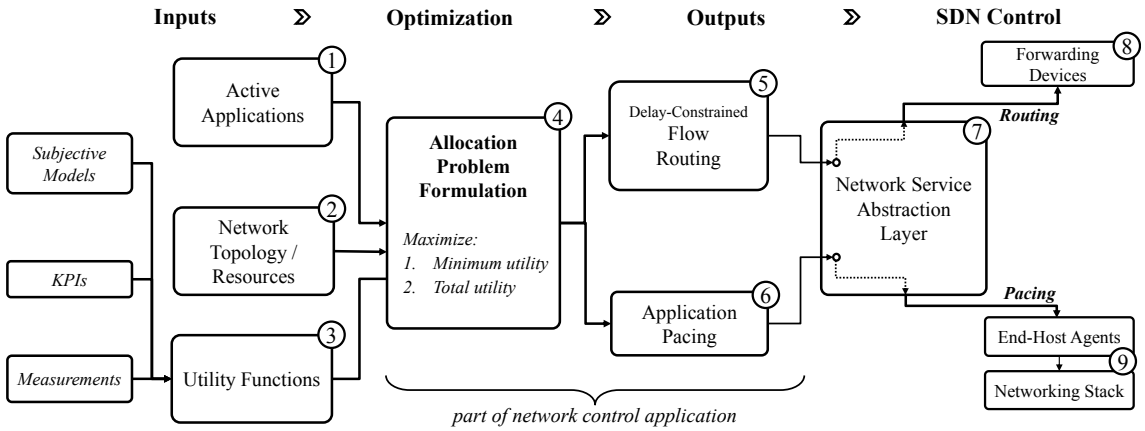


Figure 7.6: Overview over the fair resource allocation problem. Based on the set of applications (①), resources (②) and application utility functions (③), the allocation problem solver maximizes the minimum and total utility over all active applications (④). As a result, delay-constrained flow routing (⑤) and application pacing rates (⑥) are implemented by the NSAL (⑦) in the network (⑧) and on the end-hosts (⑨).

and 0 ms delay. *VoIP* can reach 5 with 100.0 Kbps and 34.5 ms. For *WEB* the highest utility is 4.5 with 11589.7 Kbps and 33.1 ms delay. *VoD* can reach its highest utility of 4.9 with 3479.3 Kbps and 41.4 ms. *Live* can reach 4.8 with 5000.0 Kbps and 41.4 ms. *DL* can reach an utility of 4.8 with 5000 Kbps and 99.3 ms delay.

7.5 User-Level Resource Allocation Formulation

We formulate the problem of network-wide application fair resource allocation as a MILP. The objective of the MILP is in the first step to maximize the minimal utility value $\theta^{(\min,1)}$ over all applications (*max-min fairness*). In the second step the MILP maximizes the sum of all utility values while the minimum utility $\theta^{(\min,2)}$ is restricted to the range $\theta^{(\min,2)} \in [\theta^{(\min,1)} - \epsilon, \theta^{(\min,1)}]$ with $\epsilon = 0.3$.

Next, we give the complete description of the resource allocation problem formulated as a MILP. The MILP has to consider the two-dimensional utility function of every application, the capacities of all paths between application endpoints and the queuing time on intermediate nodes depending on the link utilization. The decision variables describe which pacing rate to apply to which application and how to configure the routing between application endpoints. The problem can be summarized with the following inputs, objectives, high-level constraints and outputs.

Inputs
(I) Number of applications.
(II) Utility function U of each application.
(III) Network topology with link capacity information and delay on the links based on link utilization.

Objectives
(I) Min-max utility fairness in the first step.
(II) Increasing average utility in the second step.

Constraint Unidirectional application routing (source to destination) has to be valid, considering link capacity and maximum delay per application.

Outputs
(I) Target utility value and allocated throughput per application.
(II) Application flow routing.

7.5.1 Notation

Table 7.4 summarizes the notation. $\mathcal{A}, a \in \mathcal{A}$ is the set of all unidirectional application flows a. For simplification, application a and intent i are merged in the notation to only a. The two directions of a bidirectional application flow are considered as two independent applications by the formulation. This allows different paths and utility functions for both flow directions. We define the topology as a directed graph $G(\mathcal{V}, \mathcal{E})$ with nodes $v \in \mathcal{V}$ and edges $(u, v) \in \mathcal{E}$ and edge capacity $C_{u,v}$. A flow a is defined by the source node S_a, target node T_a and its utility function U. The utility function describes the relationship between allocated throughput and delay and the application's resulting utility (Fig. 7.5). It can be determined for example through measurements and user experience models, as we do in the paper at hand in Chapter 7.4. Mathematically, the utility function is split into its three components, the utility values (U), the throughput demands (η) and the delay demands (D). F describes the application flow routing. An edge (u, v) is traversed by an application a if $F_{a,u,v}$ equals 1. Delay on a link is describes as a function of the link usage. ψ and Ψ define the piece-wise defined relationship between usage (ψ) and resulting delay (Ψ).

7.5.2 Objective

The objective of the MILP is in the first step to maximize the minimal utility value $\theta^{(\min)}$ over all applications. In the second step the MILP maximizes the sum of all utility values while the minimum utility is $\theta^{(\min)}$ restricted to range based on the minimum value determined by the first step, denoted as $\theta^{(\min,1)}$, $\theta^{(\min)} \in [\theta^{(\min,1)} - \epsilon, \theta^{(\min,1)}]$ with $\epsilon = 0.3$. The second step allows the problem formulation to improve the average utility over all applications by relaxing

Table 7.4: Notation Allocation Problem Formulation

Symbol	Type	Unit	Description
		Constants	
$G(\mathcal{V}, \mathcal{E})$			Network topology graph with nodes $\mathcal{V}$ and edges $(u, v) \in \mathcal{E}$.
$\mathcal{A}, a \in \mathcal{A}$			Set of all unidirectional application flows.
S, T	$\in \mathcal{V}^{\lvert\mathcal{A}\rvert}$		Start and target nodes of application flows.
ψ, Ψ	$\in \mathbb{R}_+^{\lvert\mathcal{E}\rvert\times\lvert\mathcal{E}\rvert\times m}$		Translation between link usage and delay for a specific link.
C	$\in \mathbb{R}_+^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{E}\rvert}$	Kbps	Unidirectional link capacity between u and v.
τ	$\in \mathbb{R}_+^{\lvert\mathcal{A}\rvert\times n}$	Kbps	Utility functions' throughput demands of the applications.
δ	$\in \mathbb{R}_+^{\lvert\mathcal{A}\rvert\times n}$	ms	Utility functions' delay demands of the applications.
U	$\in ([1, 5])^{\lvert\mathcal{A}\rvert\times\lvert\tau\rvert\times\lvert\delta\rvert}$		Utility functions' utility values of the applications.
		Decision Variables	
$\theta^{(\min)}$	$\in [1, 5]$		Minimum utility for all applications.
T	$\in \{0, 1\}^{\lvert\mathcal{A}\rvert\times\lvert\tau\rvert}$		1 if a specific throughput demand index is selected.
Δ	$\in \{0, 1\}^{\lvert\mathcal{A}\rvert\times\lvert\delta\rvert}$		1 if a delay demand index for application is selected.
F	$\in \{0, 1\}^{\lvert\mathcal{A}\rvert\times\lvert\mathcal{E}\rvert\times\lvert\mathcal{E}\rvert}$		1 if an edge is traversed by an application.
		Functions	
$\eta(a)$	$\mathcal{A} \mapsto \mathbb{R}_+$	Kbps	Selected throughput for application a.
$D(a)$	$\mathcal{A} \mapsto \mathbb{R}_+$	ms	Selected delay requirement for application a.
$\Lambda(a)$	$\mathcal{A} \mapsto [1, 5]$		Target utility value of application a.
$\Omega(u, v)$	$\mathcal{E} \mapsto \mathbb{R}_+$	Kbps	Assigned throughput to link (u, v) in Kbps.
$\omega(u, v)$	$\mathcal{E} \mapsto \mathbb{R}_+$	ms	Delay on link (u, v) in milliseconds.
$\Upsilon(a)$	$\mathcal{A} \mapsto \mathbb{R}_+$	ms	End-to-end delay of application a in milliseconds.
		Miscellaneous	
$\theta^{(\min,\{1\lvert 2\})}$	$\in [1, 5]$		Solution of $\theta^{(\min)}$ in first and second step.
ϵ [= 0.3]	$\in \mathbb{R}^+$		Slack parameter for $\theta^{(\min)}$ in the second step.
n, m	$\in \mathbb{N}$		Quantification factors for the utility and link delay functions.

the max-min fairness constrain using the slack parameter ϵ. This prevents solutions where the optimization would stop when the utility of a single application can not be increased further, but where there are plenty of resources left to increase the utility of other applications.

We define θ_a as utility value of an application a. In the first step we maximize the minimum utility value (*max-min fairness*) subject to all application utilities have to be larger than the minimum utility value $\theta^{(\min)}$:

$$\text{maximize:} \quad \theta^{(\min)} \tag{7.2}$$

$$\text{subject to:} \quad \Lambda(a) \geq \theta^{(\min)} \quad \forall a \in \mathcal{A} \tag{7.3}$$

$$\text{and (7.8) - (7.27)} \tag{7.4}$$

We denote the optimal value of $\theta^{(\min)}$ of the first step as $\theta^{(\min,1)}$. In the second step we relax the max-min constraint by ϵ and maximize the sum of all utility values. We denote the optimal value of $\theta^{(\min)}$ of the second step as $\theta^{(\min,2)}$ and add the additional constraint to bound $\theta^{(\min,2)}$ by $\theta^{(\min,1)} - \epsilon = 0.3$:

$$\text{maximize:} \quad \sum_{a \in \mathcal{A}} \Lambda(a) \tag{7.5}$$

$$\text{subject to:} \quad \theta^{(\min)} \geq \theta^{(\min,1)} - \epsilon \tag{7.6}$$

$$\text{and (7.8) - (7.27)} \tag{7.7}$$

For remainder of this formulation and if not otherwise stated, $\theta^{(\min)}$ denotes the optimal value as determined by the second step ($\theta^{(\min,2)}$). Next we formulate the constraints. Table 7.5 summarizes the constraints.

Table 7.5: Overview of all constraints

Type	Constraints	Description
Objectives	(7.3), (7.6)	Maximize minimum utility (1st step) and sum of utilities (2nd step).
Utility	(7.8) - (7.12)	Select target utility, throughput allocation and maximum allowed delay per application.
Routing	(7.13) - (7.15)	Application routing (multi-commodity flow problem).
Capacity	(7.16) - (7.17)	Link capacity (in Kpbs) can not be exceeded by applications.
Delay	(7.18) - (7.27)	Determine delay per link (in milliseconds) depending on link usage. Ensure applications' maximum delay demand is not exceeded.

7.5.3 Utility Selection Constraints

For each application, one throughput, delay and target utility value have to be selected. We first introduce the equations and afterwards illustrate the selection process by a simplified example. Eq. 7.8 and Eq. 7.9 dictate that only one throughput and delay demand for application a can be chosen at a time:

$$\sum_{i=1}^{|\mathrm{T}_a|} \mathrm{T}_{a,i} = 1 \quad \forall a \in \mathcal{A} \tag{7.8}$$

$$\sum_{i=1}^{|\Delta_a|} \Delta_{a,i} = 1 \quad \forall a \in \mathcal{A} \tag{7.9}$$

Hence the chosen throughput demand in Kbps η^a and delay requirement in milliseconds D^a for application a are given by the following element-wise multiplications.

$$\eta(a) := \mathrm{T}_a^T \cdot \tau_a \tag{7.10}$$

$$D(a) := \Delta_a^T \cdot \delta_a \tag{7.11}$$

The resulting utility value of application a, $\Lambda(a)$, is then selected from the quantified utility functions (Fig. 7.5) by the following equation:

$$\Lambda(a) := \sum_{tp=1}^{|\mathrm{T}|} \sum_{d=1}^{|\Delta|} (\mathrm{T}_{a,tp} \cdot \Delta_{a,d} \cdot U_{a,tp,d}) \tag{7.12}$$

Next we give an example for a target utility, throughput and delay demands calculations for an arbitrary application a. The discretized utility function U_a has a domain of [100, 500, 1000] Kbps for the throughput and [150, 100, 50] milliseconds for the delay demand. At an allocation of 1000 Kbps and 50 ms the utility of the application reaches its highest point with 4.9, while for 100 Kbps and 150 ms the target utility drops to 1.3. In the following example the decision variables $\mathrm{T}_{a,1}$ and $\Delta_{a,1}$ are set to 1 by the solver based on other constraints like the available link capacity. Hence, an allocation of $\eta(a) = 500$ Kbit/s is chosen with a target utility of $\Lambda(a) = 3.0$.

$$\Lambda(a) = \begin{array}{c} \\ \Delta_{a,0} \\ \Delta_{a,1} \\ \Delta_{a,2} \end{array} \begin{array}{c} \begin{array}{ccc} \mathrm{T}_{a,0} & \mathrm{T}_{a,1} & \mathrm{T}_{a,2} \end{array} \\ \begin{pmatrix} U_{a,0,0} & U_{a,1,0} & U_{a,2,0} \\ U_{a,0,1} & U_{a,1,1} & U_{a,2,1} \\ U_{a,0,2} & U_{a,1,2} & U_{a,2,2} \end{pmatrix} \end{array} = \begin{array}{c} \\ 0 \\ \mathbf{1} \\ 0 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & \mathbf{1} & 0 \end{array} \\ \begin{pmatrix} 1.3 & 1.6 & 2.1 \\ 2.9 & \mathbf{3.0} & 3.5 \\ 4.2 & 4.3 & 4.9 \end{pmatrix} \end{array} = 3.0$$

$$\eta(a) = \begin{pmatrix} \tau_{a,0} \\ \tau_{a,1} \\ \tau_{a,2} \end{pmatrix} \cdot \begin{pmatrix} \mathrm{T}_{a,0} & \mathrm{T}_{a,1} & \mathrm{T}_{a,2} \end{pmatrix} = \begin{pmatrix} 100 \\ \mathbf{500} \\ 1000 \end{pmatrix} \cdot \begin{pmatrix} 0 & \mathbf{1} & 0 \end{pmatrix} = 500\,\text{Kbit/s}$$

$$D(a) = \begin{pmatrix} \delta_{a,0} \\ \delta_{a,1} \\ \delta_{a,2} \end{pmatrix} \cdot \begin{pmatrix} \Delta_{a,0} & \Delta_{a,1} & \Delta_{a,2} \end{pmatrix} = \begin{pmatrix} 150 \\ \mathbf{100} \\ 50 \end{pmatrix} \cdot \begin{pmatrix} 0 & \mathbf{1} & 0 \end{pmatrix} = 100\,\text{ms}$$

7.5.4 Routing Constraints

We formulate the application flow routing problem as the multi-commodity flow problem [85] with non-fractional flows. First we formulate the constraints required to route the flow from source to destination. Afterwards we formulate the link capacity and application delay

constraints. A flow is subject to the following routing constraints. Number of incoming and outgoing edges of in-between nodes has to be equal (*flow conservation*):

$$\sum_{w \in \mathcal{V}} F_{a,u,w} = \sum_{w \in \mathcal{V}} F_{a,w,u} \mid u \neq T_a, S_a \quad \forall a \in \mathcal{A}, \forall u \in \mathcal{V} \tag{7.13}$$

Flow conservation at the source (Eq. 7.14) and destination (Eq. 7.15):

$$\sum_{w \in \mathcal{V}} F_{a,S_a,w} - \sum_{w \in \mathcal{V}} F_{a,w,S_a} = 1 \quad \forall a \in \mathcal{A} \tag{7.14}$$

$$\sum_{w \in \mathcal{V}} F_{a,w,T_a} - \sum_{w \in \mathcal{V}} F_{a,T_a,w} = 1 \quad \forall a \in \mathcal{A} \tag{7.15}$$

7.5.5 Capacity Constraints

Capacity constraints ensure that the assigned throughput to a link does not exceed the capacity of the link. Next we formulate the required link capacity constraints. We define the link usage in Kbps $\Omega(u, v)$ on the directed edge (u, v) as the sum of the throughput values of all applications traversing that edge/link:

$$\Omega(u, v) := \sum_{a \in \mathcal{A}} F_{a,u,v} \cdot \eta(a) \tag{7.16}$$

And assigned throughput can not exceed the capacity:

$$\Omega(u, v) \leq C_{u,v} \quad \forall (u, v) \in \mathcal{E} \tag{7.17}$$

7.5.6 Delay Constraints

We define the delay of each link as a function of the link usage. That way, the delay function can express a combination of constant, e.g., propagation delay, and dynamic, e.g., queuing and processing delay, use cases. For example, an added constant delay can describe significant propagation delay, or the queuing delay can be modeled based on the target link utilization. We first provide the necessary equations and then provide a simple example.

We do a piece-wise linear interpolation to approximate the link delay for edge (u, v), denoted as $\omega(u, v)$, for a given link usage $\Omega(u, v)$ of the edge. $\psi_{u,v,i}$ and $\Psi_{u,v,j}$ describe the piece-wise defined translation sets between a usage in Kbps with index i and delay in milliseconds with index j for a link (u, v) with $|\psi_{u,v}| = |\Psi_{u,v}|$. We introduce the variables $l_{u,v,p}$

with $l_{u,v,p} \in \{0, 1\}$ and $S^{u,v,p} \in [0, 1]$ for $p = \{0, 1, .., |\psi_{u,v}| - 1\}$. Variable l selects the closest, lower, link usage from ψ and S is the linear scaling factor. l and S are subject to:

$$S_{u,v,p} \leq l_{u,v,p} \quad \forall (u, v) \in \mathcal{E}, \;\; p = \{0, 1, .., |\psi_{u,v}| - 1\} \tag{7.18}$$

Constrain the selection variable $l_{u,v,p}$ and scale variable $S_{u,v,p}$ according to the link usage $\Omega_{u,v}$:

$$\Omega(u, v) - \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \psi_{u,v,p} + (\psi_{u,v,p+1} - \psi_{u,v,p}) \cdot S_{u,v,p}] = 0 \quad \forall (u, v) \in \mathcal{E} \tag{7.19}$$

$\omega(u, v)$ then defines the delay for the given link usage:

$$\omega(u, v) := \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \Psi_{u,v,p} + (\Psi_{u,v,p+1} - \Psi_{u,v,p}) \cdot S_{u,v,p}] \tag{7.20}$$

Let's consider the following simple example. A hypothetical link (u, v) has a maximum capacity of 1000 Kbps and a propagation delay of 10 ms. Up to a link usage of 100 Kbps, there is no queuing delay. Between 100 Kbps and 1000 Kbps the queuing delay increases linearly up to a maximum of 70 ms. Hence, at a link usage of 1000 Kbps the delay on the link is 70 ms + 10 ms = 80 ms. We can model this by setting ψ and Ψ as follows:

$$\psi_{u,v} = \begin{pmatrix} \psi_{u,v,0} \\ \psi_{u,v,1} \\ \psi_{u,v,2} \end{pmatrix} = \begin{pmatrix} 0 \\ 100 \\ 1000 \end{pmatrix} Kbps \quad \Psi_{u,v} = \begin{pmatrix} \Psi_{u,v,0} \\ \Psi_{u,v,1} \\ \Psi_{u,v,2} \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \\ 80 \end{pmatrix} ms$$

Let us assume the decision variables assign link (u, v) a total link usage of 500 Kbps. The resulting total delay on that link can then be calculated by first determining l and S:

$$\Omega(u, v) - \sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \psi_{u,v,p} + (\psi_{u,v,p+1} - \psi_{u,v,p}) \cdot S_{u,v,p}] = 0$$

$$\leftrightarrow 500 - ([l_{u,v,0} \cdot 0 + (100 - 0) \cdot S_{u,v,0}] + [l_{u,v,1} \cdot 100 + (1000 - 100) \cdot S_{u,v,1}]) = 0$$

The statement is true for $l_{u,v} = [0, 1]$ and $S_{u,v} = [0, 0.\bar{4}\bar{4}]$. The delay on the link is then calculated as follows:

$$\begin{aligned}\omega(u,v) = &\sum_{p=0}^{|\psi_{u,v}|-1} [l_{u,v,p} \cdot \Psi_{u,v,p} + (\Psi_{u,v,p+1} - \Psi_{u,v,p}) \cdot S_{u,v,p}] \\ =&[l_{u,v,0} \cdot 10 + (10-10) \cdot 0] + [1 \cdot 10 + (80-10) \cdot 0.\bar{4}4] \approx 41\,\text{ms}\end{aligned}$$

The end-to-end delay of an application is then the sum of delays on the links traversed by the application. We denote the end-to-end delay of application a with $\Upsilon(a)$:

$$\Upsilon(a) := \sum_{(u,v)\in\mathcal{E}} \omega(u,v) \cdot F_{a,u,v} \tag{7.21}$$

As $\omega(u,v) \cdot F_a(u,v)$ is quadratic, we re-write it to a linear constraint. We introduce the function $\vartheta_a(u,v)$, which describes the delay of an application flow a on a single link (u,v). Furthermore we set $D^{(\max)} := 100000$ to a arbitrary large number as the maximal observable delay. ϑ_a, $\forall a \in \mathcal{A}$, is subject to:

$$0 \leq \vartheta_a(u,v) \tag{7.22}$$

$$\vartheta_a(u,v) \leq D^{(\max)} \cdot F_a(u,v) \tag{7.23}$$

$$0 \leq \omega(u,v) - \vartheta_a(u,v) \tag{7.24}$$

$$\omega(u,v) - \vartheta_a(u,v) \leq D^{(\max)} \cdot (1 - F_a(u,v)) \tag{7.25}$$

The end-to-end delay of application a can then be defined as Υ_a:

$$\Upsilon^a := \sum_{(u,v)\in E} \vartheta_a(u,v) \tag{7.26}$$

The delay of the flow is not allowed to exceed the requirement:

$$\Upsilon_a \leq D_a \quad \forall a \in \mathcal{A} \tag{7.27}$$

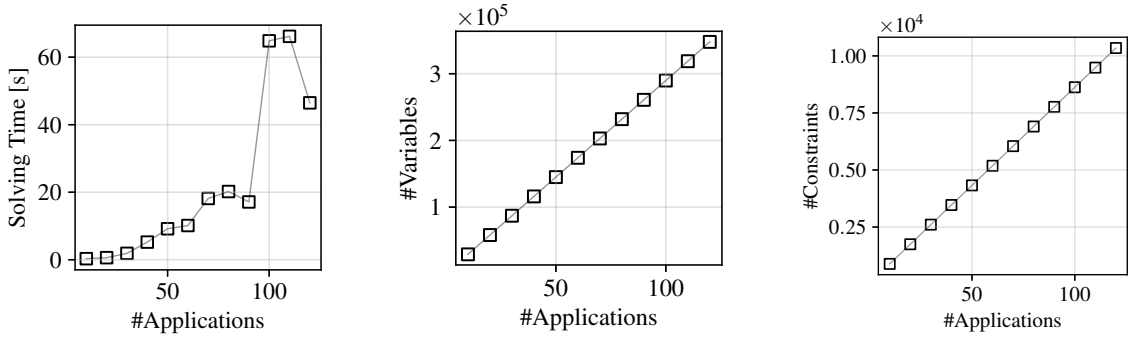
7.5.7 Problem Complexity and Possible Solving Strategies

The optimization formulation combines variations of the non-splittable multi-commodity flow problem (routing) and of the knapsack problem (balancing demand and utility), both known to be non-deterministic polynomial-time (NP)-hard. Hence, approximation algorithms have to be found to solve the formulation in a reasonable runtime for larger topologies with potentially multiple bottleneck links and a large number of simultaneous applications. The efficient and fast solving of the problem is out of scope of this work and is left to future work. This work provides the necessary abstractions and implementation proof that once the allocation decision is made, it can be efficiently and accurately be implemented in the network. As with other network resource allocation problems, such as the Virtual Network Embedding (VNE) problem, the efficient solving of the theoretical problem can now be explored independently of the implementation concepts.

Solving the problem for our evaluation scenario (one bottleneck link, $\leq$ 120 applications) takes on average less than one minute on a standard eight-core Intel Core i7-4770 3.4 GHz desktop PC with 32 GB RAM using the commercial Gurobi[1] solver. In detail, Figures 7.7 illustrate the solving time, total number of variables and total number of constraints of the problem instances with increasing number of applications with one bottleneck link. The solving time stays below 10 s up to approximately 50 applications. Between above 90 applications the solving time increases drastically up to 66.2 s. Afterwards, when high number of applications does not leave much room for allocating higher utility values, the solving time decreases again. Figures 7.7b and 7.7c show that the number of variable increases linearly with the number of applications with 2889 variables and 86 constraints for each additional application. Thus, the total number of variables and constraints depends on the number of applications, on the used quantification of the utility and link delay functions and on the size of the network topology.

One greedy algorithm for finding a viable solution could be to start with a target utility of 1.0 for all application flows and shortest path routing. Subsequently the utility can be increased by increments of 0.1 in a round-robin order until an allocation is reached where no application's utility can be increased anymore without violating capacity or delay constraints. One problem with this algorithm is that it does not find sophisticated solutions where the utilization of one path is kept low to support low volume-low delay applications, e.g., web browsing, and other paths are dedicated to batch transfers, e.g., file download.

[1] http://www.gurobi.com/



(a) Solving Time (b) Variables (c) Constraints

Figure 7.7: Problem size and solving time of the optimization formulation for increasing number of applications ($|\mathcal{A}|$) sharing *one* bottleneck link. Maximum of 66.2 s solving time for 110 applications. 2896 variables and 86 constraints for each additional application.

7.6 Experiment Design and Set-up

The objective of the experiments is to show the dependability and scalability of resource allocation via end-host pacing through the NSAL and how the different application classes profit and/or suffer from the enforced packet pacing. The experiments are conducted in a set-up where we monitor sets of increasing number of parallel applications sharing a throughput-constrained link. For each set of applications we measure the utility with and without resource allocation and discuss the differences in the evaluation. Dynamic embedding of applications at run-time and additional intents are out of scope of this evaluation. Next, we elaborate on the deployed experimental set-up (Section 7.6.1) and the custom pacing implementation (Section 7.6.2). Afterwards, we discuss the experiment parameters (Section 7.6.3). The results of the evaluation are presented in the subsequent Section 7.7.

7.6.1 Experiment Set-up

The set-up consists of two groups of hosts, one server and one client group, connected via a link. The link is throughput-constrained and the applications running on the host groups have to share the limited throughput. Figure 7.8 illustrates the experiment set-up. The network consists of two switches, one SDN-enabled Pica8 P-3290 (①) and one unmanaged off-the-shelf 100 Mbps switch (②). The link between the two switches constrains the available data-rate between the hosts on the left and on the right side to 100 Mbps. The Pica8 switch is equipped with a maximum queue size of 1 MB and maximum queuing delay of about 80 ms towards the 100 Mbps link. We deploy three modern desktop PCs on each side to meet the processing and memory resources required by the experiment scenarios.

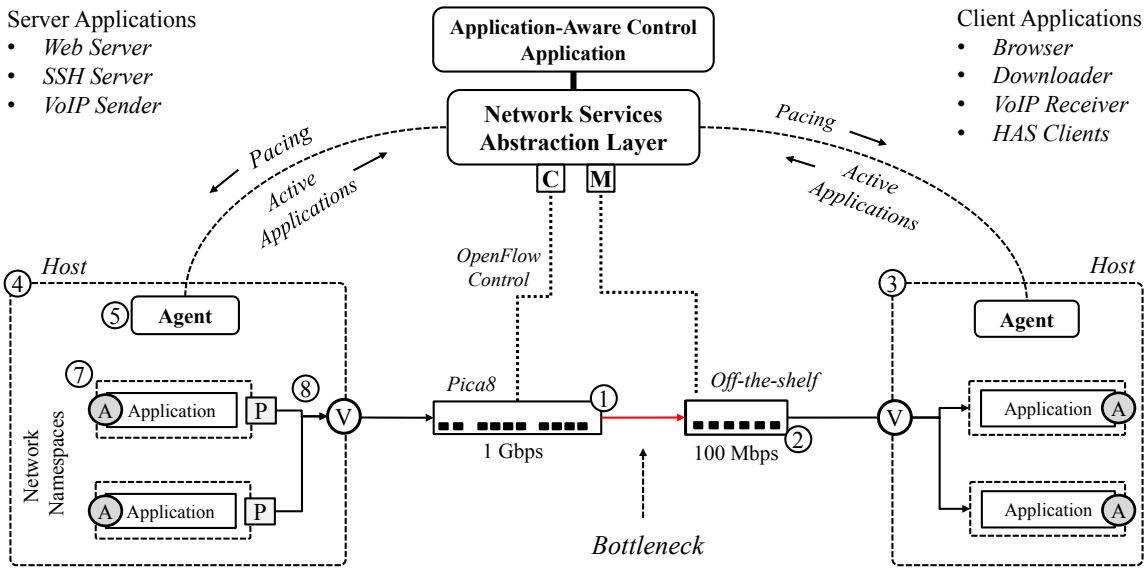


Figure 7.8: Evaluation set-up. Two groups of hosts, one server (④) and one client (③) applications group, are connected via an SDN-capable 1 Gbps switch (①) and an unmanaged 100 Mbps legacy device (②), i.e., a bottleneck, to each other. The application-aware control application on top uses the NSAL to collect statistics, available resources and active applications (Ⓐ) in the network. Afterwards the control application calculates the resource shares and uses the NSAL to configure the pacers [P] accordingly.

Each application consists of a server and client endpoint, e.g., a web server and a browser. All endpoints are confined to a separate network namespace (⑦) and connected via virtual interfaces and a software bridge to the host's physical interface (⑧). Each namespace is configured with a unique Internet Protocol (IP) and Medium Access Control (MAC) address. Furthermore, every client is connected to an exclusive server application. That way, the pacing rate can be set per namespace and no further control is needed to assign outgoing server packets to different pacers. In case of web browsing, video streaming, and web download, each client is assigned to an exclusive light-weight HTTP server, but with shared content. The server endpoints are placed left of the bottleneck and the client endpoints to the right of the bottleneck, which makes the egress queue and interface of the Pica8 the bottleneck. Pacers ([P]) based on our *cfq* implementation (Section 7.6.2) restrict the egress rate of the namespaces/applications towards the hosts' software bridges.

All management and monitoring operations are performed out-of-band. The KPIs of each application are measured at the client endpoint, e.g., the page load time at the browser, and reported to the NSAL and the application-aware control application. Additionally, we frequently poll the statistics counters of all physical and virtual network interfaces to measure throughput, queue length and packet loss.

7.6.2 Pacing Implementation

In Linux, pacing is implemented as a *queuing discipline*. Furthermore, a mechanism called TCP small queues [174] exerts backpressure on the applications to mitigate buffer bloat and packet loss by limiting the allowed number of Bytes per flow in the queuing discipline and device queue (default: 128 Kilobytes). Other operation systems offer similar pacing mechanisms. We implemented a custom queuing discipline based on the existing Fair Queuing (*fq*) discipline [173], referred to as Custom Fair Queuing (*cfq*). Every conversation defined by *(class, intent)* and by one or multiple sockets, can be assigned to an exclusive queue with a target packet release rate as configured by the network controller through the local agent. Packets from the queues are released time-based. The departure time of the next packet $time\_next\_packet$ is determined by the current time now, the size of the current packet pkt_len and the target pacing rate $target\_rate$: $time\_next\_packet = now + \frac{pkt\_len}{target\_rate}$

7.6.3 Parameter Space and Experiment Procedure

The parameter space of the experiments is limited to the *number* and *types* of the applications and whether the experiment is *managed* or *best effort*. In detail, the bottleneck link is shared by $\{2, 4, .., 24\}$ applications per class, in total $|\mathcal{A}| \in \{10, .., 120\}$. For video streaming, half of the applications are of type *Live* and the other half of *VoD*.

At the start of the experiment, the applications register at the NSAL through the local agent. Once all applications are registered, the network control application calculates the resource shares of utility for each application and pushes the corresponding pacing rates to the agents via the NSAL. The agents configure the pacers accordingly. The SDN-enabled Pica8 switch is configured via OpenFlow for simple forwarding. Besides the forwarding rule configuration, the OpenFlow connection is used to poll queue and interface statistics.

The duration of one experiment run is 15 minutes with an additional 1 minute warm-up and cool-down phase. The applications are started at random times during the warm-up phase and requests during the warm-up or cooldown phase are discarded for the evaluation. Each experiment is repeated 11 times. The applications are configured with a constant inter-request time of 100 ms. One request equals one video view for *VoD* and *Live*. For *VoIP*, one request equals one 30 s phone call. The reason for the static inter-request time of 100 ms is that this results in an almost constant number of concurrent applications using the bottleneck link. Hence, each application in a specific scenario is constantly sending/receiving requests/responses, except of a 100 ms break between requests to allow for a reset of an application's state. Increasing the inter-arrival time between requests would effectively decrease the number of concurrently active applications at a specific point in time. *Cubic* is

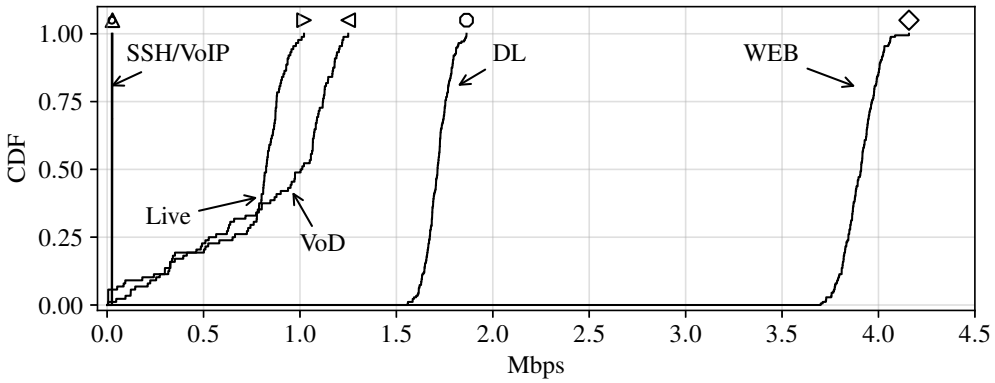
configured as TCP congestion control algorithm. Cubic is chosen as comparison as it shows better performance on congested links compared to Compound and New Reno TCP [21] and it is the default algorithm for many Linux server variants. BBR congestion control proposed by Google fails to show performance benefits and fairness in heterogeneous environments [82] compared to Cubic.

There exist valid optimal solutions to the allocation problem formulation with applications of the same type to be assigned different utility values. For easier presentation of the results, we constrain the problem formulation to choose one utility value per type tuple. The bottleneck link is modeled with a capacity of 100 Mbps. As the sum of all paced flow rates does not exceed the available capacity, and due to the short inter-request pauses of the application requests, the link in the managed case is slightly under-provisioned. Thus, a large queue build-up is unlikely and the link delay of the bottleneck is modeled with a constant delay of 2 ms. In the best effort case, the link is already over-utilized with 10 competing applications and experiences 58 ms delay and 0.5 % packet loss (discussed later in Section 7.7.3).

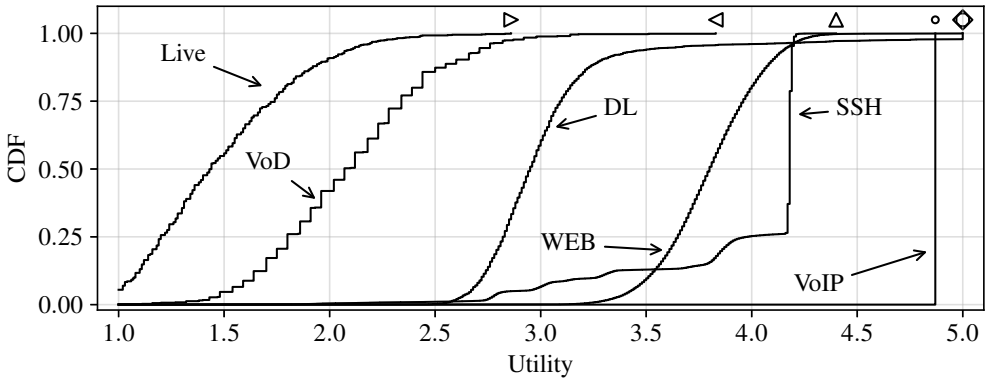
7.7 Evaluation

We evaluate the performance of an increasing number of applications sharing a throughput-constrained link with and without data-rate management. The evaluation is pursuing the following questions. i) How does the minimum and average utility of the applications compare between the managed and best effort scenarios? ii) Which applications benefit, which utility values are decreased, and why? iii) Can pacing result in configurable and thus predictable application performance in terms of the difference between the target and the measured utility? iv) How fair, in terms of utility, are the best effort and the managed utility distribution?

First, we evaluate how the available data-rate is distributed among the applications in a best effort scenario and present the resulting utility distribution. Second, we solve the allocation formulation for the scenario, implement the pacing in the set-up and present the gains in terms of utility. Third, we present how pacing affects the QoS parameters, such as packet loss and jitter, of the link. Fourth, we conduct a parameter study on the number of parallel applications and show how the gains and fairness changes with increasing number of parallel applications. Error bars in the result figures indicate the standard deviation over 11 experiment repetitions if not otherwise stated. In cases the error bars are not clearly visible on the presented scale, they are omitted from the figures.



(a) Throughput



(b) Utility

Figure 7.9: Best effort throughput and utility of the different application types for 16 clients per application class. The markers at the top are for better visual indication of the application types.

7.7.1 Best Effort Throughput and Utility Distribution

First, we take a close look at the best effort application performance for single scenario with 16 clients per application class, $16 \times \mathit{WEB}$, $16 \times \mathit{DL}$, $16 \times \mathit{SSH}$, $16 \times \mathit{VoIP}$, $8 \times \mathit{VoD}$, $8 \times \mathit{Live}$, in total $|\mathcal{A}| = 80$. The scenario with 80 applications is selected due to the fact that among the investigated scenarios, one of the highest gains is observed here. With the 80 applications competing for the bandwidth, the link is fully utilized resulting in an average packet loss of 4 % and queuing delay of 80 ms.

Figure 7.9a presents the Cumulative Distribution Functions (CDFs) of the average throughput and Figure 7.9b the CDFs of the utility values of all requests per application type. Multiple observations can be made from the figures. First, the throughput as well as the utility is distributed non-uniformly between the application types. For example, while *WEB* enjoys high throughput and utility (median $\geq$ 3.9 Mbps, 3.8 utility), *Live*'s achieved throughput is less than 1 Mbps and median utility is about 1.4. *WEB*'s high throughput is due to the use of multiple persistent parallel TCP connections, while video streaming clients, *DL*, and *SSH* establish only

one TCP connection. Parallel TCP connections allow an application to receive a proportional larger fraction of the available throughput. As web download has no idle periods during the download, web download exhibits a higher average throughput than video streaming.

Second, even *VoD* and *Live*, which belong to the same application class (video streaming) and achieve similar throughput rates, suffer from unfair utility distribution (1.3 vs. 2.1). This is due to the smaller playback buffer for live streaming and the increased encoding overhead for the shorter video chunks. Third, the average throughput of *SSH* and Voice-over-IP (*VoIP*) is below 100 Kbps, while the utility is 3.7 and 4.9, respectively. *SSH*'s performance is influenced by delay, caused by queuing at the bottleneck link, and retransmissions, due to lost packets when the bottleneck's queue is overflowing. *VoIP* is barely influenced in this scenario, as the maximum delay and packet loss over the single bottleneck is acceptable for *VoIP* traffic according to the user experience model. Details on the performance of *VoIP* is given in Section 7.7.3. Fourth, the utility distributions per application type are varying with a standard deviation of 0.2 (*WEB*) to 0.5 (*DL*), with the exception of *VoIP*. Hence, application performance is not consistent across requests of the same application type, and, as a consequence, there is an unfair distribution of shares, even within the same application type.

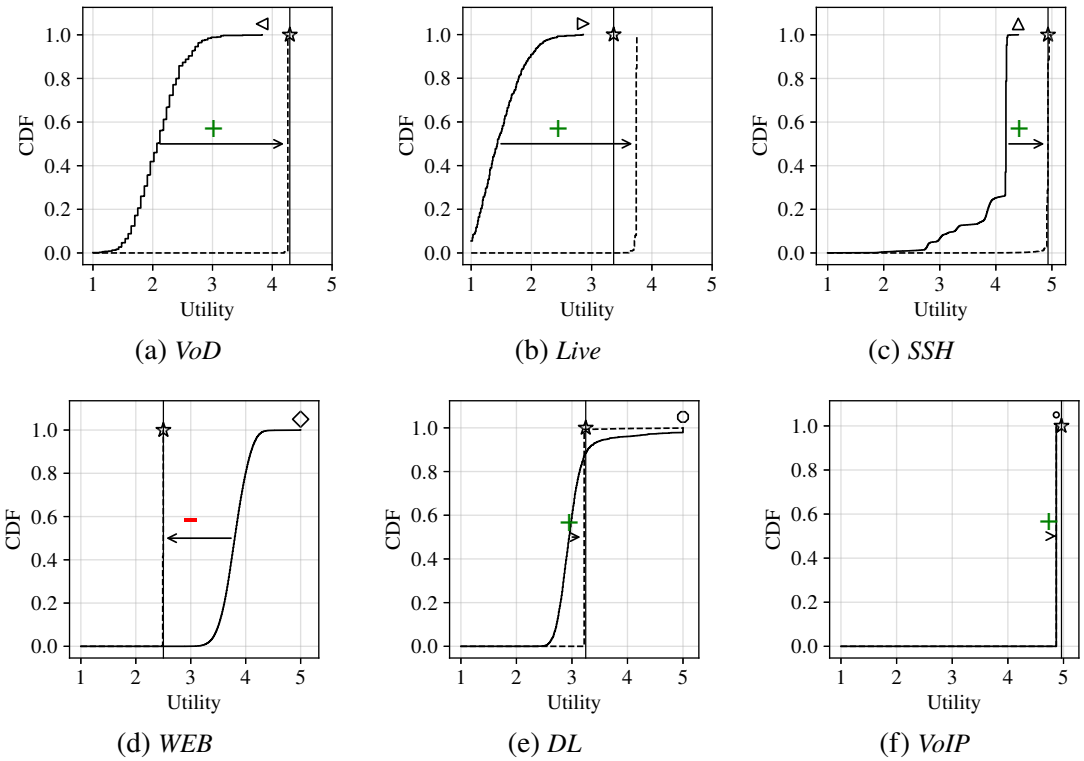
In summary, best effort delivery is inadequate to provide fair and consistent application performance for multiple applications sharing a constrained link. Best effort delivery does not consider different demands (throughput vs. delay-sensitivity), transport protocols (TCP vs. UDP), or multiple flows per application. Furthermore, the constrained link is overloaded, resulting in lost packets and queuing delay.

7.7.2 Managed Utility Distribution

Next, we solve the allocation problem formulation with the max-min fairness criteria for the scenario with 80 parallel applications and apply the calculated pacing rates. Figures 7.10a to 7.10f illustrate the best effort (solid lines) and managed utility (dashed lines) for the scenario with 16 clients per application class. Improvements in median utility due to the data-rate management are indicated by ($\rightarrow$, +). Deteriorations are shown by ($\leftarrow$, −). The target utility per application type, as calculated by the allocation formulation, is indicated by (|, $\star$).

The figures 7.10a to 7.10f show that all application types, except *WEB* and *VoIP*, profit from the management. *Live* benefits most from the management, with a median increase of 3.1 (from 1.3 to 4.4). *VoD*, *SSH* and *DL*'s median utility improve by 2.0, 1.0, and 0.4, respectively. On the other hand, *WEB*'s median utility decreases by 1.3 (from 3.8 to 2.5). No noteworthy improvement or deterioration in utility is measurable for *VoIP*.

Live (b) exhibits a deviation of about 0.5 between the target and measured utility. The deviation is the result of an inaccuracy in the live streaming utility function. The samples



(a) *VoD* (b) *Live* (c) *SSH*

(d) *WEB* (e) *DL* (f) *VoIP*

Figure 7.10: Figures (a) to (f) show measured best effort and managed application utility for $|\mathcal{A}| = 80$ applications sharing the constrained link. The dashed lines indicate the utility CDF for the managed scenario, the solid lines the best effort scenario. The star and vertical line mark the target utility *UV* for the application type. Arrows to the right highlight the improvement in median utility.

collected from the utility measurement setup are supplemented with interpolated values to build the quantized utility function. In the case of live streaming and low delay values, the interpolation results in an utility error of about 0.5. The error can be reduced by collecting more measurement samples from the throughput-delay parameter space and/or fine-tuning the interpolation algorithm.

Figure 7.11 presents the *standard deviations per client* of a specific type for the *best effort* scenario. The smaller the standard deviation is, the more consistent is the experience of a single user. The dashed vertical line indicates the maximum (= 0.05) of the standard deviations in the managed case (per type CDFs are not shown for the managed case). *DL* (◯) clients exhibit the largest median standard variation (0.64) among the application types, followed by *SSH* (△) with 0.41. *WEB* (◇) clients' median variation is the second smallest with 0.25. There is no visible variation for *VoIP* (○). The figure also shows that not only the utility value per client request varies, but also the behavior of each client. For example for *VoD* (◁), the standard deviation varies between 0.1 and 0.43. Hence, some clients experience a smaller quality variation for their video views than other clients.

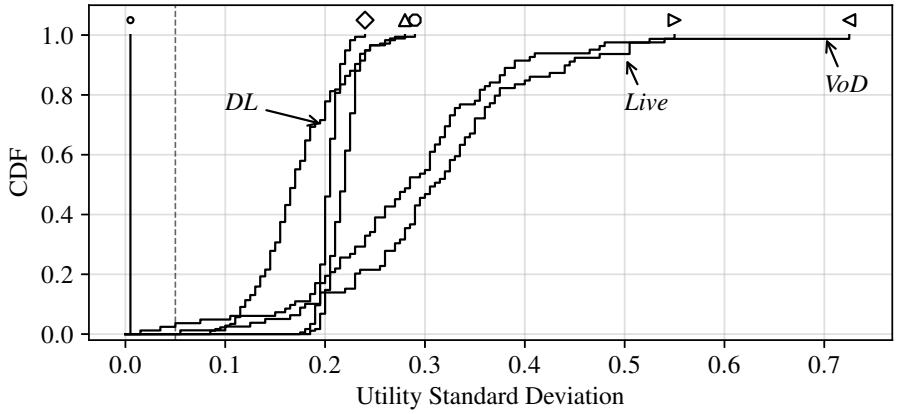


Figure 7.11: Standard deviations of a client's utility values per application type in the best effort scenario. Smaller values indicate a more consistent behavior of a client instance. The dashed vertical line indicates the maximum standard deviation for the managed case (CDFs of managed scenario not shown). Video streaming exhibits most unstable behavior. *VoIP*'s clients (○) are unaffected.

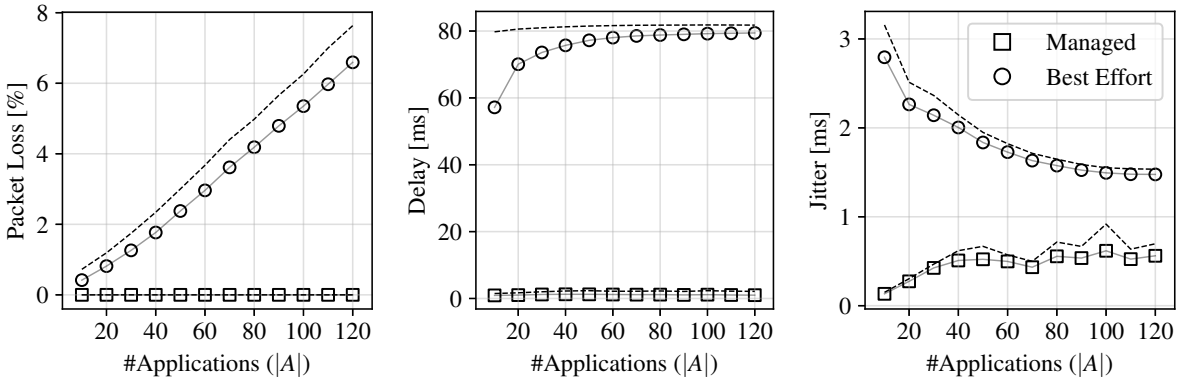
7.7.3 Link QoS and VoIP Performance Details

Next, we take a closer look at the QoS metrics of the constrained link in terms of packet loss, queuing delay and jitter for an increasing number of parallel applications. As the MOS and utility functions of *VoIP* are based on the QoS metrics, we also discuss why the QoS metrics have only minor influence on the *VoIP* performance in the evaluation.

Figure 7.12a shows the median packet loss as measured by the VoIP clients during a call for 10 to 120 parallel applications. The dashed lines indicate the 95 % percentile. The figure shows that there is no packet loss for the investigated number of applications in the managed experiments. In the best effort experiments, the packet loss increases linearly from 0.5 % to 7.1 % (0.9 % to 8.1 % for the 95 % percentile).

Figure 7.12b shows the RTT. Note that the client-to-server flow direction of the constrained link is only lightly utilized and therefore, the given RTT approximates the one-way delay experienced by the applications. In the best effort case, the delay increases roughly logarithmic from 53 ms for 10 applications and saturates for 70 parallel applications at 79 ms. The 95 % percentile shows that even at 10 parallel application the experienced RTT is in 5 % of the cases already greater than 79 ms. In the managed experiments the measured RTT increases linearly from 0.9 ms to 1.1 ms (1.5 ms to 2.4 ms).

Figure 7.12c shows the median and 95 % percentile of the average jitter as measured by the VoIP clients during a call. In general, the figure shows that in the best effort case the jitter decreases for increasing application count, while for the managed experiments the jitter increases. The decrease in jitter in the best effort case shows that due to the link saturation, there are almost constant inter-arrival times of packets. The high link utilization results in a



(a) Packet Loss (b) RTT (c) Jitter

Figure 7.12: QoS metrics of the constrained link in terms of packet loss, delay and jitter for increasing number of applications ($|\mathcal{A}|$) as recorded by the VoIP clients. The dashed lines without markers indicate the 95 % percentile. The markers indicate the managed (□) and best effort (○) median values. Without data-rate management the queue at the bottleneck is overflowing quickly even at low numbers of parallel applications and thus causing packet loss and delay.

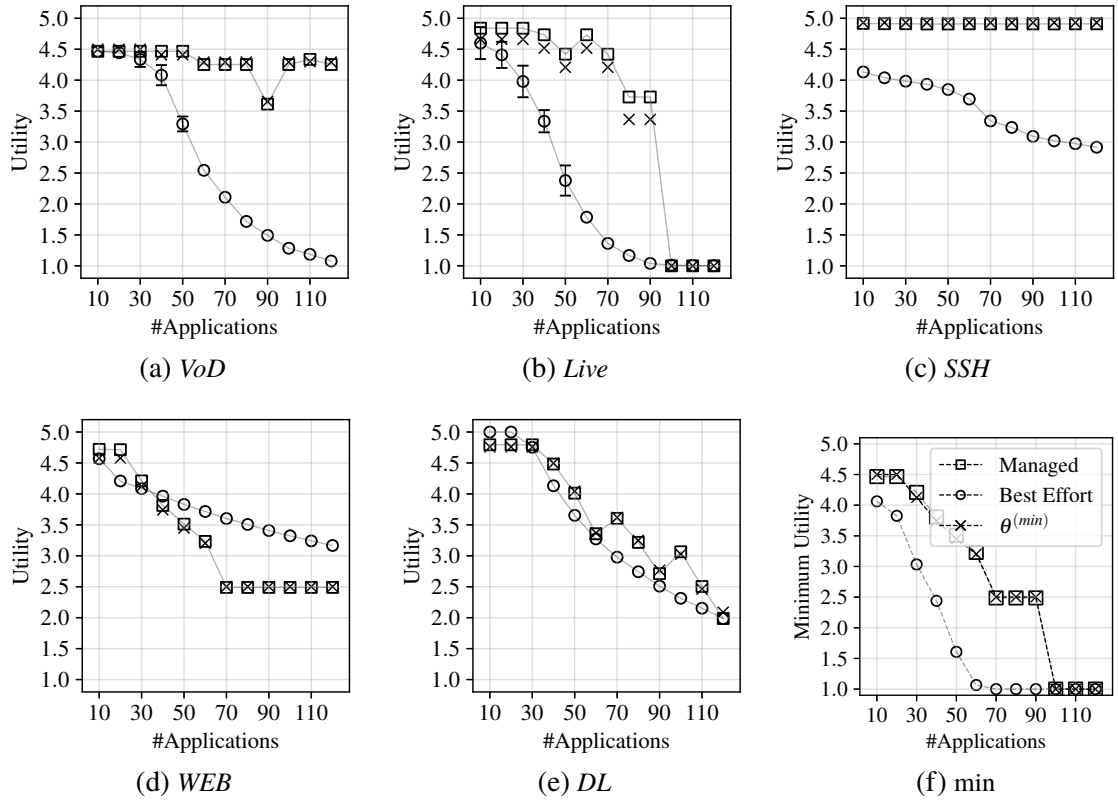
full link queue and packets are processed at line-rate by the switch's outgoing interface. In the managed case, the arrivals of the multiplexed requests of the clients result in minor RTT variations, but even for 120 applications the 95 % percentile of the jitter stays below 0.9 ms.

As there are no retransmissions for VoIP, the maximum delay for the successful transmission of a voice sample is about 80 ms in our set-up. For 8 % packet loss and 80 ms delay, the utility for VoIP is estimated as 4.9 ($U_{VOIP}(80, 0.08) = 4.9$). Hence, as defined by utility function U_{VOIP}, there is a maximum utility difference of 0.1 in the set-up (5 - 4.9).

In summary, data-rate management significantly improves the QoS metrics of the constrained link. There is no packet loss, the RTT stays in most cases far below 2.5 ms and the jitter is at least halved. Regarding the influence of the QoS metrics on the VoIP utility, the VoIP clients in combination with the selected audio codec are marginally affected by the unmanaged link degradation. However, one can imagine how applications with stricter QoS requirements or VoIP calls with longer network paths profit from the QoS improvements.

7.7.4 Increasing Number of Applications

Figure 7.13 illustrates the gain in utility per application type for increasing number of simultaneous applications. Results are shown as the mean of the 10 % percentiles of the utility values per application. The 10 % tail as summary metric is chosen to allow for a small budget of random error compared to the minimal utility over all requests, e.g., for random delays in processing on the experiment PCs or requests which take longer due to rare latency spikes in the network. Hence, on average 90 % of the requests of a client result in an utility equal or better than the given value.



(a) *VoD* (b) *Live* (c) *SSH*

(d) *WEB* (e) *DL* (f) min

Figure 7.13: Comparison of managed measurements (□), target utility (×) and best effort (○) measurements per application type for increasing number of applications sharing the constrained link. In Figure (f), the crosses and the dashed line indicate the solution to the allocation formulation ($\theta^{(\min)}$) over all types.

Figures 7.13a to 7.13e present the findings per application type. The application class VoIP is omitted as there is no significant difference between the managed and best effort scenario. Figure 7.14 summarizes the difference in utility per application type between the managed and best effort experiments. Application types with a positive difference (top half of the figure) profit from management. The performance of application types with negative differences deteriorate. The following general observations can be made based on the figures.

First, the utility for all shown types decreases with increasing number of applications in the best effort case. This is expected as with increasing $|\mathcal{A}|$ more flows compete for the scarce constrained link capacity. In the managed case, only *DL* and *WEB* exhibit an equivalent degradation in utility. *VoD*, *Live*, and *SSH* on the other hand can sustain a high utility in the managed experiments even while the number of competing flows increases. Second, for $|\mathcal{A}| < 40$ the potential gain is low as the available capacity is sufficient to reach close to maximum utility for all applications in the managed and best effort cases. Third, the performance of *WEB* deteriorates while all other classes (except *VoIP*) profit for most of the evaluated values of $|\mathcal{A}|$. Fourth, the minimum utility over all applications ($\theta^{(\min)}$) in the

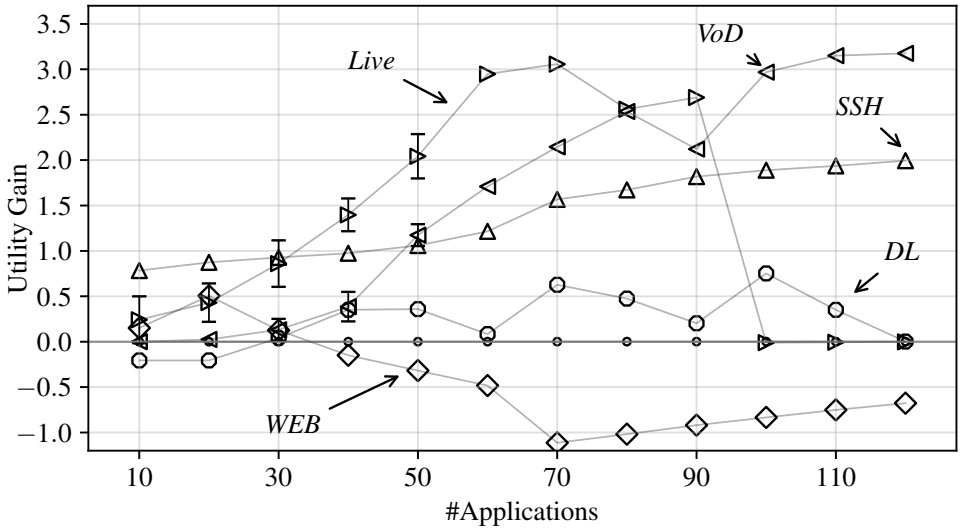


Figure 7.14: Summary of the differences between best effort and managed utility for all application classes for increasing number of applications. *SSH* and *VoD* exhibit the highest gains for > 90 number of parallel applications. *WEB* looses up to 1.1 utility by the data-rate management.

managed case is mostly determined by *WEB* and *Live*. The minimum for the best effort case is mostly dictated by *SSH* for $|\mathcal{A}| < 30$ and by *Live* for $|\mathcal{A}| \geq 30$. Fifth, the measurements from the managed scenario deviate less than 0.5 from the target utility as determined by the solution to the optimization formulation for all application types. For *DL*, *WEB*, *VoD*, and *SSH* the deviation is even less than 0.2 for the investigated number of parallel applications. Hence, data-rate management leads to predictability of application performance. Furthermore, the results show that pacing can implement the output of the allocation optimization formulation accurately. Next, we investigate the measurement results for each application type in detail.

For ***VoD*** (Fig. 7.13a), the utility decreases approximately linear with an increasing number of parallel applications for $|\mathcal{A}| > 30$. For $|\mathcal{A}| \leq 30$, best effort management is sufficient to provide a utility of 4.5 or higher. With data-rate management, the fairness formulation can allocate enough resources to the *VoD* clients to sustain a high utility value even for up to $|\mathcal{A}| = 120$. Hence, for $|\mathcal{A}| = 120$ the utility gain is about 3.1.

For ***Live*** (Fig. 7.13b), the figure shows that the utility decreases rapidly without data-rate management. There, data-rate management is most effective at 60 to 70 parallel applications where the increase is up to 3.4. In terms of predictable performance, the target utility is met most of the time with a deviation of 0.1 to 0.3. However for $|\mathcal{A}| \geq 100$, the fairness formulation decreases the utility target to the minimum of 1.0, which is the same low utility as *Live* reaches in the best effort case for the same number of applications.

For ***SSH*** (Fig. 7.13c), profit increases roughly linear with $|\mathcal{A}|$, from about 0.7 up to 2.1 for $|\mathcal{A}| = 120$. Data-rate management avoids bursts and keeps the total data-rate under

the constrained link capacity. Hence, there is little queuing delay and the delay-sensitive applications like *SSH* can sustain a high utility even for large $|\mathcal{A}|$.

For ***WEB*** (Fig. 7.13d), the difference between managed and best effort is less than 1 utility (maximum difference of 0.9 at $|\mathcal{A}| = 90$). For $|\mathcal{A}| < 90$ and $|\mathcal{A}| > 90$ the difference decreases. The target utility is close to the measured managed utility.

DL (Fig. 7.13e) exhibits the smallest utility gains (besides *VoIP*). The gain is below 0.8 for $|\mathcal{A}| \leq 90$ and around zero for $|\mathcal{A}| = 100$. The decrease of utility with increasing $|\mathcal{A}|$ is roughly linear for the managed and best effort experiments. For $|\mathcal{A}| \geq 100$, the solution to the fairness problem increases the utility target for *DL* again, which results in a utility gain close to 1.0. Managing the utility is accurate and the deviation from the target utility can be neglected for all investigated numbers of parallel applications.

VoIP exhibits no benefit or degradation from the activated management according to the user experience model (further discussed in Section 7.7.3).

Figure 7.13f shows the minimum 10 % percentile utility as measured in the best effort and managed experiments and as calculated by the fairness formulation. The figure shows that in the managed scenario, every client's utility is at least 3.0 up to 80 parallel applications, which is denoted as *fair* on the MOS scale. In the best effort case, the observed minimum utility drops below 3 for 40 applications and down to 1.0 for 80. When comparing $\theta^{(\min)}$ ($\times$) and managed ($\square$), the managed minimum utility does not differ more than 0.1 from the calculated minimum utility.

In summary, the presented measurements for increasing number of parallel applications sharing the constrained link highlight the benefits of the proposed approach. *VoD*, *Live*, *DL*, and *SSH* exhibit gains in utility between 0.5 and up to 3.3, even for 100 and more applications sharing the 100 Mbps link. *WEB*'s utility degrades, but the decrease is less than 1.0. The minimum utility $\theta^{(\min)}$ can be greatly increased, especially for $|\mathcal{A}| > 30$, and the target utility is mostly met, resulting in predictable application performance. *VoIP* shows no benefit or degradation due to the nature of its user experience model.

7.7.5 Video Streaming Performance Details

Stallings and switches are not part of our utility functions U_{VoD} and U_{Live}, but studies show their negative influence on the user's perceived video quality [4, 2]. Therefore, for video streaming, we now consider the two additional KPIs *initial stalling time* and *quality switching frequency*. Initial stalling is the short unavoidable stalling time at the beginning of the video to download the first segment. Stalling during the playback was prevented in all experiment runs by the quality adaptation algorithm.

Figures 7.15 illustrate the median initial stalling time ((a) and (b)) and the median number of quality switches ((c) and (d)) per video view for *VoD* and *Live*. In terms of initial stalling the figures show for the best effort case a strong linear correlation with the number of applications. For *VoD*, the median stalling time increases approximately linear from 0.9 s to 4.5 s for 10 to 120 parallel applications. For *Live*, the increase is from 2.0 s to 9.3 s. In the managed case the initial stalling duration does not exceed 1.9 s and 5 s for *VoD* and *Live* up to $|\mathcal{A}| < 100$, respectively. For $|\mathcal{A}| \geq 100$, the initial stalling increases to about 10 s. This is due to fairness formulation assigning *Live* the minimum bandwidth which allows interruption-free playback of the video. But with minimum bandwidth the initial buffer time increases.

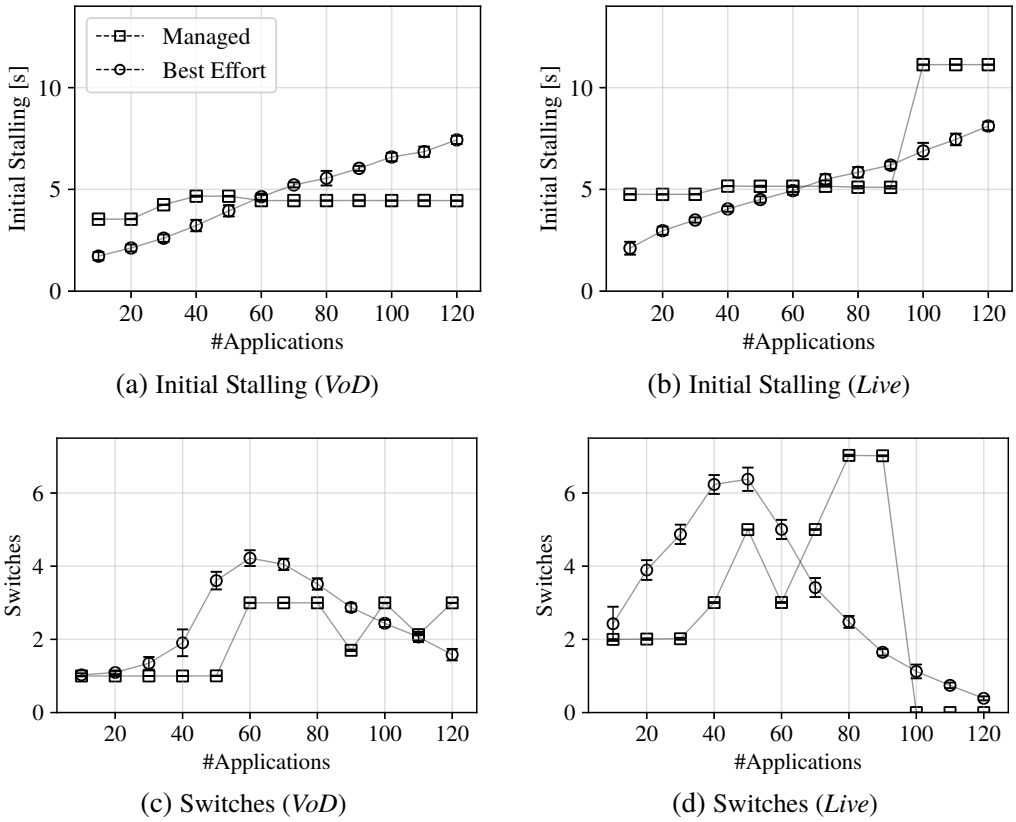
In terms of mean quality switches per playback minute, which is equal to the mean number of switches in the case of our 60 s video, we observe a difference in behavior between *VoD* and *Live*. For *VoD*, the switching frequency is low (approximately one or two switches per minute) for a small number of applications ($|\mathcal{A}| \leq 30$) and for a large number of applications ($|\mathcal{A}| > 110$). In-between, the frequency ranges between $1\,m^{-1}$ and $3\,m^{-1}$ for the managed experiments and between $2\,m^{-1}$ and $5\,m^{-1}$ for the best effort experiments. The reduced switching frequency for $|\mathcal{A}| \leq 30$ and $|\mathcal{A}| > 110$ can be explained by the fact that there are fewer reasonable choices for a quality level to choose when the goodput demand of the highest quality level is satisfied or the goodput demand for the lowest quality level is barely met.

For *Live*, the number of switches increases in the best effort experiments from 2.5 up to 6.5 switches and then decreases roughly linear to 0 for 120 applications. As shown by Figure 7.13b, the utility, and therefore the mean quality level, drops fast to the lowest level with increasing number of applications for *Live* intent and best effort congestion control. Hence, the number of switches for *Live* is also decreasing as the adaption logic has limited options and must often choose the lowest quality level. In the managed case, the switches increase from $2\,m^{-1}$ to $7\,m^{-1}$ at 90 applications. For $|\mathcal{A}| \geq 100$ the switches drop to zero as only the lowest quality level is shown to the user. The managed results show that the adaptation logic is not able to take advantage of the stable goodput for the *Live* intent.

In summary, data-rate management results in most cases in a smoother playback experience for the user in terms of quality switching. There is also an opportunity for novel adaptation logics to further improve video adaptation. If the bit-rate variations of all video chunks are known beforehand, which can be the case for DASH-based video streaming, a stable goodput allows for computing near-optimal adaptation decisions.

7.7.6 QoE Fairness

To the best of our knowledge, there is no fairness measure to quantify the fairness for different application types with orthogonal resource demands, e.g., throughput-sensitive and



(a) Initial Stalling (*VoD*)

(b) Initial Stalling (*Live*)

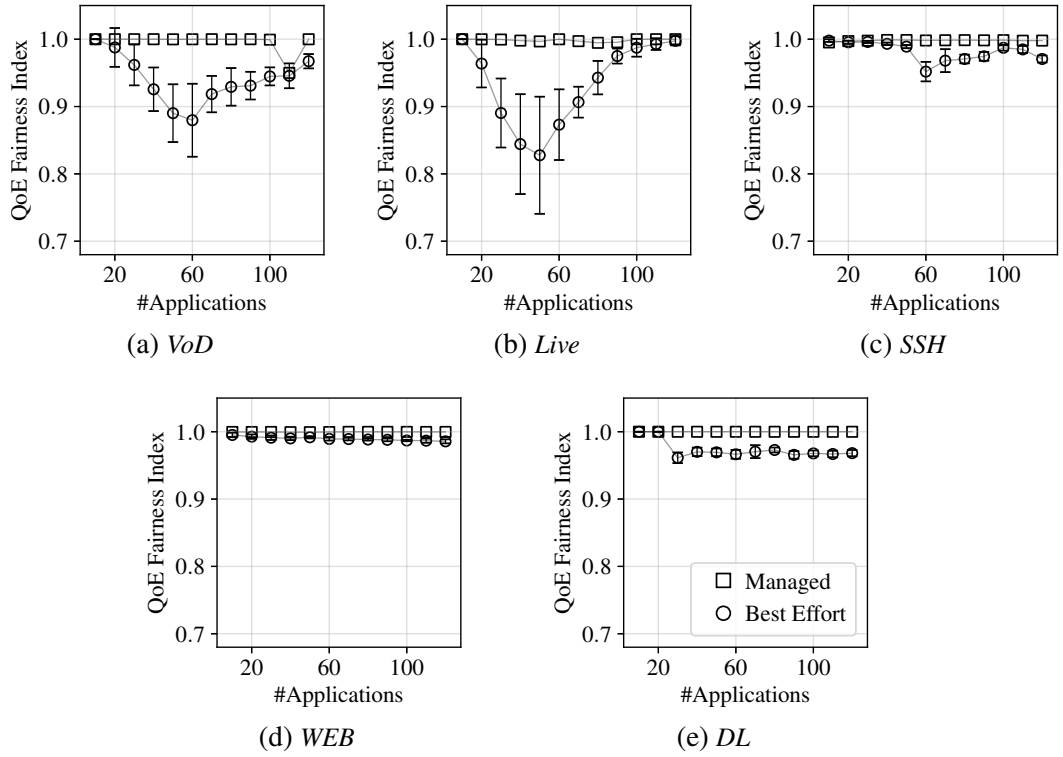
(c) Switches (*VoD*)

(d) Switches (*Live*)

Figure 7.15: *Initial stalling* duration and *switching frequency* for the *VoD* and *Live* clients for increasing number of total applications. There are no recorded stallings during the playback sessions. The results in terms of benefit of the pacing for video streaming clients show unstable adaptation behavior. There is an opportunity for novel quality adaptation logics to benefit from the stable throughout.

delay-sensitive demands. For example, *VoIP* is in our set-up always close to a utility of 5.0, independent of other applications. Hence, any fairness measure which considers only differences between values will consider this as unfair. But enforcing equal utility for all application types, including artificially restricting *VoIP*, would result in a non-Pareto-optimal utility distribution where the target utility of *VoIP* could be increased without negatively impacting other applications. Therefore, we evaluate the inter-application fairness per application type. Note that for the evaluation we are restricting the allocation formulation to allocate only one target utility value per application type. Hence the target utilities per type exhibit always perfect fairness and are omitted.

We evaluate the inter-application fairness using the F-index [84] defined by $F = 1 - \frac{2\sigma}{4}$ for a utility scale of 1 to 5. The F-index is selected as fairness measure as it is specifically designed and evaluated for user experience fairness. An F-index of 1.0 indicates perfect fairness between the applications. An F-index of 0.0 is the result of half of the application experiencing a utility of 1.0 and the other half an utility of 5.0. Figures 7.16a to 7.16e illustrate the F-index



(a) *VoD* (b) *Live* (c) *SSH*

(d) *WEB* (e) *DL*

Figure 7.16: Comparison of the F-index between managed (□) and best effort (○) scenarios per application type for increasing number of applications sharing the constrained link. An F-index of 1.0 denotes perfect fairness. All managed scenarios exhibit nearly perfect fairness, while best effort scenarios show unfairness when applications compete for the link.

per application type t for $|\mathcal{A}| = \{10, 20, .., 120\}$ applications sharing the constrained link, measured for the best effort and managed scenarios. From the figures, we conclude that in the managed case, the F-index does not drop below 0.98 for any of the evaluated scenarios and application types.

In the best effort case, the fairness depends strongly on the application type and the number of parallel applications. The *WEB* clients exhibit a fairness similar to that in the managed scenario (≥ 0.98). For *SSH* and *DL*, the fairness fluctuations are larger, but in general the fairness is still high (≥ 0.95). The two video streaming types *VoD* and *Live* suffer the most in the best effort scenarios. For *Live*, the fairness drops down to 0.7 for $|\mathcal{A}| = 44$ and for *VoD* down to 0.77 for $|\mathcal{A}| = 55$. However, for video streaming there is a high level of fairness for $|\mathcal{A}| < 30$ and $|\mathcal{A}| > 100$. This is due to the fact that for low number of parallel applications, there is sufficient capacity for all clients to reach close to maximum utility while for a high number of parallel applications all clients are close to a utility value of 1.0.

In summary, the evaluation of the fairness per application type shows that *VoD* and *Live* profit the most from the management. *SSH* and *DL* show some improvement. *WEB* and *VoIP*

improve only marginally. In the managed measurements, we observe nearly perfect fairness for all application types.

7.7.7 Summary

The evaluation set out to discuss the following four subjects: i) comparison of minimum and average utility for managed and best effort scenarios, ii) advantages and disadvantages of central data-rate management for each application class, iii) predictability of application performance, and iv) fairness between the applications.

First, a scenario with 80 applications sharing a 100 Mbps link is presented. The measurements show that for the best effort case, web browsing consumes about four times more of the available throughput than the other applications. This is due to web browsers using multiple parallel TCP connections. As a consequence, the utility of the web browsing sessions is high (3.5 to 4.0), while other applications like live video streaming suffer (≤ 2). Next, the allocation formulation is solved for the 80 applications and pacing is applied to the applications. The results show that video streaming, remote terminal work, and file download can increase their utility by 1 to 3 while web's utility is only decreased by 1. The measurements for 10 to 120 parallel applications sharing the link support the findings of the 80 applications scenario. The evaluation of the fairness shows that in the managed scenarios, the application types exhibit close to perfect fairness. For the best effort scenarios, the fairness results show that the two video streaming intents profit the most from the management, followed by *DL* and *SSH*. The *WEB* clients do not profit much from the management in terms of fairness.

In summary, the results show that there is a significant benefit of centrally controlled application pacing via the NSAL in terms of utility, inter-application fairness, and predictability. Furthermore, compared to classical QoS measures in the network, the approach can be implemented with heterogeneous forwarding devices without any special features, it does not require expensive switch buffer space, and it is fully software-based.

7.8 Conclusion

In this chapter we propose an NSAL extension for the resource allocation in heterogeneous networks based on central software-defined network control, fine-grained per-application pacing at the end-hosts, and utility functions derived from measurements and user-experience models. Traditional methods of QoS control in the network, such as policing or scheduling, interact badly with end-host congestion control (Section 3.3) and do not scale to larger number of applications and application classes. Moving application pacing from in-network QoS methods to the end hosts, e.g., to user PCs, servers, smartphones, and tablets, is scalable,

increases transmission efficiency, reduces the required complexity of forwarding devices, and allows cost-efficient high link utilizations. To the best of our knowledge, this is the first work proposing, formulating, and evaluating a scalable NSAL architecture for resource allocation for end-user applications in enterprise environments based on real applications and user-experience models.

We define application- and user-level utility using selected user-experience models from the literature. Based on the models, we derive per-application utility models for the five common network use cases web browsing, file download, remote terminal, adaptive video streaming, and Voice-over-IP. Afterwards, we determine sensible resource allocations by formulating a two-stage mixed-integer linear program based on the number and types of applications, their utility functions, and network resources. The mixed-integer linear program decides on how to embed the applications in the network in terms of the allowed data-rate per application and the delay-constrained routing of the application flows. Once the allowed rate and routing is determined, the flow routing is configured through the NSAL and the pacing is enforced through local agents at the end-hosts.

We evaluate the methodology by implementing an experiment set-up with a throughput-constrained link and an increasing number of parallel applications sharing the link. The results show that QoS metrics, such as delay and packet loss of the link, considerable improve with pacing, due to the controlled link utilization. When looking at the fairness per application type, the results show that there is near perfect fairness between the clients. For the five evaluated application types, the results show that web browsing's utility decreases, as it has an unfair advantage in the best effort case due to its multiple parallel TCP-connections. However, the loss in utility of web browsing is low, compared to the gain for the other types. Real-time applications, such as remote terminal work and VoIP, profit due to the reduced delay and packet loss. From the experiments, we conclude that the proposed architecture enables scalable resource allocation and predictable application performance. This chapter is a step towards extending the NSAL towards the edge of the network with scalable resource allocations from the perspective of the human users.

Chapter 8

Conclusions and Outlook

Traditional forwarding devices are closed boxes with proprietary Command Line Interfaces (CLIs) and no standardized way for the discovery of supported features. Hence, network management often involves human experts writing device-specific snippets of code. As a result, evolving an existing network towards more efficient operation or implementing new use cases is slow and cumbersome. Furthermore, studies show that human errors account for 50 % to 80 % of network outages [98] and that precautionary measures such as redundant links are broken by too complex configurations [72].

Software-Defined Networking (SDN) disaggregates the devices by removing the control-plane from the individual devices and logically centralizing the forwarding decision into the SDN controller. Through an standardized interface and protocol, e.g., OpenFlow, the controller can add and modify forwarding rules on the devices. Research and production experience of SDN by large Internet companies shows that SDN enables fast adaptation to changes in the network and increased network utilization for data-centers and wide-area networks [92].

Techno-economical considerations force most network operators to migrate to SDN incrementally, instead of replacing all devices at once. Therefore, approaches are needed for how to operate traditional and SDN devices side-by-side in the same network. This thesis investigates three main aspects of the migration towards fully deployed SDNs. First, the design of an abstraction layer which can combine control and management of the traditional and SDN devices in the network and offer a unified view to control applications. Second, modeling the performance of such an abstraction layer. On the one side from an abstract perspective which considers the diversity of devices in the network and their different reconfiguration timing characteristics. And on the other side from a deployment perspective where the abstraction layer is implemented as a software component in virtualized environments. Third, enabling Quality of Service (QoS) and resource allocation for applications and general application-awareness through the abstraction layer despite the diverse set of devices with different feature sets.

8.1 Summary and Discussion

Next, we summarize and discuss the challenges, results and contributions of this thesis.

Background on Software-Defined Networking and Hybrid Networks (Chapter 2)

The thesis first elaborates on SDN in general, on the technical implementation via OpenFlow and on the evolution of SDN from laboratories to comprehensive Network Services Abstraction Layers (NSALs). Furthermore, three popular migration strategies are discussed in detail and the state of the art of hybrid networking is introduced. The analyses of the state of the art reveals that there is a lack of research on how to manage hybrid networks, specifically concerning QoS configuration and features discovery.

Measurements (Chapter 3)

Measurements of hardware and software components are conducted as part of this thesis. The goal of the measurements is to better understand the flexibility of forwarding devices in terms of timings characteristics and effects of management actions. Especially, we measure the timing of configuring Virtual Local Area Networks (VLANs), a popular approach for combining traditional and SDN routing. The measurement results show that VLAN reconfiguration on traditional devices is by more than two orders of magnitude slower and than on SDN devices (s. 600 ms vs. 1 ms). Furthermore, we measure QoS configuration on traditional and SDN devices. There the results show that changing the QoS packet processing can result in data-plane interruptions of up to 6 s. We also measure the impact of configuring traffic policing on Transmission Control Protocol (TCP) and on applications. The results highlight the negative impact of in-network policing on the data-plane. Congestion control algorithm show an degradation of performance due to the dropped packets of up to one sixth of the expected performance. For the measured video streaming use case, the quality adaptation algorithm can not reliably determine the available throughput and decide on a suitable quality level. As a consequence, quality switches and re-buffering events can be observed.

NSAL Design, Challenges and Trade-Offs (Chapter 4)

Chapter 4 proposes an NSAL architecture that consists of an extended network graph that provides a hardware detail-level that includes the QoS processing pipelines of devices. Via a northbound interface, control applications can run networking tasks through the vendor- and device-neutral NSAL. By example of Panopticon, an approach for OpenFlow networking in mixed-SDN/traditional networks, we show how a control application on top of the NSAL can run configuration tasks. These configuration tasks include, e.g., VLAN tagging on non-SDN devices, discovering of QoS options in the network, or changing the packet scheduling. Furthermore, the architecture provides mechanisms to estimate traffic interruptions due to

the triggered management actions, e.g., due to the change of the scheduler configurations. By example, we show how a smart task scheduling based on the known timings of the management actions mitigates service interruptions. Besides, the proposed NSAL enables novel management applications capable of autonomous network management.

Performance Modeling (Chapter 5 and 6)

The results of Chapter 5 show that even a small number of inflexible traditional devices severely reduce the maximum reconfiguration rate of the network. Even at higher deployment ratios, the global reconfiguration rate does not increase by more than factor five compared to the all-traditional deployment. Therefore the results raise the question how advanced SDN use cases, which require timely reconfigurations, can be realized in networks with only partial SDN deployment. This is a gap in current research and has to be investigated further to enable a smooth and beneficial migration phase of existing networks to SDN-enabled networks.

Chapter 6 proposes and evaluates an online machine learning pipeline for the capacity estimation of NSAL instances in dynamic cloud environments. The evaluation shows that the learned performance model provides accurate estimations of the control message rate budget at run-time. Furthermore, a reduction or increase of available compute resources assigned to the NSAL is detected by the pipeline and the estimations are adapted accordingly. The proposed pipeline is an important step towards autonomous scaling and load-balancing of virtualized NSAL instances..

Application-Awareness (Chapter 7)

In Chapter 7, we propose a methodology for application control through the NSAL based on fine-grained per-application pacing at the end-hosts and utility functions derived from measurements and user-experience models. Traditional methods of QoS control in the network, such as policing or scheduling, require expensive buffer space and support for specific features by the switching Application-Specific Integrated Circuit (ASIC), have complex configurations, interact badly with end-host congestion control and do not scale to larger number of applications and application classes. We evaluate the methodology by implementing a proof-of-concept testbed with a throughput-constrained link and increasing number of parallel applications sharing the link. The results show that QoS metrics such as delay and packet loss of the link considerable improve with pacing due to the controlled link utilization. Furthermore, when looking at the fairness in terms of utility, the results show that the minimum and total utility over all applications is significantly improved. From the experiments, we conclude that the proposed extension to the NSAL enables a fair resource allocation and predictable application performance in hybrid networks.

8.2 Future Work

The operation and management of hybrid networks is still an active area of research and the results presented in this thesis suggest the following further directions of research.

NSAL Design

The NSAL itself should be extended towards a logically centralized, but physically distributed design. Furthermore, the design should be evaluated against a larger set of use cases to confirm or refute the choice of the level of abstraction. Some use cases may also require a more dynamic level of abstraction where additional details can be made available to control applications on-demand. In general, the amount of required human effort can be further reduced. Analyzing the QoS processing pipeline of a device requires a human expert studying the device's handbook and its interfaces. It remains an open question how this could be automated. First proposals are made how machine learning could be used to derive the features of a device from its CLI [119], but they should be extended towards QoS.

Performance Modeling

The performance modeling of NSALs lacks a model for the distribution and types of reconfigurations in production networks. Future research here should analyze current production deployments and derive models for the types and frequency of reconfigurations. Further research is also required regarding the reconfiguration timings of traditional and SDN devices. Here the focus should be on increasing the number of measured devices to generate an abstract model of the reconfiguration times of different devices and explore the possibility of configuration load-balancing to circumvent slow configuration interfaces of traditional devices. Regarding the machine learning for online performance estimation, future work should investigate the convergence time of the extended model and evaluate the trade-off between Support Vector Machine (SVM) sensitivity and convergence time in more detail.

Application-Aware Resource Allocation

Integrating applications and end-host pacers into the NSAL enables the application-aware resource allocation. To further improve the resource allocation, the end-host pacers should be combined with the available in-network QoS features. Future work in this area should also focus on how to autonomously create and update utility functions and investigating the impact of inaccurate utility functions. Furthermore, a fast heuristics has to be developed for the optimization problem formulation which also enables solving the problem of dynamically embedding applications at run-time.

Bibliography

Publications by the author

Journal publications

[1] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia. “Close to Optimum? User-centric Evaluation of Adaptation Logics for HTTP Adaptive Streaming.” In: *PIK - Praxis der Informationsverarbeitung und Kommunikation, 37.4* 11 pages (2014). DOI: 10.1515/pik-2014-0029.

[2] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia. “Identifying QoE Optimal Adaptation of HTTP Adaptive Streaming Based on Subjective Studies.” In: *ELSEVIER Computer Networks* 81.23 pages (2015). DOI: 10.1016/j.comnet.2015.02.015.

[3] C. Sieber, S. Schwarzmann, A. Blenk, T. Zinner, and W. Kellerer. “Scalable Application- and User-aware Resource Allocation in Enterprise Networks Using End-host Pacing.” In: Under minor revision for *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)* 34 pages (2018). arXiv: 1811.02367.

Patents

[P-NAII] C. Sieber and P. Sharma. *Network Affinity Index Increase (Under Review)*. Application number: 15/611095. July 2017.

Conference publications

[4] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. “Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming.” In: *Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*. 6 pages. Singapore, Sept. 2014. DOI: 10.1109/QoMEX.2014.6982305.

[5] T. Hoßfeld, M. Seufert, and C. Sieber. "Impact of Intermediate Layer on Quality of Experience of HTTP Adaptive Streaming." In: *11th International Conference on Network and Service Management (CNSM)*. 6 pages. Barcelona, Spain, 2015. DOI: 10.1109/CNSM.2015.7367367.

[6] C. Moldovan, K. Hagn, C. Sieber, W. Kellerer, and T. Hoßfeld. "Keep Calm and Don't Switch: About the Relationship Between Switches and Quality in HAS." In: *Modeling Communication Networks Workshop at the International Teletraffic Congress (ITC)*. 6 pages. Stockholm, Sweden, 2017. DOI: 10.23919/ITC.2017.8065802.

[7] C. Moldovan, C. Sieber, P. E. Heegaard, W. Kellerer, and T. Hoßfeld. "YouTube Can Do Better: Getting the Most Out of Video Adaptation." In: *Workshop on QoE Centric Management (QCMan)*. 6 pages. Würzburg, Germany, 2016. DOI: 10.1109/ITC-28.2016.309.

[8] S. Schwarzmann, T. Zinner, C. Sieber, and S. Geissler. "Evaluation of the Benefits of Variable Segment Durations for Adaptive Streaming." In: *QoE Management Workshop at the Tenth International Conference on Quality of Multimedia Experience (QoMEX)* 6 pages (2018).

[9] C. Sieber, A. Blenk, M. Hinteregger, and W. Kellerer. "The Cost of Aggressive HTTP Adaptive Streaming: Quantifying YouTube's Redundant Traffic." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 6 pages. Ottawa, Canada, 2015. DOI: 10.1109/INM.2015.7140478.

[10] C. Sieber, A. Blenk, D. Hock, M. Scheib, T. Hohn, et al. "Network Configuration with Quality of Service Abstractions for SDN and Legacy Networks." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2 pages. Ottawa, Canada, 2015. DOI: 10.1109/INM.2015.7140446.

[11] C. Sieber, A. Basta, A. Blenk, and W. Kellerer. "Online Resource Mapping for SDN Network Hypervisors using Machine Learning." In: *2nd IEEE Conference on Network Softwarization (NetSoft)*. 5 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502447.

[12] C. Sieber, A. Blenk, A. Basta, D. Hock, and W. Kellerer. "Towards a Programmable Management Plane for SDN and Legacy Networks." In: *IEEE Conference on Network Softwarization (NetSoft)*. 9 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502428.

[13] C. Sieber, A. Blenk, A. Basta, and W. Kellerer. “hvbench: An open and scalable SDN network hypervisor benchmark.” In: *IEEE NetSoft Conference and Workshops: Software-Defined Infrastructure for Networks, Clouds, IoT and Services*. 4 pages. Seoul, South Korea, 2016. DOI: 10.1109/NETSOFT.2016.7502475.

[14] C. Sieber, R. Durner, M. Ehm, W. Kellerer, and P. Sharma. “Towards Optimal Adaptation of NFV Packet Processing to Modern CPU Memory Architectures.” In: *Workshop on Cloud-Assisted Networking (CAN) at the 13th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 6 pages. ACM. Seoul, South Korea, 2017. DOI: 10.1145/3155921.3158429.

[15] C. Sieber, R. Durner, and W. Kellerer. “How fast can you reconfigure your partially deployed SDN network?” In: *IFIP Networking Conference*. 9 pages. Stockholm, Sweden, 2017, p. 9. DOI: 10.23919/IFIPNetworking.2017.8264845.

[16] C. Sieber, P. E. Heegaard, T. Hoßfeld, and W. Kellerer. “Sacrificing Efficiency for Quality of Experience: YouTube’s Redundant Traffic Behavior.” In: *IFIP Networking Conference*. 9 pages. Vienna, Austria, 2016. DOI: 10.1109/IFIPNetworking.2016.7497231.

[17] C. Sieber, T. Hoßfeld, T. Zinner, P. Tran-Gia, and C. Timmerer. “Implementation and User-centric Comparison of a Novel Adaptation Logic for DASH with SVC.” In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 6 pages. Ghent, Belgium, 2013.

[18] C. Sieber, A. Obermair, and W. Kellerer. “Online learning and adaptation of network hypervisor performance models.” In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 9 pages. Lisbon, Portugal, 2017. DOI: 10.23919/INM.2017.7987462.

Technical Reports

[19] T. Hoßfeld, M. Seufert, T. Zinner, and C. Sieber. “Crowdsourced Subjective User Study Results on QoE Influence Factors of HTTP Adaptive Streaming.” In: *Technical Report 491, Lehrstuhl for Informatik III, University of Würzburg* 26 pages (2014).

[20] C. Sieber, K. Hagn, C. Moldovan, T. Hoßfeld, and W. Kellerer. “Towards Machine Learning-Based Optimal HAS.” In: 9 pages (Aug. 2018). arXiv: 1808.08065. URL: http://arxiv.org/abs/1808.08065.

Request-For-Comments (RFCs)

[RFC7426] S. Denazis, E. Haleplidis, J. H. Salim, O. Koufopavlou, D. Meyer, et al. *Software-Defined Networking (SDN): Layers and Architecture Terminology*. 2015. DOI: 10.17487/rfc7426.

[RFC5810] W. Wang, R. Haas, J. H. Salim, A. Doria, and H. M. Khosravi. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. 2010. DOI: 10.17487/rfc5810.

General publications

[21] I. Abdeljaouad, H. Rachidi, S. Fernandes, and A. Karmouch. "Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno." In: *25th Biennial Symposium on Communications*. IEEE. 2010, pp. 80–83. DOI: 10.1109/BSC.2010.5472999.

[22] S. Agarwal, M. Kodialam, and T. V. Lakshman. "Traffic engineering in software defined networks." In: *Proc. of IEEE INFOCOM*. IEEE. 2013, pp. 2211–2219. DOI: 10.1109/INFCOM.2013.6567024.

[23] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, et al. "Data center TCP (DCTCP)." In: *Proc. of ACM SIGCOMM conference*. 2010, pp. 63–74. DOI: 10.1145/1851275.1851192.

[24] R. Amin, N. Shah, B. Shah, and O. Alfandi. "Auto-Configuration of ACL Policy in Case of Topology Change in Hybrid SDN." In: *IEEE Access* 4 (2016), pp. 9437–9450. DOI: 10.1109/ACCESS.2016.2641482.

[25] H. Ballani and P. Francis. "CONMan: a step towards network manageability." In: *ACM SIGCOMM Computer Communication Review*. Vol. 37. 4. 2007, pp. 205–216.

[26] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, et al. "Data Center Network Virtualization: A Survey." In: *IEEE Communications Surveys & Tutorials* 15.2 (2013), pp. 909–928. DOI: 10.1109/SURV.2012.090512.00043.

[27] B. Belter, D. Parniewicz, L. Ogrodowczyk, A. Binczewski, M. Stroinski, et al. "Hardware abstraction layer as an SDN-enabler for non-OpenFlow network equipment." In: *Proc. of IEEE 3rd European Workshop on Software-Defined Networks (EWSDN)*. 2014, pp. 117–118. DOI: 10.1109/EWSDN.2014.16.

[28] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, et al. “ONOS: towards an open, distributed SDN OS.” In: *Proc. of 3rd ACM SIGCOMM Workshop on Hot Topics in Software-Defined Networking (HotSDN)*. CM. 2014, pp. 1–6. DOI: 10.1145/2620728.2620744.

[29] A. Blenk, A. Basta, J. Zerwas, and W. Kellerer. “Pairing SDN with network virtualization: The network hypervisor placement problem.” In: *Proc. of IEEE Network Function Virtualization and Software Defined Network (NFV-SDN)*. 2015. DOI: 10.1109/NFV-SDN.2015.7387427.

[30] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer. “Control Plane Latency with SDN Network Hypervisors: The Cost of Virtualization.” In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 366–380. DOI: 10.1109/TNSM.2016.2587900.

[31] P. T. Boggs, R. H. Byrd, and R. B. Schnabel. “A stable and efficient algorithm for nonlinear orthogonal distance regression.” In: *SIAM Journal on Scientific and Statistical Computing* 8.6 (1987), pp. 1052–1078. DOI: 10.1137/0908085.

[32] T. Bonald, L. Massoulié, A. Proutiere, and J. Virtamo. “A queueing analysis of max-min fairness, proportional fairness and balanced fairness.” In: *Springer Queueing systems* 53.1-2 (2006), pp. 65–84. DOI: 10.1007/s11134-006-7587-7.

[33] T. Bonald and J. Roberts. “Internet and the Erlang formula.” In: *ACM Computer Communication Review* 42.1 (2012), pp. 23–30. DOI: 10.1145/2096149.2096153.

[34] A. Botta, A. Dainotti, and A. Pescapè. “A tool for the generation of realistic network workload for emerging networking scenarios.” In: *Elsevier Computer Networks* 56.15 (2012). DOI: 10.1016/j.comnet.2012.02.019.

[35] Z. Cai, A. L. Cox, and T. S. Ng. *Maestro: A system for scalable openflow control*. Tech. rep. 2010.

[36] M. Canini, A. Feldmann, D. Levin, F. Schaffert, and S. Schmid. “Software-defined networks: Incremental deployment with Panopticon.” In: *IEEE Computer* 47.11 (2014), pp. 56–60.

[37] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. “BBR: Congestion-Based Congestion Control.” In: *ACM Queue* 14.5 (2016), p. 50. DOI: 10.1145/3012426.3022184.

[38] M. Caria, T. Das, A. Jukan, and M. Hoffmann. "Divide and conquer: Partitioning OSPF networks with SDN." In: *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 467–474. DOI: 10.1109/INM.2015.7140324.

[39] M. Caria and A. Jukan. "Link Capacity Planning for Fault Tolerant Operation in Hybrid SDN/OSPF Networks." In: *IEEE Global Communications Conference (GLOBECOM)*. 2016. DOI: 10.1109/GLOCOM.2016.7841957.

[40] M. Caria and A. Jukan. *On the IP Traffic Matrix Problem in Hybrid SDN/OSPF Networks*. Tech. rep. 2016. arXiv: 1610.08256. URL: http://arxiv.org/abs/1610.08256.

[41] M. Caria, A. Jukan, and M. Hoffmann. "A performance study of network migration to SDN-enabled Traffic Engineering." In: *Proc. of IEEE Global Communications Conference (GLOBECOM)*. 2013, pp. 1391–1396. DOI: 10.1109/GLOCOM.2013.6831268.

[42] M. Caria, A. Jukan, and M. Hoffmann. "SDN Partitioning: A Centralized Control Plane for Distributed Routing Protocols." In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 381–393. DOI: 10.1109/TNSM.2016.2585759.

[43] P. Casas, M. Seufert, S. Egger, and R. Schatz. "Quality of experience in remote virtual desktop services." In: *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2013. ISBN: 978-1-4673-5229-1.

[44] D. J. Casey and B. E. Mullins. "SDN shim: Controlling legacy devices." In: *Proc. of IEEE 40th Conference on Local Computer Networks (LCN)*. 2015, pp. 169–172. DOI: 10.1109/LCN.2015.7366298.

[45] M. A. Chang, L. Vanbever, and M. Happe. "Supercharge me: Boost Router Convergence with." In: *Proc. of ACM SIGCOMM (Poster/demo session)*. 2015, pp. 341–342. DOI: 10.1145/2829988.2790007.

[46] V. D. Chemalamarri, P. Nanda, and K. F. Navarro. "SYMPHONY - A Controller Architecture for Hybrid Software Defined Networks." In: *Proc. of IEEE European Workshop on Software Defined Networks (EWSDN)*. 2015, pp. 55–60. DOI: 10.1109/EWSDN.2015.61.

[47] C.-C. Chen, P. Sun, L. Yuan, D. A. Maltz, C.-N. Chuah, et al. "SWIM: A Switch Manager for Datacenter Networks." In: *IEEE Internet Computing* 18.4 (2014), pp. 30–36. DOI: 10.1109/MIC.2014.41.

[48] G. Chen, G. Hu, Y. Jiang, and C. Zhang. “SAVSH: IP source address validation for SDN hybrid networks.” In: *IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2016, pp. 409–414. DOI: 10.1109/ISCC.2016.7543774.

[49] X. Chen, Y. Mao, Z. M. Mao, and J. der Merwe. “Declarative configuration management for complex and dynamic networks.” In: *Proc. of ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 2010, p. 6. DOI: 10.1145/1921168.1921176.

[50] X. Chen, Z. M. Mao, and J. der Merwe. “PACMAN: A Platform for Automated and Controlled Network Operations and Configuration Management.” In: *Proc. of ACM CoNEXT*. 2009, pp. 277–288. ISBN: 978-1-60558-636-6. DOI: 10.1145/1658939.1658971.

[51] C. Y. Chu, K. Xi, M. Luo, and H. J. Chao. “Congestion-aware single link failure recovery in hybrid SDN networks.” In: *Proc. of IEEE Conference on Computer Communications (INFOCOM)*. Vol. 26. 2015, pp. 1086–1094. DOI: 10.1109/INFOCOM.2015.7218482.

[52] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, et al. “DevoFlow: scaling flow management for high-performance networks.” In: *Proc. of ACM SIGCOMM conference*. 2011, pp. 254–265. DOI: 10.1145/2043164.2018466.

[53] T. Das, M. Caria, A. Jukan, and M. Hoffmann. “A Techno-economic Analysis of Network Migration to Software-Defined Networking.” In: (2013). arXiv: 1310.0216.

[54] T. Das, M. Caria, A. Jukan, and M. Hoffmann. “Insights on SDN migration trajectory.” In: *Proc. of IEEE International Conference on Communications (ICC)*. 2015, pp. 5348–5353. DOI: 10.1109/ICC.2015.7249174.

[55] W. Dawoud, I. Takouna, and C. Meinel. “Dynamic scalability and contention prediction in public infrastructure using internet application profiling.” In: *Proc. of IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*. 2012, pp. 208–216.

[56] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. “TAPAS - Tool for rApid Prototyping of Adaptive Streaming algorithms.” In: *Proc. of ACM Workshop on Design, Quality and Deployment of Adaptive Video Streaming*. 2014. DOI: 10.1145/2676652.2676654.

[57] R. Durner, C. Sieber, and W. Kellerer. “Towards reducing Last-Level-Cache Interference of co-located Virtual Network Functions.” In: *Submitted to IEEE Conference on Network Softwarization (NETSOFT)*. 9 pages. 2019.

[58] S. Egger, P. Reichl, T. Hoßfeld, and R. Schatz. ““Time is bandwidth”? Narrowing the gap between subjective time perception and Quality of Experience.” In: *Proc. of IEEE International Conference on Communications (ICC)*. 2012, pp. 1325–1330. DOI: 10.1109/ICC.2012.6363769.

[59] D. Erickson. “The beacon openflow controller.” In: *Proc. of 2nd ACM SIGCOMM Workshop on Hot Topics in Software-Defined Networking (HotSDN)*. 2013, pp. 13–18. DOI: 10.1145/2491185.2491189.

[60] F. Farias, J. Salvatti, P. Victor, and A. Abelem. “Integrating Legacy Forwarding Environment to OpenFlow/SDN Control Plane.” In: *Proc. of IEICE Network Operations and Management Symposium (APNOMS)*. 2013. DOI: 10.13140/2.1.4455.1684.

[61] N. Feamster, J. Rexford, and E. Zegura. “The road to SDN: an intellectual history of programmable networks.” In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 87–98. DOI: 10.1145/2602204.2602219.

[62] Z. Fei, C. Xing, and N. Li. “QoE-driven resource allocation for mobile IP services in wireless network.” In: *Springer Science China Information Sciences* 58.1 (2015), pp. 1–10.

[63] T. Feng and J. Bi. “OpenRouteFlow: Enable legacy router as a software-defined routing service for hybrid SDN.” In: *Proc. of IEEE International Conference on Computer Communications and Networks (ICCCN)*. 2015. DOI: 10.1109/ICCCN.2015.7288441.

[64] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. “Participatory networking: an API for application control of SDNs.” In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. 2013, pp. 327–338.

[65] A. D. Ferguson, A. Guha, J. Place, R. Fonseca, and S. Krishnamurthi. “Participatory Networking.” In: *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*. 2012.

[66] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, et al. “An Internet-Wide Analysis of Traffic Policing.” In: *Proc. of ACM SIGCOMM conference*. 2016, pp. 468–482. DOI: 10.1145/2934872.2934873.

[67] L. C. Freeman. “A Set of Measures of Centrality Based on Betweenness.” In: *JSTOR Sociometry* 40.1 (1977), pp. 35–41. DOI: 10.2307/3033543.

[68] V. Fuentes, J. Matias, A. Mendiola, M. Huarte, J. Unzilla, et al. “Integrating complex legacy systems under OpenFlow control: The DOCSIS use case.” In: *Proc. of IEEE 3rd European Workshop on Software-Defined Networks (EWSDN)*. 2014, pp. 37–42. DOI: 10.1109/EWSDN.2014.35.

[69] A. Gämperli, V. Kotronis, and X. Dimitropoulos. “Evaluating the Effect of Centralization on Routing Convergence on a Hybrid BGP-SDN Emulation Framework.” In: *Proc. of ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. 2014, pp. 369–370. arXiv: 1611.03113.

[70] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, et al. “Predicting multiple metrics for queries: Better decisions enabled by machine learning.” In: *Proc. of IEEE 25th International Conference on Data Engineering*. 2009, pp. 592–603. DOI: 10.1109/ICDE.2009.130.

[71] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. “Towards network-wide QoE fairness using openflow-assisted adaptive video streaming.” In: *Proc. of ACM SIGCOMM Workshop on Future human-centric Multimedia Networking (FhMN)*. 2013, pp. 15–20. DOI: 10.1145/2491172.2491181.

[72] P. Gill, N. Jain, and N. Nagappan. “Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications.” In: *Proc. of ACM SIGCOMM conference*. New York, NY, USA: ACM, 2011, pp. 350–361. ISBN: 978-1-4503-0797-0.

[73] G. Gómez, J. Lorca, R. García, and Q. Pérez. “Towards a QoE-driven resource control in LTE and LTE-A networks.” In: *Journal of Computer Networks and Communications* Article ID 505910 (2013). DOI: 10.1155/2013/505910.

[74] Z. Gong, X. Gu, and J. Wilkes. “PRESS: PRedictive Elastic ReSource Scaling for cloud systems.” In: *Proc. of IEEE International Conference on Network and Service Management (CNSM)*. 2010. DOI: 10.1109/CNSM.2010.5691343.

[75] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, et al. “NOX: towards an operating system for networks.” In: *ACM SIGCOMM Computer Communication Review* 38.3 (2008), pp. 105–110. DOI: 10.1145/1384609.1384625.

[76] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu. “Traffic engineering in SDN/OSPF hybrid network.” In: *Proc. of IEEE International Conference on Network Protocols (ICNP 2014)*. 2014, pp. 563–568. DOI: 10.1109/ICNP.2014.90.

[77] Y. Guo, Z. Wang, X. Yin, X. Shi, J. Wu, et al. "Incremental deployment for traffic engineering in hybrid SDN network." In: *Proc. of IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. 2015. DOI: 10.1109/PCCC.2015.7410320.

[78] S. Ha, I. Rhee, and L. Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." In: *ACM SIGOPS Operating Systems Review* 42.5 (2008), pp. 64–74. DOI: 10.1145/1400097.1400105.

[79] J. He and W. Song. "Achieving near-optimal traffic engineering in hybrid Software Defined Networks." In: *Proc. of 14th IFIP Networking Conference*. 2015. DOI: 10.1109/IFIPNetworking.2015.7145321.

[80] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, et al. "Measuring control plane latency in SDN-enabled switches." In: *Proc. of 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*. 2015. DOI: 10.1145/2774993.2775069.

[81] L. He, X. Zhang, Y. Jiang, and I. Engineering. "Design and Implementation of SDN/IP Hybrid Space Information Network Prototype." In: *Proc. of IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. 2016. DOI: 10.1109/ICCChinaW.2016.7586705.

[82] M. Hock, R. Bless, and M. Zitterbart. "Experimental evaluation of BBR congestion control." In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE. 2017. DOI: 10.1109/ICNP.2017.8117540.

[83] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao. "Incremental Deployment of SDN in Hybrid Enterprise and ISP Networks." In: *Proc. of 2nd ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*. 2016. DOI: 10.1145/2890955.2890959.

[84] T. Hoßfeld, L. Skorin-Kapov, P. E. Heegaard, and M. Varela. "Definition of QoE Fairness in Shared Systems." In: *IEEE Communications Letters* 21.1 (2017), pp. 184–187. DOI: 10.1109/LCOMM.2016.2616342.

[85] T. C. Hu. "Multi-commodity network flows." In: *Operations research* 11.3 (1963), pp. 344–360.

[86] Y. Hu, W. Wang, X. Gong, X. Que, Y. Ma, et al. "Maximizing Network Utilization in Hybrid Software-Defined Networks." In: *Proc. of IEEE Global Communications Conference (GLOBECOM)*. 2015. DOI: 10.1109/GLOCOM.2014.7417144.

[87] M. Huang and W. Liang. “Incremental SDN-Enabled Switch Deployment for Hybrid Software-Defined Networks.” In: *Proc of. IEEE 26th International Conference on Computer Communication and Networks (ICCCN)*. 2017. DOI: 10.1109/ICCCN.2017.8038498.

[88] S. Huang, J. Zhao, and X. Wang. “HybridFlow : A Lightweight Control Plane for Hybrid SDN in Enterprise Networks.” In: *Proc. of IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. 2016. DOI: 10.1109/IWQoS.2016.7590411.

[89] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, et al. “A Survey of Deployment Solutions and Optimization Strategies for Hybrid SDN Networks.” In: *IEEE Communications Surveys & Tutorials* (2018). DOI: 10.1109/COMST.2018.2871061.

[90] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. “Performance Anomaly Detection and Bottleneck Identification.” In: *ACM Computing Surveys (CSUR)* 48.1 (2015), p. 4. DOI: 10.1145/2791120.

[91] R. Jain and S. Paul. “Network virtualization and software defined networking for cloud computing: a survey.” In: *IEEE Communications Magazine* 51.11 (2013), pp. 24–31. DOI: 10.1109/MCOM.2013.6658648.

[92] S. Jain, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, et al. “B4: Experience with a globally-deployed software defined WAN.” In: *Prof. of ACM SIGCOMM conference*. 2013, pp. 3–14. DOI: 10.1145/2486001.2486019.

[93] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, et al. “Modeling and performance evaluation of an OpenFlow architecture.” In: *Proc. of 23rd International Teletraffic Congress (ITC)*. 2011. ISBN: 978-0-9836283-0-9.

[94] X. Jia, Y. Jiang, and Z. Guo. “Incremental Switch Deployment for Hybrid Software-Defined Networks.” In: *Proc. of IEEE 41st Conference on Local Computer Networks (LCN)*. 2016, pp. 571–574. DOI: 10.1109/LCN.2016.95.

[95] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang. “Telekinesis: Controlling Legacy Switch Routing with OpenFlow in Hybrid Networks.” In: *Proc. of 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*. 2015. DOI: 10.1145/2774993.2775013.

[96] C. Jin, C. Lumezanu†, Q. Xu, H. Mekky, Z.-L. Zhang, et al. “Exerting Fine-Grained Path Control over Legacy Switches in Hybrid Networks.” In: *TR 16-035* January (2016), pp. 81–105. DOI: 10.1089/lap.2006.05083.

[97] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, et al. "Dynamic scheduling of network updates." In: *Proc. of ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. 2014.

[98] Juniper Networks. *What's behind network downtime? Proactive Steps to Reduce Human Error and Improve Availability of Networks*. Tech. rep. 2008. URL: https://www-935.ibm.com/services/au/gts/pdf/200249.pdf.

[99] Y. Kanaumi, S.-i. Saito, E. Kawai, S. Ishii, K. Kobayashi, et al. "RISE: A Wide-Area Hybrid OpenFlow Network Testbed." In: *IEICE Transactions on Communications* E96.B.1 (2013), pp. 108–118. DOI: 10.1587/transcom.E96.B.108.

[100] B. Kar, E. H.-k. Wu, and Y.-d. Lin. "The Budgeted Maximum Coverage Problem in Partially Deployed Software Defined Networks." In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 394–406. DOI: 10.1109/TNSM.2016.2598549.

[101] R. Katiyar, P. Pawar, A. Gupta, and K. Kataoka. "Auto-Configuration of SDN Switches in SDN/Non-SDN Hybrid Network." In: *Proc. of ACM Asian Internet Engineering Conference (AINTEC)*. 2015, pp. 48–53. DOI: 10.1145/2837030.2837037.

[102] H. Kim, T. Benson, A. Akella, and N. Feamster. "The evolution of network configuration: a tale of two campuses." In: *Proc. of ACM SIGCOMM IMC*. 2011, pp. 499–514.

[103] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, et al. "Network Virtualization in Multi-tenant Datacenters." In: *Proc. of USENIX 11th Symposium on Networked Systems Design and Implementation (NSDI)*. 2014, pp. 203–216. ISBN: 978-1-931971-09-6.

[104] D. Kreutz, F. M. V. Ramos, et al. "Software-Defined Networking: A Comprehensive Survey." In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. DOI: 10.1109/JPROC.2014.2371999.

[105] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, et al. "BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing." In: *Proc. of ACM SIGCOMM conference*. Vol. 45. 4. 2015. DOI: 10.1145/2785956.2787478.

[106] M. Kuźniar, P. Perešini, and D. Kostić. "What you need to know about SDN flow tables." In: *Proc. of International Conference on Passive and Active Network Measurement (PAM)*. 2015. DOI: 10.1007/978-3-319-15509-8_26.

[107] J. Kwon, D. Seo, M. Kwon, H. Lee, A. Perrig, et al. “An incrementally deployable anti-spoofing mechanism for software-defined networks.” In: *Elsevier Computer Communications* 64 (2015), pp. 1–20. DOI: 10.1016/j.comcom.2015.03.003.

[108] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, et al. “Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization.” In: *Proc. of ACM 10th International on Conference on emerging Networking Experiments and Technologies (CoNEXT)*. 2014, pp. 199–212. DOI: 10.1145/2674005.2675011.

[109] K. Lee, A. C. König, V. Narasayya, B. Ding, S. Chaudhuri, et al. “Operator and Query Progress Estimation in Microsoft SQL Server Live Query Statistics.” In: *Proc. of ACM International Conference on Management of Data (SIGMOD)*. 2016, pp. 1753–1764. DOI: 10.1145/2882903.2903728.

[110] D. Levin and M. Canini. “Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks.” In: *Proc. of USENIX Annual Technical Conference*. 2014, pp. 333–345. DOI: 10.1109/JPROC.2014.2371999.

[111] D. Levin, S. Schmid, F. Schaffert, M. Canini, and A. Feldmann. *Logical SDNs: Reaping Software-Defined Networking Benefits Through Incremental Deployment*. Tech. rep. 2013.

[112] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, et al. “Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale.” In: *IEEE Journal on Selected Areas in Communications* 32.4 (2014), pp. 719–733. DOI: 10.1109/JSAC.2014.140405.

[113] F. Liu, W. Xiang, Y. Zhang, K. Zheng, and H. Zhao. “A novel QoE-based carrier scheduling scheme in LTE-Advanced networks with multi-service.” In: *Proc. of IEEE Vehicular Technology Conference (VTC Fall)*. 2012. DOI: 10.1109/VTCFall.2012.6398912.

[114] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, et al. “HybNET: network manager for a hybrid network infrastructure.” In: *Proc. of ACM/IFIP/USENIX 13th International Middleware Conference, Industrial Track*. 2013. DOI: 10.1145/2541596.2541602.

[115] T. Lukovszki, M. Rost, S. Schmid, and S. Schmid. “It’s a Match! Near-Optimal and Incremental Middlebox Deployment.” In: *Proc. of ACM SIGCOMM Computer Communication Review*. Vol. 46. 1. 2016, pp. 30–36. DOI: 10.1145/2875951.2875956.

[116] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel. "Modelling of OpenFlow-based software-defined networks: the multiple node case." In: *IET Networks* 4.5 (2015), pp. 278–284. DOI: 10.1049/iet-net.2014.0091.

[117] L. M. Manevitz and M. Yousef. "One-class SVMs for document classification." In: *Journal of Machine Learning Research* 2.Dec (2001), pp. 139–154.

[118] M. Markovitch and S. Schmid. "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes." In: *Proc. of IEEE International Conference on Network Protocols (ICNP)*. 2015, pp. 78–89. DOI: 10.1109/ICNP.2015.47.

[119] A. Martinez, M. Yannuzzi, J. E. L. De Vergara, R. Serral-Gracia, and W. Ramirez. "An Ontology-Based Information Extraction System for bridging the configuration gap in hybrid SDN environments." In: *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 441–449. DOI: 10.1109/INM.2015.7140321.

[120] A. Matsunaga and J. A. B. Fortes. "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications." In: *Proc. of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*. 2010, pp. 495–504. DOI: 10.1109/CCGRID.2010.98.

[121] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, et al. "OpenFlow: Enabling Innovation in Campus Networks." In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), p. 69. DOI: 10.1145/1355734.1355746.

[122] J. Medved, R. Varga, A. Tkacik, and K. Gray. "Opendaylight: Towards a model-driven SDN controller architecture." In: *Proc. of IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 2014. DOI: 10.1109/WoWMoM.2014.6918985.

[123] A. Mishra, D. Bansod, and K. Haribabu. "A Framework for OpenFlow-like Policy-based Routing in Hybrid Software Defined Networks." In: *Proc. of 11th International Network Conference (INC)*. 2016, pp. 97–102.

[124] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, et al. "Virtual routers as a service: the routeflow approach leveraging software-defined networks." In: *Proc. of ACM 6th International Conference on Future Internet Technologies (CFI)*. 2011, pp. 34–37. DOI: 10.1145/2002396.2002405.

[125] T. Nelson, A. D. Ferguson, D. Yu, R. Fonseca, and S. Krishnamurthi. “Exodus: Toward Automatic Migration of Enterprise Network Configurations to SDNs.” In: *Proc. of ACM 1st SIGCOMM Symposium on Software Defined Networking Research (SOSR)*. 2015. DOI: 10.1145/2774993.2774997.

[126] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. “Agile: Elastic distributed resource scaling for infrastructure-as-a-service.” In: *Proc. of USENIX 10th International Conference on Autonomic Computing (ICAC)*. 2013, pp. 69–82.

[127] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. “A survey of software-defined networking: Past, present, and future of programmable networks.” In: *IEEE Communications Surveys & Tutorials* 16.3 (2014), pp. 1617–1634.

[128] L. Ogrodowczyk, B. Belter, A. Binczewski, K. Dombek, A. Juszczyk, et al. “Hardware abstraction layer for non-OpenFlow capable devices.” In: (2014).

[129] D. Parniewicz, B. Belter, E. Jacob, K. Pentikousis, R. Doriguzzi Corin, et al. “Design and implementation of an OpenFlow hardware abstraction layer.” In: *Proc. of ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC)*. 2014. DOI: 10.1145/2627566.2627577.

[130] Y. Peng, L. Guo, Q. Deng, Z. Ning, and L. Zhang. “A novel hybrid routing forwarding algorithm in sdn enabled wireless mesh networks.” In: *Proc. of IEEE High Performance Computing and Communications (HPCC), IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), IEEE 17th International Conference on Embedded Software and Systems (ICESS)*. 2015, pp. 1806–1811. DOI: 10.1109/HPCC-CSS-ICESS.2015.271.

[131] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, et al. “The Design and Implementation of Open vSwitch.” In: *Proc. of USENIX 12th Symposium on Networked Systems Design and Implementation (NSDI)*. 2015.

[132] M. Polverini, A. Iacovazzi, A. Cianfrani, A. Baiocchi, and M. Listanti. “Traffic matrix estimation enhanced by SDNs nodes in real network topology.” In: *Proc. of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2015. DOI: 10.1109/INFCOMW.2015.7179401.

[133] S. Rathee, Y. Sinha, and K. Haribabu. “A Survey: Hybrid SDN.” In: *Elsevier Journal of Network and Computer Applications* 100.December (2017), pp. 35–55. DOI: 10.1016/j.jnca.2017.10.003.

[134] C. E. Rothenberg, A. Vidal, M. R. Salvador, C. Correa, S. Lucena, et al. "Hybrid networking towards a software defined era." In: *Network Innovation through OpenFlow and SDN: Principles and Design*. 2014, pp. 153–198.

[135] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, et al. "Revisiting routing control platforms with the eyes and muscles of software-defined networking." In: *Proc. of 1st Workshop on Hot Topics in Software Defined Networks (HotSDN)*. 2012, pp. 13–18. DOI: 10.1145/2342441.2342445.

[136] C. Sacchi, F. Granelli, and C. Schlegel. "A QoE-Oriented Strategy for OFDMA Radio Resource Allocation Based on Min-MOS Maximization." In: *IEEE Communications Letters* 15.5 (2011), pp. 494–496. DOI: 10.1109/LCOMM.2011.031411.101672.

[137] A. Saeed, N. Dukkipati, V. Valancius, C. Contavalli, A. Vahdat, et al. "Carousel: Scalable Traffic Shaping at End Hosts." In: *Proc. of ACM SIGCOMM conference*. 2017, pp. 404–417. DOI: 10.1145/3098822.3098852.

[138] R. M. Salles and J. A. Barria. "Fair and efficient dynamic bandwidth allocation for multi-application networks." In: *Computer Networks* 49.6 (2005), pp. 856–877. DOI: 10.1016/j.comnet.2004.12.008.

[139] S. Salsano, P. L. Ventre, F. Lombardo, and G. Siracusano. "Hybrid IP/SDN Networking: Open Implementation and Experiment Management Tools." In: *IEEE Transactions on Network and Service Management* 13.December (2015), pp. 138–153. DOI: 10.1109/TNSM.2015.2507622.

[140] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, et al. "OSHI - Open source hybrid IP/SDN networking (and its emulation on mininet and on distributed SDN testbeds)." In: *Proc. of 3rd European Workshop on Software-Defined Networks (EWSDN)*. 1. 2014, pp. 13–18. DOI: 10.1109/EWSDN.2014.38.

[141] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, et al. "OpenVirteX: A Network Hypervisor." In: *Open Networking Summit (ONS)*. 2014.

[142] P. Sharma, S. Banerjee, S. Tandel, R. Aguiar, R. Amorim, et al. "Enhancing network management frameworks with SDN-like control." In: *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE. 2013, pp. 688–691. ISBN: 978-1-4673-5229-1.

[143] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. “Cloudscale: elastic resource scaling for multi-tenant cloud systems.” In: *Proc. of ACM 2nd Symposium on Cloud Computing*. ACM. 2011. DOI: 10.1145/2038916.2038921.

[144] R. Sherwood, G. Gibb, K.-k. Yap, G. Appenzeller, M. Casado, et al. *FlowVisor: A Network Virtualization Layer*. Tech. rep. 2009.

[P-NAII] C. Sieber and P. Sharma. *Network Affinity Index Increase (Under Review)*. Application number: 15/611095. July 2017.

[145] H. Song. “Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane.” In: *Proc. of ACM 2nd SIGCOMM Workshop on Hot Topics in Software-Defined Networking*. 2013, pp. 127–132.

[146] L. Sun and E. C. Ifeachor. “Voice quality prediction models and their application in VoIP networks.” In: *IEEE Transactions on Multimedia* 8.4 (2006), pp. 809–820. DOI: 10.1109/TMM.2006.876279.

[147] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, et al. “A network-state management service.” In: *ACM SIGCOMM Computer Communication Review* 44.4 (2015), pp. 563–574.

[148] X. Sun, Z. Jia, M. Zhao, and Z. Zhang. “Multipath Load Balancing in SDN/OSPF Hybrid Network.” In: *IFIP International Conference on Network and Parallel Computing (NPC), Lecture Notes in Computer Science*. Vol. 9966. 2016. DOI: 10.1007/978-3-319-47099-3_8.

[149] P. Tang, P. Wang, N. Wang, and V. N. Ngoc. “QoE-Based Resource Allocation Algorithm for Multi-Applications in Downlink LTE Systems.” In: *Proc. of International Conference on Computer, Communications and Information Technology (CCIT)*. Atlantis Press, 2014, pp. 1011–1016.

[150] X. Tu, X. Li, J. Zhou, and S. Chen. “Splicing MPLS and OpenFlow Tunnels Based on SDN Paradigm.” In: *Proc. of IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2014, pp. 489–493. DOI: 10.1109/IC2E.2014.20.

[151] L. Vanbever and S. Vissicchio. “Enabling SDN in Old School Networks with Software-Controlled Routing Protocols.” In: *Open Networking Summit (ONS)*. 2014.

[152] S. Vissicchio, L. Vanbever, and O. Bonaventure. “Opportunities and research challenges of hybrid software defined networks.” In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014), pp. 70–75. DOI: 10.1145/2602204.2602216.

[153] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford. "Central Control Over Distributed Routing." In: *ACM SIGCOMM Computer Communication Review* 45.5 (2015), pp. 43–56. DOI: 10.1145/2829988.2787497.

[154] S. Vissicchio, L. Vanbever, L. Cittadini, G. Xie, and O. Bonaventure. "Safe Update of Hybrid SDN Networks." In: *IEEE/ACM Transactions on Networking (TON)* 25.3 (2017), pp. 1649–1662. DOI: 10.1109/TNET.2016.2642586.

[155] S. Vissicchio, L. Vanbever, and J. Rexford. "Sweet Little Lies: Fake Topologies for Flexible Routing." In: *Proc. of ACM 13th Workshop on Hot Topics in Networks*. ACM. 2014. DOI: 10.1145/2670518.2673868.

[156] H. Wang, Y. Li, D. Jin, P. Hui, and J. Wu. "Saving Energy in Partially Deployed Software Defined Networks." In: *IEEE Transactions on Computers* 65.5 (2016), pp. 1578–1592. DOI: 10.1109/TC.2015.2451662.

[157] L. Wang, Q. Li, Y. Jiang, and J. Wu. "Towards mitigating Link Flooding Attack via incremental SDN deployment." In: *Proc. of IEEE Symposium on Computers and Communications (ISCC)*. 2016, pp. 397–402. DOI: 10.1109/ISCC.2016.7543772.

[158] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image quality assessment: from error visibility to structural similarity." In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[159] R. Weingärtner, G. B. Bräscher, and C. B. Westphall. "Cloud resource management: A survey on forecasting and profiling models." In: *Journal of Network and Computer Applications* 47 (2015), pp. 99–106. DOI: 10.1016/j.jnca.2014.09.018.

[160] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, et al. "SmartSLA: Cost-Sensitive Management of Virtualized Resources for CPU-Bound Database Services." In: *IEEE Transactions on Parallel and Distributed Systems* 26.5 (2015). DOI: 10.1109/TPDS.2014.2319095.

[161] G. Yang, K. Lee, W. Jeong, and C. Yoo. "Flo-v: Low Overhead Network Monitoring Framework in Virtualized Software Defined Networks." In: *Proc. of ACM 11th International Conference on Future Internet Technologies (CFI)*. 2016, pp. 90–94. DOI: 10.1145/2935663.2935677.

[162] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali. "On scalability of software-defined networking." In: *IEEE Communications Magazine* 51.2 (2013), pp. 136–141. DOI: 10.1109/MCOM.2013.6461198.

[163] K. Yeunwoong, N. T. M., H. Kiwon, P. Jongkwan, and P. Jinwoo. “Software Defined Service Migration through Legacy Service Integration into 4G Networks and Future Evolutions.” In: *IEEE Communications Magazine* September (2015), pp. 108–114. DOI: 10.1109/MCOM.2015.7263353.

[164] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. “Scalable flow-based networking with DIFANE.” In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 351–362. DOI: 10.1145/1851275.1851224.

[165] A. Zaalouk and K. Pentikousis. “Network configuration in OpenFlow networks.” In: *Proc. of International Conference on Mobile Networks and Management*. Springer. 2014, pp. 91–104. DOI: 10.1007/978-3-319-16292-8_7.

[166] Y. Zhang, X. Gong, Y. Hu, W. Wang, and X. Que. “SDNMP: Enabling SDN management using traditional NMS.” In: *Proc. of IEEE International Conference on Communication Workshop (ICCW)*. 2015, pp. 357–362. DOI: 10.1109/ICCW.2015.7247205.

Miscellaneous

[167] I. T. U. (ITU). *P.800: Methods for subjective determination of transmission quality*. Published: 30.08.1996. URL: https://www.itu.int/rec/T-REC-P.800-199608-I/en (Last Accessed: 7.11.2018).

[168] C. R. A. Vidal E. Fernandes and M. Salvador. *libfluid*. Published: 25.08.2014. URL: http://opennetworkingfoundation.github.io/libfluid/ (Last Accessed: 7.11.2018).

[169] *Apache kafka*. URL: https://kafka.apache.org/ (Last Accessed: 7.11.2018).

[170] *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. Published: 15.07.2017. 2018. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html (Last Accessed: 7.11.2018).

[171] J. Corbet. *Batch processing of network packets*. Published: 2018-08-21. URL: https://lwn.net/Articles/763056/ (Last Accessed: 7.11.2018).

[172] *crossbar.io*. URL: https://crossbar.io/ (Last Accessed: 7.11.2018).

[173] E. Dumazet. *pkt_sched: fq: Fair Queue packet scheduler*. Published: 24.08.2013. URL: https://lwn.net/Articles/564825/ (Last Accessed: 7.11.2018).

[174] E. Dumazet. *tcp: TCP Small Queues*. Published: 10.07.2012. URL: https://lwn.net/Articles/506237/ (Last Accessed: 7.11.2018).

[175] *etcd*. URL: https://coreos.com/etcd/ (Last Accessed: 7.11.2018).

[176] *EU FP7 ALIEN project*. URL: http://www.fp7-alien.eu/ (Last Accessed: 7.11.2018).

[177] infosim GmbH & Co. KG. *StableNet*. URL: https://www.infosim.net/ (Last Accessed: 7.11.2018).

[178] *Open Compute Project*. URL: https://www.opencompute.org/ (Last Accessed: 7.11.2018).

[179] *OPENCONFIG*. URL: http://www.openconfig.net/ (Last Accessed: 7.11.2018).

[180] *OpenFlow, OF-Conf and Core Information Model (CIM)*. URL: https://www.opennetworking.org/ (Last Accessed: 7.11.2018).

[181] *OpenStack*. URL: https://www.openstack.org (Last Accessed: 7.11.2018).

[182] *Paramiko*. URL: http://www.paramiko.org/ (Last Accessed: 7.11.2018).

[183] *RabbitMQ*. URL: https://www.rabbitmq.com/ (Last Accessed: 7.11.2018).

[184] *Selenium*. URL: https://www.seleniumhq.org/ (Last Accessed: 7.11.2018).

[185] *tc-tbf*. URL: https://www.systutorials.com/docs/linux/man/8-tc-tbf/ (Last Accessed: 7.11.2018).

[186] *The internet topology zoo*. URL: www.topology-zoo.org (Last Accessed: 7.11.2018).

[187] Themezy. *Science Lab Website Template*. URL: https://www.themezy.com/free-website-templates/121-science-lab-free-responsive-website-template (Last Accessed: 7.11.2018).

[188] *TOSCA*. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca (Last Accessed: 7.11.2018).

[189] J. Warner. *packet buffers*. URL: https://people.ucsc.edu/~warner/buffer.html (Last Accessed: 7.11.2018).

[190] *ZooKeeper*. URL: https://zookeeper.apache.org/ (Last Accessed: 7.11.2018).

Acronyms

ACL Access Control List 15, 19, 21, 23

API Application Programming Interface 15, 26, 48, 62, 114

ARP Address Resolution Protocol 19, 20

ASIC Application-Specific Integrated Circuit 1, 8, 54, 55, 67, 153

BGP Border Gateway Protocol 1, 13, 19, 20

CDF Cumulative Distribution Function 75, 137, 138, 139, 138

CLI Command Line Interface 1, 13, 21, 24, 33, 49, 50, 52, 54, 151, 154

CPU Central Processing Unit 1, 6, 8, 67, 81, 82, 84, 86, 88, 89, 90, 91, 93, 97, 100

DASH Dynamic Adaptive Streaming over HTTP 118, 145

DBMS Database Management System 48

DSL Domain Specific Language 6, 44, 49, 60

DUT Device Under Test 32

FIB Forwarding Information Base 19

FIFO First-In First-Out 67

ForCES Forwarding and Control Element Separation 14, 43, 44, 48

FPGA Field-Programmable Gate Array 21

FV FlowVisor 83, 85, 86, 89, 90, 97, 98

HAL Hardware Abstraction Layer 18, 21, 22

HAS HTTP Adaptive Streaming 40

HPP hypervisor placement problem 83

HTML Hyper Text Markup Language 122

HTTP Hypertext Transfer Protocol 1, 21, 36, 37, 40, 49, 115, 117, 118, 121, 122, 133

HTTPS Secure Hypertext Transfer Protocol 39

ID Identifier 85

IGP Interior Gateway Protocol 19, 20

IP Internet Protocol 23, 24, 29, 85, 133

ISP Internet Service Provider 67

KPI Key Performance Indicator 5, 7, 8, 10, 109, 110, 112, 113, 114, 116, 117, 118, 119, 120, 121, 134, 144

LLDP Link Layer Discovery Protocol 86

LTE Long Term Evolution 113

MAC Medium Access Control 20, 133

MIB Management Information Base 47

MILP Mixed-Integer Linear Program 8, 10, 108, 111, 124, 125

MOS Mean Opinion Score 110, 113, 114, 119, 120, 121, 139

MPLS Multiprotocol Label Switching 17, 19, 74

MSE Mean Squared Error 90

NFV Network Function Virtualization 26

NMS Network Management System 1, 6, 15, 17, 18, 22, 23, 26, 50, 54

NP non-deterministic polynomial-time 26, 131

NSAL Network Services Abstraction Layer 2, 3, 4, 5, 6, 8, 9, 10, 13, 16, 22, 26, 29, 35, 36, 38, 40, 43, 44, 45, 46, 47, 48, 49, 50, 49, 50, 51, 52, 53, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 67, 76, 77, 79, 80, 81, 82, 83, 84, 85, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 92, 93, 94, 96, 97, 98, 106, 107, 108, 109, 110, 111, 114, 115, 124, 132, 134, 135, 148, 149, 152, 153, 154, 179, 180, 183

ODR Orthogonal Distance Regression 10, 89, 94, 96

ONF Open Networking Foundation 48

OPEX operational expenditure 26

OSHI Open Source Hybrid IP/SDN 23

OSPF Open Shortest Path Forwarding 1, 13, 19, 24, 25

OVS OpenvSwitch 32, 35, 34, 89, 90

OVX OpenVirtX 83, 85, 86, 89, 91, 98

POF Protocol-Oblivious Forwarding 14

PQ Priority Queuing 51, 55

QoE Quality of Experience 5, 7, 8, 10, 107, 110, 112, 113, 114, 119, 120, 183

QoS Quality of Service 3, 6, 8, 9, 10, 14, 15, 17, 18, 23, 29, 35, 36, 40, 43, 44, 47, 49, 53, 55, 57, 59, 61, 62, 64, 107, 108, 111, 112, 111, 113, 114, 136, 139, 141, 148, 149, 151, 152, 153, 154

RAM Random Access Memory 81, 97

RBF Radial Basis Function 94

RED Random Early Drop 17

REST Representational state transfer 21, 50, 49

RFC Request for Comment 44

RIB Routing Information Base 21

RTP Real-Time Transport Protocol 118

RTT Round Trip Time 113, 140, 141

SDN Software-Defined Networking 2, 3, 4, 5, 6, 8, 9, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 29, 31, 32, 33, 34, 40, 43, 44, 49, 57, 59, 58, 60, 62, 63, 64, 65, 66, 67, 69, 70, 71, 72, 73, 74, 73, 74, 75, 76, 75, 76, 77, 83, 114, 133, 134, 135, 151, 152, 153, 154

SNMP Simple Network Management Protocol 13, 14, 15, 21, 23, 24, 47, 52

SSH Secure Shell 117

SSIM Structural Similarity Index 112, 114

STP Spanning Tree Protocol 19, 20

SVM Support Vector Machine 6, 10, 93, 94, 102, 105, 106, 154

TB Token Bucket 37, 39

TCP Transmission Control Protocol 6, 9, 30, 36, 37, 38, 40, 67, 117, 122, 134, 137, 138, 148, 149, 152

UDP User Datagram Protocol 32, 36, 118, 138

vCPU Virtual Central Processing Unit 84

VLAN Virtual Local Area Network 1, 5, 9, 15, 16, 17, 19, 20, 21, 22, 29, 31, 33, 34, 40, 43, 49, 50, 56, 57, 58, 61, 60, 67, 77, 152

VNE Virtual Network Embedding 131

VoIP Voice-over-IP 107, 116, 118, 121, 140, 141, 149

WFQ Weighted Fair Queuing 17, 54, 55

List of Figures

List of Tables